Coursework 2: Topics in Security

Due date: 12 noon, 11th December, 2008

Answer all five of the following problems. Either handwritten or typeset solutions are fine, but if you write by hand, please ensure your answers are legible.

Show all work! We cannot award credit for correct answers if their complete derivation isn't shown. Please state clearly all assumptions you make while solving a problem.

Collaboration is *not permitted* on this problem set; you may not discuss the problems or their solutions with anyone else (whether or not the other person is taking the class), apart from the instructor. All work you submit must be your own. You may of course refer to all lecture notes and readings, and any other materials you wish (textbooks, papers, or material found on the Internet).

This coursework is worth 5% of the total marks for M030/GZ03.

1. Suppose you are given an efficient algorithm, `RSA-Crack()`, that, for a given RSA public key (n, e), is able to decrypt 1% of the messages encrypted with that key (without knowledge of the corresponding private key). Describe an efficient algorithm that uses `RSA-Crack()` as a building block, and can decrypt *any* message.

   [10 marks]

2. You and your friend discover a format string vulnerability in a popular server, and decide to write an exploit for it that will make the server crash. An excerpt of the C source code for the function containing the vulnerability follows:

```
int vulnerable(void)
{
   char userinput[1024];
   ...
   sprintf(outstr, userinput);
   ...
}
```

   Both `outstr` and `userinput` are of type `char *`. Each of these two pointers is four bytes in length. `userinput` is a string that the server reads directly from a network socket (*i.e.,* the content of the string will be taken unmodified from within a request you can send the server). Assume that `outstr` is extremely large, such that you can be certain you won't overflow it, no matter how many characters are printed by `sprintf()` during the processing of your exploit.

   Your friend analyzes the behavior of the server, and determines the following facts:

   - When the vulnerable server software is run under the version of Linux used on the server you wish to target with your exploit, inside the function `vulnerable()`, the return address for resuming execution in `vulnerable()`'s caller is stored on the stack at memory address `0xbfff8218`.

- When `sprintf()` begins processing the format string `userinput`, its "next argument to print" pointer points at a memory location that is exactly 48 bytes lower in memory than the location of the buffer `userinput`.

Assume that memory address `0xdeadbeef` is unmapped in the server process, so that if execution at this address is attempted, the server will crash.

Design and supply the exact format string that, when placed verbatim by the server into `userinput`, will cause the server to crash by attempting execution at address `0xdeadbeef` when the function `vulnerable()` returns. Provide diagrams of the stack showing the steps in the execution of your format string exploit.

N.B. that you should *not* use a buffer overflow vulnerability in your answer—you must use a format string vulnerability only.

[10 marks]

3. Recall that SSL 3.0 uses hybrid cryptography; it uses public-key cryptography during the handshake to establish shared secret keys between the client and server, and then subsequently uses these secret keys with symmetric ciphers (and MACs). After the handshake, SSL 3.0 most commonly encrypts connection data using a stream cipher. The SSL 3.0 implementation of the stream cipher does not pad the plaintext before encryption—unlike block ciphers, stream ciphers do not require the input plaintext to be of fixed length.

   Suppose Alice sends SSL 3.0-encrypted HTTPS requests to Bob's web server. Bob's web server serves only static content—that is, requesting a given URL always returns the exact same page in response. Bob's web server is open to all; no login is required to browse the content on it. One may simply connect to Bob's web server using HTTPS, and follow links as desired to retrieve further pages using HTTPS.

   Eve can passively monitor all network traffic to and from Alice's workstation. Eve can also use her own workstation and browser to originate her own HTTPS requests to Bob's web server. She wishes to deduce (as best she can) which URLs Alice is browsing on Bob's web server, and the corresponding pages returned by Bob's web server in response.

   Assume that Eve *cannot* decrypt any ciphertexts she eavesdrops, and that SSL 3.0 is robust against replay attacks.

   Describe how Eve can nevertheless make her own requests of Bob's web server and analyze the traffic between Alice's workstation and Bob's web server in such a way as to deduce useful information that will help reveal to Eve which URLs Alice has browsed (and which documents have been returned in response by Bob's web server).

[10 marks]

4. Kerberos Version 4 (the version of the protocol in the paper assigned for class) uses authenticators to protect against replay attacks.

   Suppose that all nodes in a Kerberos realm have properly synchronized clocks, and that the clock synchronization system is secure against an adversary's manipulation of any node's clock.

The MIT Athena Kerberos deployment honored a Kerberos authenticator for 5 minutes beyond the timestamp within the authenticator. An eavesdropper could thus replay an overheard ticket and authenticator for five minutes, given that servers in this deployment didn't cache past authenticators to ensure they weren't reused.

(a) Describe an alternate, bidirectional protocol between server and client to replace the authenticator, that prevents such replay attacks, and requires no new keys beyond those already in use in the Kerberos system.

[7 marks]

(b) TAOS allows delegation: a user may delegate authority to a workstation acting on his behalf, and that workstation may in turn delegate authority to another workstation, acting on behalf of the same user. Does Kerberos support this kind of delegation? That is, can a user on workstation *A* obtain a ticket granting access to a service on some server, and forward that ticket as-is to another workstation, for use by that other workstation when requesting services from the same server? Why or why not?

[3 marks]

5. SSL 3.0 with RSA authentication during the SSL handshake doesn't provide forward secrecy. Describe a modification to the SSL 3.0 handshake (still only using RSA during the handshake) that will provide forward secrecy. Please show a timeline for your modified SSL handshake (of the form of the one given in lecture), indicating the interleaving of messages sent and received by the client and server, the contents of each message, and showing when the client and server execute any other operations required to implement your modification correctly. What, if any, are the added costs of your solution over those of the "basic" RSA SSL handshake? *Hint: consider what fundamental property the key the server uses to decrypt the pre-master secret must have for forward secrecy to hold.*

[10 marks]

Problem set total: 50 marks