

Homework 1

Due: February 11, 2009, 10:00 PM EST

Problem 1

1. An attacker can do a dictionary attack: Since the domain of the encrypted messages is restricted (i.e., there are 900 cases for the bailout amount and 10 different banks), the attacker could use the public key to encrypt each one of the possible plaintexts and compare them with what Barack sends. If there is a match, then, without knowing the private key of the recipient of the message, the attacker can know what Barack is sending.
2. (a) Barack can make the encrypted plaintext random by adding some random bits in front of the plaintext message. The recipient of the message will have to discard them after decrypting; (b) Barack can use public key cryptography to exchange a secret key (there is no way to perform a dictionary attack since the secret key is random) and then use that key for the communication.
3. Barack simply uses his private key to sign the message. Nobody can impersonate him since no one has his private key.
4. Suppose it does not. That means that someone can find a joke x' and compute $h(k||x')$. However, this is not possible unless he finds the key k . A way to do that is to eavesdrop $h(k||x)$ and then try to invert the function. This is also not possible since the cryptographic function is one-way.
5. Since k is known to other members of the cabinet, then everyone can use it to compute $h(k||x)$. Therefore Tim can claim that he came up with ideas instead of Barack.

Problem 2

```
int append(String source_file, string target_file) {
    try {
        // get the uid of the caller of the current process
        int caller_id = getUid();
        // get the uid of the owner of this program
        int my_id = getEuid();
        // set the caller_id to be the effective uid
        setEuid(caller_id);
        // open the caller's file
        int sfd = open(source_file, READ_ONLY);
        byte[] sourceBuffer = read(sfd);
        // restore my id
        setEuid(my_id);
        // open and read the target file
```

```

int tfd = open(target_file, READ_ONLY);
byte[] targetBuffer = read(tfd);
// append the two buffers
byte[] finalBuffer = targetBuffer + sourceBuffer;
tfd = open(target_file, WRITE_ONLY);
write (tfd, finalBuffer);
return 0;
} catch (SystemCallFailed e) {
  print ("System call has failed: " + e);
  return -1;
}
}

```

Problem 3 The `shar(1)` utility with HP9000 servers running HP-UX versions 11.11, 11.04, and 11.00 creates a temporary file in `/tmp` with an easily predictable file name, which could allow a local attacker to launch a symlink attack.

1. An attacker knows the predictable temporary filename N that `shar` uses.
2. The (local) attacker creates a symbolic link with the exact name N in `/tmp` to an arbitrary file on the system, for example, to a victim's personal file.
3. When the victim unpacks the `shar` file, `shar` does not create a temporary file with name N , as it already exists.
4. `shar` opens the temporary file for writes. This allows the attacker to overwrite files with privileges of the victim, as `shar` is run by the victim.

A simple fix to `shar` utility is to specify a safe directory for temporary file creation using the `TMPDIR` environment variable.

Problem 4 The unencrypted virus code is xored with the suspected program at different offsets. If the resulting bytes have sections in which 6-byte chunks are repeated several times, then the program is probably infected.

To detect the repeated sequence in the suspected program, one can compare all the bytes of 6-byte apart to see if there is a repeated pattern. Repeat this operation at different starting points. See the below figure.

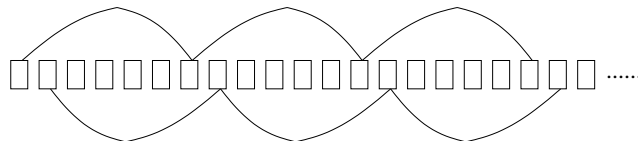


Figure 1: Each rectangle represents a byte of a suspected program. Arcs mean that bytes are compared to see whether they are the same. Not all comparisons are shown.