CS 361S - Network Security and Privacy
Fall 2015

Homework #2

Due: 12:30pm CST (in class), November 17, 2015

YOUR NAME: ________________________________

Collaboration policy

No collaboration is permitted on this assignment. Any cheating (e.g., submitting another person’s work as your own, or permitting your work to be copied) will automatically result in a failing grade. The Department of Computer Science code of conduct can be found at https://www.cs.utexas.edu/academics/conduct.

Late submission policy

This homework is due at the beginning of class on November 17. All late submissions will be subject to the following policy.

You start the semester with a credit of 3 late days. For the purpose of counting late days, a “day” is 24 hours starting at 2pm on the assignment’s due date. Partial days are rounded up to the next full day. You are free to divide your late days among the take-home assignments (3 homeworks and 2 projects) any way you want: submit three assignments 1 day late, submit one assignment 3 days late, etc. After your 3 days are used up, no late submissions will be accepted and you will automatically receive 0 points for each late assignment.

You may submit late assignments to Oliver Jensen (GDC 6.818A—place it on my desk if I am not there). If you are submitting late, please indicate how many late days you are using.

Write the number of late days you are using: ______
Homework #2: Return to Molvania (50 points)

Problem 1

Koobecaf is Molvania’s most popular online social network. It allows other websites to include an iframe, linked to Koobecaf, with JavaScript code that generates a “Hate” button. If a visitor is logged into Koobecaf, he can click on the button to express his disapproval of the page or its contents. This disapproval is then announced on the user’s news feed on the Koobecaf site.

Problem 1a (4 points)

Why is the “Hate” button implemented in an iframe, rather than within the enclosing page itself?

Problem 1b (4 points)

Is clickjacking a concern for the “Hate” button? What is the worst attack you can think of that is enabled by clickjacking these buttons?

Problem 1c (4 points)

Should the JavaScript code making up the “Hate” button include framebusting code to prevent clickjacking attacks?
Problem 2 (4 points)

The Molvanian Institute for Advanced Cryptology developed a new method for protecting network communications. Their idea is to use *Wide-Area Authenticated, Confidential Keys (WACK)* to guarantee authentication, confidentiality, and integrity for every network packet.

Assume that the endpoints of the connection already share key $K$, their secret WACK. Let $p$ be the plaintext packet. The sender encrypts $p$ with $K$ using the AES block cipher in counter mode, appends SHA-1$(K, p)$ to guarantee integrity and authentication, and sends the resulting tuple over the insecure network.

What, if anything, is wrong with this method?

Problem 3

All Molvanian C compilers for x86 insert stack canaries into generated code to prevent stack-overflow attacks. Nevertheless, Molvanian Cyber-Security Bureau mandates the use of *libsafe* with all executables compiled from C.

Problem 3a (4 points)

What additional protections are gained by using *libsafe* with stack-canary-equipped executables?
Problem 3b (5 points)

Give a short snippet of C code that contains a single call to a *libsaf* -protected *strcpy*, and yet is vulnerable to a buffer overflow attack as a result of this call. Your attack must also bypass compiler-inserted stack canaries.

Problem 4

x68 is Molvanian homegrown chip architecture. Unlike on x86, the stack on x68 grows upwards (that is, if *foo()* calls *bar()* , then *bar()*’s stackframe is at a higher memory address than *foo()*’s).

Problem 4a (4 points)

How does a stack-based overflow attack work on x68?

Problem 4b (4 points)

How would you implement StackGuard on x68? What would be the main differences from x86?
Problem 4c (4 points)
How would you implement libsafe on x68? What would be the main differences from x86?

Problem 5
Molvanian Security Defense Operating System (MS-DOS) protects against memory corruption exploits as follows. MS-DOS ensures that heap, stack, and code sections of the program are all page-aligned. On the 32-bit x86 machine (the only kind they use in Molvanía), each page is 4096 bytes. MS-DOS randomizes memory layout using a modified runtime loader: each time a binary is loaded in the process address space, the loader selects a random page-aligned address for the code, data, and heap segments.

Problem 5a (2 points)
What is this defense based on?

Problem 5b (5 points)
A famous Molvanian hacker R00tkowski has found an exploitable stack-based buffer overflow in the MS-DOS login program and decides to exploit it using a return-to-libc attack. He must correctly guess the buffer start address, as well as the location of execve call (which is the target of the control transfer in his attack) in the system call table.

How many guesses are necessary, on average, to ensure a successful attack? Explain. You can assume that if the guess is incorrect, the login program crashes and then automatically restarts.
Problem 6

*Blind IP spoofing* is an attack that hijacks an existing TCP connection between two hosts. Suppose hosts A and B are communicating using TCP. The attacker first establishes a connection with A. Because TCP sequence numbers are often assigned sequentially, the attacker uses the sequence numbers in his own connection to approximately guess the sequence numbers in the connection between A and B. The attacker then sends packets with B’s source address and an appropriate sequence number to A. Host A believes that these packets were sent from B and executes commands contained in them.

**Problem 6a (2 points)**

Why is this attack called “blind?”

**Problem 6b (4 points)**

Suppose that when the connection between A and B is established, TCP sequence numbers are computed in the same way as in the SYN cookie defense against denial of service. Does this help against blind IP spoofing? Explain.