CS 361S - Network Security and Privacy
Spring 2014

Homework #3

Due: 11am CDT (in class), May 1, 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 **May 1**. 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 #3: Molvanîa Forever (50 points)

Problem 1 (6 points)

It is easy to design an application-level gateway (proxy) for the FTP protocol. The proxy decides, for example, whether an outsider should be able to access a particular directory in the file system and issues a corresponding command to the inside file manager or refuses the outsider's request.

List **three** protocols that would be prohibitively difficult or impossible to proxy and explain why.

Problem 2

Consider the following UNIX program (written in pseudo-code):

```
networkWrite();
while (..) {
   networkRead();
   if (...) fileRead();
   else fileWrite();
   if (... fileOpen() ...) ...;
   else fileRead();
   fileClose();
}
geteuid();
setuid();
```

In the above code, the system calls made by the program are networkWrite, networkRead, fileRead, fileWrite, fileOpen, fileClose, geteuid, and setuid. Their arguments are omitted for simplicity.

Problem 2a (3 points)

How can system call interposition prevent an attacker from exploiting this program to execute a shell via the exec system call?

Problem 2b (6 points)

By looking at the program code, you notice that the program can only execute certain sequences of system calls when it is running normally. For example, it never performs a network read or write after calling setuid().

Write a regular expression that captures all legitimate sequences of system calls that this program can make. Your expression should involve concatenation, alternation, and Kleene star over the symbols NW, NR, FR, FW, FO, FC, GU, SU corresponding to the system calls listed above.

Problem 2c (3 points)

How does system call interposition based on the regular expression you wrote above can help enforce the following policy: "the program should never write to the network after reading a file"? Give an example of an attack that such a policy might prevent.

Problem 3 (8 points)

Molvanîan Organization for Security Defense (MOSDef) developed a new intrusion detection system designed to detect rootkits and adware. For the purposes of this problem, assume that the following three possibilities are exhaustive and mutually exclusive: either the host is normal, or rootkit-infected, or adware-infected. After conducting large-scale experiments, MOSDef computed the following accuracy rates for its product:

Type of host	How this host is classified		
	Rootkit	Adware	Normal
Rootkit	85%	5%	10%
Adware	5%	90%	5%
Normal	5%	5%	90%

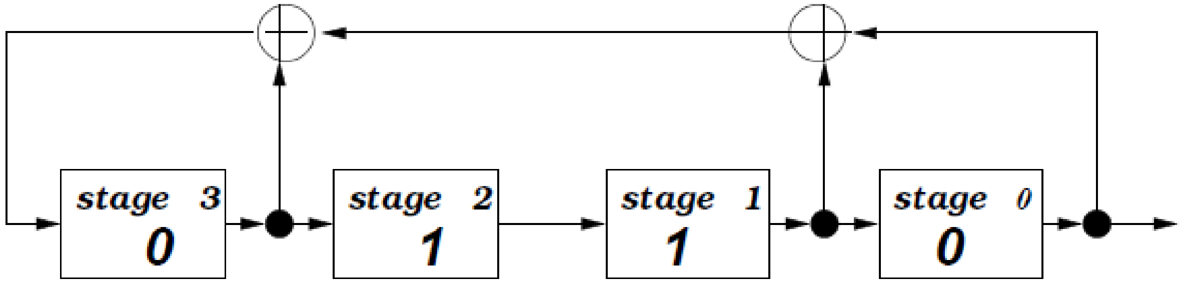
For example, when the MOSDef system analyzes a rootkit-infected host, it correctly classifies the host as rootkit-infected with probability 85%, misclassifies it as adware-infected with probability 5%, and misclassifies it as normal with probability 10%.

For the purposes of this problem, assume that 1% of all hosts are infected with rootkits, 2% are infected with adware, and the remaining 97% are normal.

When the MOSDef system announces that it has detected adware presence, what is the probability that the host is, in fact, normal? Give your calculations.

Problem 4 (6 points)

Molvanîan Telecom encrypts voice-over-IP datagrams using a stream cipher based on the following pseudo-random generator:



The generator is initialized with the key, the first 4 bits of the keystream are discarded, and the rest is used for encryption.

Suppose you want to completely recover the key using a known-plaintext attack. How long should your plaintext be? How would the attack proceed (describe all details)?

Problem 5 (6 points)

Having noticed that the mathematical operations involved in RSA signing and encryption algorithms are identical, Molvanîan Institute for Advanced Cryptology released the following implementation of the RSA signature scheme. Given a message m, their implementation signs it simply by "decrypting" m with the RSA private key d, *i.e.*, $\mathsf{sig}(m) = m^d \mod n$.

Recall that a signature scheme is secure if the attacker who does not know the private key cannot produce a signature on any message m' which was not signed by the holder of the key—even if the attacker observed many legitimate signatures.

Does the Molvanîan scheme meet this definition of security?

Problem 6 (6 points)

Molvanîan Institute for Advanced Cryptology proposed their own variant of RSA encryption. Recall that an RSA public key is a pair (n, e). To encrypt some message m, first generate a fresh random value r of the same length as m. Use r as if it were a one-time pad to encrypt m (*i.e.*, let $s = m \oplus r$), then encrypt r using plain RSA (*i.e.*, let $t = r^e \mod n$). The ciphertext is the (s, t) pair.

To decrypt the ciphertext (s, t), first compute $r = t^d \mod n$, where d is the RSA private key, then the compute plaintext $m = r \oplus s$.

Is this encryption scheme secure against the chosen-plaintext attack? Explain.

Problem 7 (6 points)

Remember the **cloning** attack on jPhone from Homework 1? The attacker (passive or active) gathers enough information from one or two phone calls to create a clone of the caller's phone. Later on, the attacker uses the clone to impersonate the victim to the cell phone company and make untraceable calls billed to the victim's number, even when the victim's own phone is switched off.

Molvanîan Telecom (MT) has released another version of its popular jPhone. For authentication, it uses the Digital Signature Standard (DSS). Each jPhone stores its own DSS private key x and secret random value k. Both are selected at random when the phone is manufactured and burned into tamper-proof read-only memory. The public key corresponding to x is stored in MT's database.

When a jPhone initiates a call, MT challenges it with a fresh, random challenge C. The phone executes the DSS signing algorithm and responds with $\mathsf{sig}_x(C)$, *i.e.*, challenge C is digitally signed with the phone's private key. MT verifies the signature using the phone's public key, and, if verification succeeds, completes the call and bills the phone.

Is jPhone secure against cloning? Give a detailed explanation.