Final Exam
CS161 Computer Security, Spring 2008

Name __________________________
Email address __________________________
Score ______

Please do not open this exam until you are instructed to begin work. Until then, you may read these instructions and fill in your name and email address in the provided blanks.

This exam consists of some multiple choice questions and some short answer questions. The questions may or may not be of equal difficulty, so use your time judiciously.

For each multiple choice question, circle the letter of your selected answer. Please circle only one answer. Be careful – some of the questions are subtle. Think through each of the possible answer choices carefully. Feel free to use the margins or page backs to make notes or calculations, but partial credit will not be given on the multiple choice questions.

For each short answer question, write your answer in the space beneath the question. If you need more space, you can use the back of another page (being careful to label it with the number of corresponding question).

If you have questions during the exam, please raise your hand and a TA will come to you.
OpenSSL key generation vulnerability

The following is a real security advisory from earlier this week. The first half of the exam is based on exploiting this vulnerability and potential defenses.

openssl vulnerability CVE-2008-0166

A weakness has been discovered in the random number generator used by OpenSSL on Debian and Ubuntu systems. As a result of this weakness, certain encryption keys are much more common than they should be, such that an attacker could guess the key through a brute-force attack given minimal knowledge of the system. This particularly affects the use of encryption keys in OpenSSH, OpenVPN and OpenSSL certificates.

This vulnerability only affects operating systems which (like Ubuntu) are based on Debian....

== Who is affected ==
Systems which are running any of the following releases:
* Ubuntu 7.10 (Gutsy)
* Ubuntu 8.04 LTS (Hardy)
* Ubuntu "Intrepid Ibex" (development): libssl <= 0.9.8g-8
* Debian 4.0 (etch) (see corresponding Debian security advisory)

and have openssh-server installed or have been used to create an OpenSSH key or X.509 (OpenSSL) certificate.
All OpenSSH and X.509 keys generated on such systems must be considered untrustworthy, regardless of the system on which they are used, even after the update has been applied.
This includes the automatically generated host keys used by OpenSSH, which are the basis for its server spoofing and man-in-the-middle protection.
For the purposes of this exam, you can assume that the attacker is able to guess the weak keys mentioned this advisory. Also, note that X.509 is the certificate format used by HTTPS, and other OpenSSL-based protocols, such as IMAP tunneled over OpenSSL or TLS.

1. **The OpenSSL Vulnerability**

1.1, 5 points

There is a worm outbreak using OpenSSL. The worm tries to login to servers by trying keys and usernames. Which of the following schemes would be the simplest and most effective for detecting whether a host is trying to spread the worm.

(a) Seeing too many failed SSH connection attempts from one host.

(b) Tarpit.

(c) Signature based filtering, where the signature is formed by finding common substrings in network traffic, as mentioned in lecture.

(d) Signature based filtering, where the signature is formed by taint tracking and symbolic execution, as mentioned in lecture.

1.2, 5 points

The attacker has compromised some machines in your network, but you don’t know how many are affected. (Some of the compromised machines are being actively used.) Which of the following is sufficient to prevent the attacker from compromising more machines?

(a) Update OpenSSL to a version that generates strong keys.

(b) Reset all passwords and change all keys on the network.

(c) Format the hard drives of all Debian/Ubuntu based systems.

(d) None of the above.
1.3, 5 points
Consider two varieties of worm spreading via SSH, using the OpenSSL key generation vulnerability.

- Variety A: The worm operates exclusively by random spread, guessing usernames and keys.
- Variety B: The worm will use local information to identify vulnerable hosts whenever possible, from the known_hosts, private keys and bash history. When local information is not available, or when the local information has been exhausted, the worm resorts to random spread, guessing usernames and keys.

Compared to using purely random spread as in variety A, which of the following precisely characterizes the effect of using local information, as in variety B?

(a) Increases the contact rate, $\beta$
(b) Decreases $\beta$
(c) Shifts the effective starting time of the infection, so $I'(t) = I(t+C)$ for some constant $C$.
(d) Decreases the propagation rate.
(e) None of the above

1.4
Recall the firewall rule syntax we learned in class. Example rules:

```
allow * *:*/in -> *:*/out
allow TCP 1.2.3.4:567/in -> 5.6.7.8:910/out
drop * *:* -> *.*
```

Consider a corporate network with a publically accessible web server at IP 1.2.3.1, and SSH server for employees to login from home, at 1.2.3.2. All other internal IPs correspond to employee workstations.

(a) (5 points) Write down firewall rules to accomplish the following security policy:
(a) By default, block all inbound connections.

(b) Allow all inbound TCP connections to HTTP (port 80) on the web server.

(c) Allow all inbound TCP connections to SSH (port 22) on the SSH server.

(d) Allow all outbound connections.

(b) (2 points) In order to defend against the worm, we could add a security policy to disallow all inbound and outbound SSH connections. Modify your firewall ruleset accordingly.

(c) (5 points) A computer on your internal network was already infected with an OpenSSL key generation based worm. Given that the firewall rules in part b are in effect, can it still infect any machines? Which ones?

