EECE 412, Fall 2010

Quiz #4

This quiz consists of 7 pages. Please check that you have a complete copy. You may use both sides of each sheet if needed.

Your Family name: ________________

Your Given name: ________________

Your student ID: ________________

#	Points	Out of
1		4
2		6
3		8
4		2
TOTAL		20

Name of your left neighbor: ________________

Name of your right neighbor: ________________

ATTENTION: When necessary, make reasonable assumptions and state them clearly in your solutions.

1. **The results of the security analysis presented by Prof. Savage during the guest lecture on November 4th suggest that the analyzed automobile was vulnerable to which of the following ways of the adversary obtaining unauthorized access to the automobile's controller area network (CAN) (mark all applicable):**

   a. packet sniffing

   b. targeted probing

   c. fuzzing

   d. reverse engineering

   e. bridging internal CAN networks

   Explain your answer:

None of the above. The author assumed the adversary could gain unauthorized access to CAN, but did not discuss how.

2. “At approximately 5 PM on November 2, 1988 the ‘Morris Worm’ was started at the MIT AI laboratory in Cambridge, Massachusetts. It quickly spread to Cornell, Stanford, and then on to other sites. By the next morning, almost the entire Internet was infected. This was the first, great Internet Panic.”
In the appendix of this quiz, you can find a fragment from “A Report on the Internet Worm.” The malware is also known as “Morris Worm” named after its author Robert Morris Junior, who is now a faculty member at MIT.

Explain how the worm did the following functions:

1. Reconnaissance

“It does a 'netstat -r -n' to find local routes to other hosts & networks, looks in /etc/hosts, and uses the yellow pages distributed hosts file if it's available.”
“Once it finds a local network (like 129.63.nn.nn for ulowell) it sequentially tries every address in that range.”

2. Attack

“There are three ways it attacks: sendmail, fingerd, and rsh/rexec.”

3. Communication

rexec, TCP

4. Command

In the case of sendmail or fingerd attack vector, the attacking host uses raw TCP/UDP/IP communications with the bootstrap program on the attacked host to upload the rest of the worm.
In the case of rsh/rexec vector, the attacking host uses rsh/rexec communications with the attacked host to upload the worm.
To avoid concurrent attempts of attacking same host from different infected hosts, each infection attempt starts with marking the victim host with “telnetd: ttloop: peer died” message in the /usr/adm/messages log.

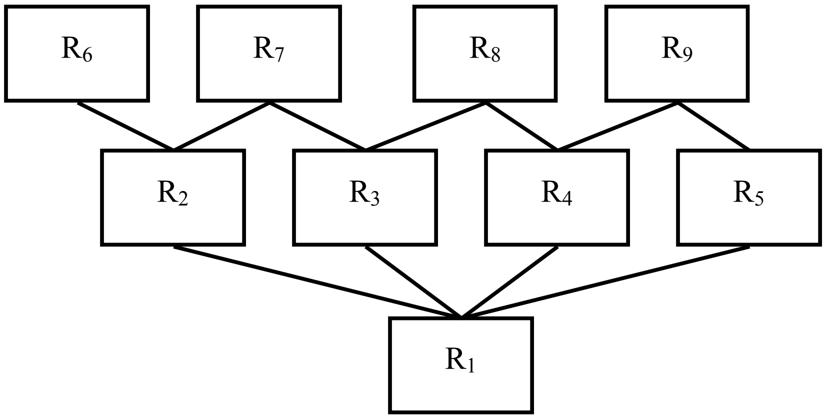
5. Intelligence

“Each time the worm is started, there is a 1/15 chance (it calls random()) that it sends a single byte to ernie.berkeley.edu on some magic port, apparently to act as some kind of monitoring mechanism.”

3. Access control. For the following two policies, determine whether they are equivalent (i.e., users have same permissions in both policies). Explain your answer.

a. Hierarchical RBAC policy

Role hierarchy (RH):



Permission-to-role assignment (PA):

permssn / role	O_1	O_2	O_3	O_4
R_1	r			
R_2	w	r		
R_3		w	r	
R_4			w	r
R_5				w
R_6				
R_7				
R_8				
R_9				

User-to-role assignment (UA):

user / role	U_1	U_2	U_3	U_4
R_1			X	
R_2		X		
R_3	X			
R_4				
R_5				
R_6				X
R_7			X	
R_8		X		
R_9	X			

b. Policy expressed through the following access matrix:

object / user	O_1	O_2	O_3	O_4
U_1	r	w	r, w	r, w
U_2	r, w	r, w	r, w	r
U_3	r, w	r, w	r	-----
U_4	r, w	r	-----	-----

The above two policies are equivalent: [✓ Yes], ____No

Because …

Access matrices for the above policies are identical.

