CS 361S - Network Security and Privacy
Spring 2014

Homework #2

Due: 11am CDT (in class), April 17, 2014

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 Deparment of Computer Science code of conduct can be found at `http://www.cs.utexas.edu/undergraduate-program/code-conduct`.

Late submission policy

This homework is due at the **beginning of class** on **April 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 11am 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 Vitaly Shmatikov or Oliver Jensen. **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 Molvanîa (50 points)

Problem 1

`molvCrypt` is a string encryption function from a popular Molvanîan software library.

```
/*
 * str: An input string to be encrypted.
 * key: An 8-byte encryption key.
 * On failure, this function should return NULL.
 * On success, this function should return a pointer to a string that
 * has the length of the encrypted string encoded in the first 4 bytes,
 * followed by the actual encrypted string (not NULL-terminated).
 */
char *molvCrypt(char *str, char *key) {

   size_t len = strlen(str);
   char *crypted = malloc(len + 4);
   unsigned int i;

   for (i = len; i != 0; i--) {

       *(crypted + i + 4) = *(str + i) ^ (key[i%8]); // XORing with the key

   }

   *(size_t *)(crypted) = len;
   return crypted;
}
```

Problem 1a (4 points)

Mark every unsafe memory operation in the above code and explain why it is unsafe.

Problem 1b (4 points)

Add the missing checks needed to ensure that all memory operations in this code execute safely. Write these checks as inline comments in the blank spaces where they should have appeared in the code.

Problem 2

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

Problem 2a (3 points)

What additional protections are gained by using `libsafe` with stack-canary-equipped executables?

Problem 2b (4 points)

Consider the following code:

```
void omgwtfbbq()
{
    char query[256];
    char *p = query;
    char uid[64];
    char pwd[64];

    strcpy(p, "UPDATE user_accts SET password = '"); p += strlen(p);
    gets(uid);
    gets(pwd);
    strcpy(p, pwd); p += strlen(p);
    strcpy(p, "' WHERE userid = '"); p += strlen(p);
    strcpy(p, uid); p += strlen(p);
    strcpy(p, "'");
    db_query(query);
}
```

In addition to the obvious SQL injection vulnerability, this code is vulnerable to stack smashing. Suppose the compiler inserts stack canaries *and* all C string functions are protected by `libsafe`. Does this protect this code from stack smashing?

Problem 3

x68 is Molvanîan homegrown chip architecture. Unlike on x86, the stack on x68 grows upwards.

Problem 3a (4 points)

How does a stack overflow attack work on x68?

Problem 3b (4 points)

How would you implement stack canaries on x68? What would be the main differences from x86?

Problem 3c (4 points)

How would you implement `libsafe` on x68? What would be the main differences from x86?

Problem 4

A famous Molvanîan hacker R00tkowski proposes the following defense against buffer overflow attacks. All code pages should be mapped to low memory addresses (*e.g.*, from `00000000` to `0000FFFF` on x86 machines) and the rest of the pages should be marked as non-executable.

Problem 4a (3 points)

Would this technique prevent standard stack smashing exploits? What about `return-to-libc`? Explain.

Problem 4b (3 points)

Assuming that hardware modifications are not feasible, which system components (OS, compiler, linker, run-time environment, *etc.*) must be modified to implement this technique?

Problem 4c (3 points)

What are the disadvantages of protecting memory in this way?

Problem 5

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 5a (2 points)

Why is this attack called "blind?"

Problem 5b (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.

Problem 6 (4 points)

The Molvanîan 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 7 (4 points)

Molvanîan Security Agency (MSA) stores important secret documents on its internal website `http://msa.mi/topsecret`. The Web server hosting this site is configured to reject all connections from IP addresses located outside the MSA's local network.

Suppose an employee of MSA visits an external website. Explain how the owner of this website can use DNS cache poisoning to steal documents stored at `msa.mi/topsecret` (you should assume that the Web browser on the employee's computer correctly enforces the same origin policy).