EECE 412, Fall 2007

Quiz #3

This quiz consists of 8 **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		10
2		10
3		10
4		5
TOTAL		

Name of your left neighbor: _____

Name of your right neighbor: _____

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

1. Suppose all passwords on a given system are 8 characters long and that each character can have any one of 32 different values. The passwords are hashed (with salt) and stored in a password file. Now suppose Trudy has a password cracking program that can test 2^{19} passwords per second. Trudy also has a dictionary of 2^{20} common passwords and the probability that any given password in her dictionary is 1/4. The password file on this system contains 512 password hashes.
   a. **(1 point) How many different passwords are possible given the above length of passwords and the number of possible values of each character?**

      Sample answer: 32^8

   b. **(7 points) How long on average, will it take Trudy to "crack" the password for the administrator's account (assume there is only one such account)? Explain your answer.**

      Sample answer: Trying on average half the dictionary, before the matching password is found, would take Trudy 2^{19} password tests. In the 1/4 of cases, Trudy would succeed just by using a dictionary. In the rest, 3/4, of the cases, Trudy would have to resort to the brute force search, which would require, on average, $32^8/2 = 2^{39}$ password tests, making the total number of tests $(2^{19} + 3*2^{39})/4$. The first factor is 2^{20} times smaller so it can be ignored. The amount of time it would take Trudy's program to crack the password, would be $3*2^{39} / 4*2^{19} = 3*2^{18}$ seconds, which is little bit over 9 days.

   c. **(2 points) What is the probability that at least one of the 512 passwords in the password file is in the dictionary? Explain your answer.**

      Sample answer: the probability is $1- (3/4)^{512} \approx 1$. We compute it by first computing the probability that none of the passwords is in the dictionary ($(3/4)^{512}$), and then finding the difference between this value and 1.

2. On October 26th, 2007, Apple released a new version of Mac OS X. Also known as Leopard, this new version of Mac OS X has over 300 new features and is considered as the biggest ever upgrade to Mac OS X. Appendix A contains a reproduction of new security-related features. After quickly reading through the features.
   a. **Explain which of these new features support, enable, or follow which of the principles for designing secure systems.**

	Tagging Downloaded Applications	Signed Applications	Application-Based Firewall	Stronger Encryption for Disk Images	Enhanced VPN Client Compatibility	Sharing and Collaboration Configuration	Sandboxing	Multiple User Certificates	Enhanced Smart Card Capabilities	Library Randomization	Windows SMB Packet Signing
Least Privilege			√			√	√				
Fail-Safe Defaults									√	√	
Economy of Mechanism										√	
Complete Mediation	√					√	√				
Open Design	√	√	√	√	√		√		√		√
Separation of Privilege											
Least Common Mechanism					√						
Psychological Acceptability					√	√		√	√		√
Defense in depth	√	√	√	√			√		√	√	√
Question assumptions	√	√		√				√			

   b. **Explain which particular aspects of malware and corresponding protection and detection techniques are used in these new features.**

	Tagging Downloaded Applications	Signed Applications	Application-Based Firewall	Stronger Encryption for Disk Images	Enhanced VPN Client Compatibility	Sharing and Collaboration Configuration	Sandboxing	Multiple User Certificates	Enhanced Smart Card Capabilities	Library Randomization	Windows SMB Packet Signing
Acting Both as Data and Code	√									√	√
Uses Privileges of Authorized Users	√		√				√				
Uses Sharing to Cross Protection Domain Boundaries		√	√								
Alters Files		√								√	√
Performs Actions Beyond Specification			√				√				
Alters Statistical Characteristics		√									√

3. "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."
At the end of this handout (Appendix B), 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:

a. 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."

b. Attack

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

c. Communication

rexec, TCP

d. 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.

e. **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."

4. Bonus question: Same setup as in problem 1. **What is the *expected* work for Trudy to recover any one of the passwords in the password file? Provide the expected work in either number of hash operations or in the terms of time that it would take Trudy to crack any one of the passwords.**

   Sample answer: $(2^{19})/4 + (3/4)(2^{20}+2^{19})/4 + (3/4)^2(2*2^{20}+2^{19})/4 + \ldots + (3/4)^{512}(512*2^{20}+2^{19})/4 < 2^{22}$, which would take Trudy's program just 8 seconds.

   See end of Section 7.3.5 of Stamp's book for an explanation of how to derive the formula.

Appendix A (You are welcome to detach this appendix and take it home with you)
Reproduced from http://www.apple.com/macosx/features/300.html#security

Feel free to refer to the features just by the following numbers.

1. **Tagging Downloaded Applications:** Protect yourself from potential threats. Any application downloaded to your Mac is tagged. Before it runs for the first time, the system asks for your consent — telling you when it was downloaded, what application was used to download it, and, if applicable, what URL it came from.

2. **Signed Applications:** Feel safe with your applications. A digital signature on an application verifies its identity and ensures its integrity. All applications shipped with Leopard are signed by Apple, and third-party software developers can also sign their applications.

3. **Application-Based Firewall:** Gain more control over the built-in firewall. Specify the behavior of specific applications to either allow or block incoming connections.

4. **Stronger Encryption for Disk Images:** Give your data even more security. Disk Utility now allows you to create encrypted disk images using 256-bit AES encryption.

5. **Enhanced VPN Client Compatibility:** Connect to a broader range of VPN clients. Leopard supports Cisco Group Filtering as well as DHCP over PPP, which allows you to dynamically acquire additional configuration options such as static routes and search domains.

6. **Sharing and Collaboration Configuration:** Share any folder on your Mac by setting it up as a shared folder in the Get Info window or in the Sharing pane of System Preferences. You can also create and edit access control lists, share with individuals in your network directory, or contacts in Address Book.

7. **Sandboxing:** Enjoy a higher level of protection. Sandboxing prevents hackers from hijacking applications to run their own code by making sure applications only do what they're intended to do. It restricts an application's file access, network access, and ability to launch other applications. Many Leopard applications — such as Bonjour, Quick Look, and the Spotlight indexer — are sandboxed so hackers can't exploit them.

8. **Multiple User Certificates:** Have more flexibility in choosing a digital certificate for encrypting email messages. With support for multiple user certificates, you can use the Keychain application to associate your certificates with various email addresses.

9. **Enhanced Smart Card Capabilities:** Let your smart card do more. Now you can use a smart card to unlock FileVault volumes and your keychain, and configure your Mac to lock the screen when a smart card is removed. Leopard supports the PIV standard for Federal employees and contractors.

10. **Library Randomization:** Defend against attackers with no effort at all. One of the most common security breaches occurs when a hacker's code calls a known memory address to have a system function execute malicious code. Leopard frustrates this plan by relocating system libraries to one of several thousand possible randomly assigned addresses.

11. **Windows SMB Packet Signing:** Enjoy improved compatibility and security with Windows-based servers.

Appendix B (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.