U1 has roles of R1, R3, R4, R5, R9.
U2 has roles of R1, R3, R4, R8.
U3 has roles of R1, R2, R3, R7.
U4 has roles of R1, R2, R6.

Based on Permission to Role assignment above, the resulted access matrix would be identical to the access matrix expressed in b.

4. Which assignment(s) can be removed from the UA in the previous without any user loosing any permission(s)?

U3 to R1 assignment can be removed from UA, because U3 is already assigned to R7; and R7 is the super role of R1, which has all the permissions R1 has.

Appendix (You are welcome to detach this appendix and take it home with you)
The original is from http://www.morrisworm.com/page_worm.txt

A REPORT ON THE INTERNET WORM

Bob Page
University of Lowell
Computer Science Department

November 7, 1988

...
The basic object of the worm is to get a shell on another machine so it can reproduce further. There are three ways it attacks: sendmail, fingerd, and rsh/rexec.

THE SENDMAIL ATTACK:

In the sendmail attack, the worm opens a TCP connection to another machine's sendmail (the SMTP port), invokes debug mode, and sends a RCPT TO that requests its data be piped through a shell. That data, a shell script (first-stage bootstrap) creates a temporary second-stage bootstrap file called x$$,l1.c (where '$$' is the current process ID). This is a small (40-line) C program.

The first-stage bootstrap compiles this program with the local cc and executes it with arguments giving the Internet hostid/socket/password of where it just came from. The second-stage bootstrap (the compiled C program) sucks over two object files, x$$,vax.o and x$$,sun3.o from the attacking host. It has an array for 20 file names (presumably for 20 different machines), but only two (vax and sun) were compiled in to this code. It then figures out whether it's running under BSD or SunOS and links the appropriate file against the C library to produce an executable program called /usr/tmp/sh - so it looks like the Bourne shell to anyone who looked there.

THE FINGERD ATTACK:

In the fingerd attack, it tries to infiltrate systems via a bug in fingerd, the finger daemon. Apparently this is where most of its success was (not in sendmail, as was originally reported). When fingerd is connected to, it reads its arguments from a pipe, but doesn't limit how much it reads. If it reads more than the internal 512-byte buffer allowed, it writes past the end of its stack. After the stack is a command to be executed ("/usr/ucb/finger") that actually does the work. On a VAX, the worm knew how much further from the stack it had to clobber to get to this command, which it replaced with the command "/bin/sh" (the bourne shell). So instead of the finger command being executed, a shell was started with no arguments. Since this is run in the context of the finger daemon, stdin and stdout are connected to the network socket, and all the files were sucked over just like the shell that sendmail provided.

THE RSH/REXEC ATTACK:

The third way it tried to get into systems was via the .rhosts and /etc/hosts.equiv files to determine 'trusted' hosts where it might be able to migrate to. To use the .rhosts feature, it needed to actually get into people's accounts - since the worm was not running as root (it was running as daemon) it had to figure out people's passwords. To do this, it went through the /etc/passwd file, trying to guess passwords. It tried combinations of: the username, the last, first, last+first, nick names (from the GECOS field), and a list of special "popular" passwords:

...

When everything else fails, it opens /usr/dict/words and tries every word in the dictionary. It is pretty successful in finding passwords, as most people don't choose them very well. Once it gets into someone's account, it looks for a .rhosts file and does an 'rsh' and/or 'rexec' to another host, it sucks over the necessary files into /usr/tmp and runs /usr/tmp/sh to start all over again.

Between these three methods of attack (sendmail, fingerd, .rhosts) it was able to spread very quickly.

THE WORM ITSELF:

The 'sh' program is the actual worm. When it starts up it clobbers its argv array so a 'ps' will not show its name. It opens all its necessary files, then unlinks (deletes) them so they can't be found (since it has them open, however, it can still access the contents). It then tries to infect as many other hosts as possible - when it successfully connects to one host, it forks a child to continue the infection while the parent keeps on trying new hosts.

One of the things it does before it attacks a host is connect to the telnet port and immediately close it. Thus, "telnetd: ttloop: peer died" in /usr/adm/messages means the worm attempted an attack.

The worm's role in life is to reproduce - nothing more. To do that it needs to find other hosts. It does a 'netstat -r -n' to find local routes to other hosts & networks, looks in /etc/hosts, and uses the yellow pages distributed hosts file if it's available. Any time it finds a host, it tries to infect it through one of the three methods, see above. Once it finds a local network (like 129.63.nn.nn for ulowell) it sequentially tries every address in that range.

If the system crashes or is rebooted, most system boot procedures clear /tmp and /usr/tmp as a matter of course, erasing any evidence. However, sendmail log files show mail coming in from user /dev/null for user /bin/sed, which is a tipoff that the worm entered.

Each time the worm is started, there is a 1/15 chance (it calls random()) that it sends a single byte to ernie.berkeley.edu on some magic port, apparently to act as some kind of monitoring mechanism.