1.5
Consider a worm that spreads by exploiting the weak SSH keys caused by the OpenSSL key generation vulnerability. It has already obtained a valid username and private key. Assume that, upon logging into the machine, the worm sends a fixed sequence of shell commands from the client to the server, over the established SSH connection.
In this problem, consider building an IDS to detect the worm. This sequence of shell commands is encrypted, so the IDS can’t simply look for the sequence. Below are two different methods the IDS could use to detect the worm, despite the encryption. Method a: The IDS analyzes the number, size and timing of packets sent to the SSH server. Method b: The IDS is host based, and resides inside the SSH server, examining the incoming bits after they have been decrypted, to see if they match the fixed sequence of shell commands.

(a) (8 points) State the conditions under which method a is better than b, and then state the conditions under which b is better than a.

(b) (6 points) For each of the above two methods, how could the worm evade the IDS?

2.

Consider the following code. filename is guaranteed to be a well-formed string; “/tmp/foo/” exists, and is a directory. is_normal_file() tests that the file is an ordinary file, rather than a symlink or device.

```c
string filename = getStringFromHTTPRequest("department");
string tmpdir = "/tmp/foo/";
string full_name = tmpdir + filename;
```

// (*)
if(is_normal_file(full_name)) {
    sendFileToWebBrowser(full_name);
}

If filename contains "..", then this code can be tricked into reading files outside
of "/tmp/foo". One way to address this problem is to insert code at (*) to check
that the filename does not contain "..". However, this approach has a few problems.
For example, a new version of your operating system adds more special directories
and files, such as "...", which means "../..". Also, some users legitimately need
to provide paths that reference the ".." directory, but still refer to a file inside of
"/tmp/foo" (for example, users want to pass in "bar/baz/..\index.html", which will
resolve to "/tmp/foo/bar/index.html").

(10 points) Write in a few sentences how the input validation should be done to
ensure the security property.

3.

Consider the following code.

char buf[42];
int array[100];

int a = read_int_from_keyboard();
int b = read_int_from_keyboard();
int c = read_int_from_keyboard();

int d = 100;
if (a>5) //Branch 1
if (b==12) //Branch 2
  if (c <= 42) //Branch 3
  {
    d *= b;
    for(int i = 0; i < c; i++)
    {
      // Insert assertion 1 here
      buf[i] = read_char_from_keyboard();
    }
    // Prove safety here
  }
if (b <= 7) //Branch 4
{
  for(int i = 0; i < c; i++)
  {
    //Insert assertion 2 here
    buf[i] = read_char_from_network();
  }
  for(int i = 0; i < d; i++)
  {
    //Insert assertion 3 here
    buf[i] = read_char_from_network();
  }
}

(a) (3 points) Write, inline with the code, the 3 assertions that must hold
if we are to avoid buffer overflows.

(b) (6 points) For each assertion, write out the symbolic expression, in
terms of inputs a, b, and c, that, if satisfied, would lead to a buffer
overflow.
(c) (10 points) Write down preconditions, postconditions, and loop invariants for this code. Then, use them to prove that, if the program reaches “d *= b;” then it will reach “Prove safety here” without overflowing a buffer or failing the assertion you added, up to this point.

4.

The version of software fault isolation we discussed in lecture only applied to RISC architectures, partially because it assumed that instructions are fixed length. On CISC architectures, instructions are variable length, and it is possible to create a valid instruction, say, 0xBADC AFEF OODD EADB EEF0. Say the instruction “DEAD BEEF” causes the program to jump to an illegal address. If the attacker could get the processor to jump to beginning of the word “DEAD”, then the attacker could subvert SFI. A partial solution to this problem is to break memory up into chunks, (sized larger than the longest instruction in the architecture), and arrange for the beginning of each chunk to be a valid (legal) instruction. We can use nop
padding to move instructions to the beginning of chunks.

Assume that instructions range from 3-8 bytes, and that chunks are 16 bytes long. All valid chunks are stored in memory addresses beginning with 0800. Assume that the value 0800 is stored in a dedicated register called %r.

To ensure security, we would like to guarantee that all jumps go to addresses started with 0800. To achieve this, we’ll need to insert some checking code before each jump and make sure that this inserted check will not be bypassed (similar to what is described in SFI).

(a) (5 points) Fill in the blank in a few lines of pseudo code to describe what the inserted checking code should be. (hint: one way to ensure that the inserted checks are not bypassed is to require all jumps go to the beginning of a chunk.)

```
%jumptarget = ... //sample valid value: 0x 0800 A23F 1594 1020
// <--- ensure that %jumptarget points to the start of a valid chunk.
jump (%jumptarget);
```

(b) (5 points) What additional property must be satisfied for the instrumentation, to satisfy the security property that each jump goes to addresses starting with 0800.

5.

In class, we studied a NIDS complexity attack against a hash table used to index IP addresses. In an attempt to cure this problem, we could replace the hash table with a binary tree.

Here is the pseudocode code for the binary tree insertion.
void insert(Tree t, IP_ADDRESS ip) {
    if (t.ip == NULL) {
        t.ip = ip;
        t.left = new Tree;
        t.right = new Tree;
        return;
    }
    if (ip >= t.ip)
        insert(t.right, ip);
    if (ip < t.ip)
        insert(t.left, ip)
}

(a) (5 points) Unfortunately, there exists a complexity attack against this NIDS. Describe in one or two sentences what the attacker would do to carry out this attack.

(b) (5 points) Describe at a high level how we should modify this NIDS to avoid this attack. Your modified NIDS should still use a binary tree to store IP addresses. Your answer should be one or two sentences long.