Name:

# Information Assurance: Homework 7 Answers and comments

Due November 12, 2010. No late handins, so we may post the answer key in time to help students study for exam 2.

1. Consider the recently posted tool, Firesheep http://codebutler.com/firesheep.

a. By using firesheep, are you attacking integrity, confidentiality, and/or availability? How?

*You are attacking confidentiality primarily. By capturing your target's login cookie, you can navigate the target web site as them. You can see all their private information.*
*You are also potentially attacking integrity. With position of their cookie, you can produce information as them which could violate data and identity integrity. For example if you grab the target's facebook cookie, you could send messages to the target's friends as the target.*

b. Consider the ethical implications associated with developing and deploying such a tool. Use the ACM/IEEE software engineering Code of Ethics as you reference point. http://www.acm.org/about/se-code. Make an argument that the developer and deployer of Firesheep has been acting ethically or unethically.

*People argue that the developer of firesheep was acting unethically because he is putting a strong attack in the hands of more people. The attack has been well known for some time, but it was limited to people who has the skill of putting together a tool like firesheep or something a bit more primitive. By increasing the pool of attackers, the firesheep developer is making the environment more dangerous.*

*The firesheep developer and his supporters argue that he is acting ethically by drawing attention to the sidejacking attack. This has been a known attack for years. For years people have been trying to educate major web site providers that end-to-end confidentiality is essential, but without an immediate, obvious threat no one had been responding.*

*The public section of the Software Engineering Code of Ethics can be used to support both arguments. First for the case against the firesheep developer. Items 1.02 and 1.03 seem to argue against the development and deployment of a tool like firesheep. Enabling more people to do sidejacking attacks does not seem to like up with the public good, and it is obvious that directly speaking firesheep is not safe and it will diminish the quality of life for some targets of the tool*

***1.02 Moderate the interests of the software engineer, the employer, the client and the users with the public good.***

Name:

***1.03. Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.***

*But a number of the other items in section 1 could be used to argue the case for the firesheep developer.*

***1.04. Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents.***

*The firesheep developer and others have been trying to educate the public about the dangers and the ease of a sidejacking attack, but they have not been getting much attention. By providing firesheep, they have successfully educated many more people about the danger*

***1.06. Be fair and avoid deception in all statements, particularly public ones, concerning software or related documents, methods and tools.***

*The developer did not try to pass off firesheep as an innocuous tool. He was very up front that firesheep is an attack tool.*

***1.08. Be encouraged to volunteer professional skills to good causes and contribute to public education concerning the discipline.***

*The firesheep effort was all about public education.*

2. The text describes Stealth viruses that use root kit techniques to hide information about files containing the virus.

a. Describe how a virus would use root kit function hooking techniques to hide its presence from anti-virus software.

*If the virus had the ability to install some file system hooks, it could filter the results of file system calls to hide information about itself. This could involve filtering the results of "stat" calls that return the size of files. The file system call hook could look for stat being called on infected files and "correct" the file size before returning. The file system call hook could also filter results of read to remove evidence of the virus from read results. The attacker would have to be careful here though and understand which calls the OS uses to access executable files to load them for execution. Obviously you don't want to remove evidence of the virus before execution. Otherwise, you have just filtered your virus out of existence.*

b. What is one way the owner of the machine could detect the presence of such a stealth virus?

*Compare the results of multiple different system and library calls that access the same information (e.g., read and fread). This is how most rootkit scanners work. Either*

Name:

*the functions are not hooked consistently or the information must leak through certain interfaces (for example to enable reading for execution).*

*Another mount the target disk from a different operating system and perform the scan. It seems unlikely that both operating systems would be infected, so the second unhooked OS, should see evidence of the virus.*

3. Consider a program posted at http://www.cs.illinois.edu/class/fa10/cs461/hw7.zip This simple program has not one but two buffer overflow vulnerabilities in two different functions. The program takes two arguments: -f1 or -f2 to indicate which function to invoke and a count of the number of bytes to allocate for a buffer.
    a. The zip file includes a Makefile which will create two binaries (tested on remlnx.ews.uiuc.edu). The hw8-plain binary is a vanilla gcc compile. Try this program on both version of the function with buffer sizes 100 and 200. What happens?

    *Successful operation for 100 passed to both function 1 and function 2.*

    *Segmentation fault for 200 passed to function 1. Infinite loop for 200 passed to function 2.*

    b. The other binary creates hw8-protected which is the same program compiled with the stack-guard canary values (details of stack guard at http://www.usenix.org/publications/library/proceedings/sec98/full_papers/cowan/cowan.pdf). On the ews machines stack guard is not enabled by default, but this is not the case on all distributions. If you compile on some other machine, check your compiler's man page to determine whether you need to enable stack guard in this part or disable it in part a. Run the same experiments again. What happens this time?

    *Again, successful operation for 100 passed to both function 1 and function 2.8*

    *Passing 200 to function 1 and 2 resulted in the same message:*
    *Call for-loop function*
    **** stack smashing detected ***: ./hw8-protected terminated*
    *Abort*

    c. Libsafe uses a runtime approach to detect and protect against stack smashing. The man page is posted at http://www.cs.uiuc.edu/class/fa07/cs461/libsafe.8.html. Unfortunately, this library doesn't seem to work any more. Based on the man page, how do you think hw-plain would operate in the presence of libsafe?

Name:

*Libsafe would have detected the buffer overflow in function 1 which calls one of the functions that libsafe knows about and intercepts. The behavior of function 2 would be unchanged.*

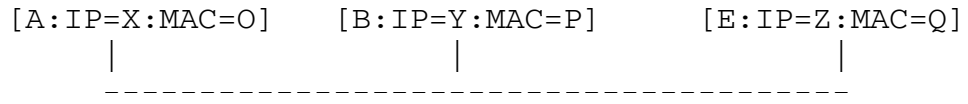
d. What is one unique benefit of each approach (stack guard and libsafe)?

*Stack-guard protects all code that could suffer from buffer over flow either through library code or unique customer code.*

*Libsafe does not need access to the source code to be effective.*

4. Consider ARP cache poisoning.
   a. Describe how poisoning a computer's ARP cache enables you to launch a Man-in-the-middle attack on that computer.

   Consider the following scenario




   *A wants to communicate with B. A and B are on the same network, so A needs B's MAC address to send a packet to B. If E can insert IP:Y → MAC:Q in A's ARP cache, then all packets that A sends to B's IP address will go to E. Then E can forward the traffic to B. If E poison's B's ARP cache so it has the entry IP:X → MAC:Q, the return traffic will also be sent first to E. Then E could send the packet onto A.*
   *All traffic between A and B will flow through E. E sees all the conversation. E could change the traffic or just observe.*

   b. Suppose there is a tool that proposes to automatically protect a computer from ARP cache poisoning. It looks for rapid changes of mappings between MAC addresses and IP addresses. If it sees more than three different MAC addresses associated with an IP address within a configurable period of time (defaults to 30 seconds), it will block the IP address from the ARP cache until the system administrator can investigate. If you know a computer has this tool installed, how can you launch a denial of service attack on this computer?

   *The attacker could spuriously send different MAC mappings for each IP request. This will cause the ARP spoof detection to trigger and halt communication to the target IP. The attacker could do this for all ARP'ed for IP addresses and effective stop it's victim from communicating.*