UCL CS M030/GZ03 Brad Karp

Coursework 2: Topics in Security

Due date: 9:05 AM, 15th December, 2010

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. Please 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. This coursework is worth 15% of your final grade for M030/GZ03.

Please monitor the M030/GZ03 Moodle forum during the period between now and the due date for the coursework. Any announcements (*e.g.,* helpful tips on how to work around unexpected problems encountered by others) will be sent to the list.

Hand-in instructions: Hand in hardcopy for your solutions at the *start* of lecture at 9:05 AM on the 15th of December, 2010. If you submit this coursework late, please turn it in (with a clear indication at the top of the first page how many late days you would like to use) to the Computer Science reception desk on the fifth floor of the Malet Place Engineering Building (MPEB). There is no provision for electronic submission of this coursework.

IMPORTANT: Because the 17th of December is the last day of term, and this coursework must submitted in hardcopy, submission will not be possible after the 17th of December. Please be sure to turn in the coursework on or before that date!

Collaboration: 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 and teaching assistant. 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).

1. **The RSA Public-Key Cryptosystem**

   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. **Format String Vulnerabilities**

   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. **Formally Modeling SSL/TLS**

In the assigned reading on TAOS, the authors describe how to use logical statements to model authentication in distributed systems formally.

Suppose that Alice uses a web browser on her desktop computer to place an order with `amazon.com`. The communication between her browser and `amazon.com`'s web server uses SSL/TLS 3.0, with RSA authentication (as described in class during the SSL/TLS lecture).

(a) Using the notation and axioms defined in Section 2.1 of the TAOS paper, write out a formal derivation that proves that Alice's web browser can trust that a response *RSP* received over the SSL/TLS channel from `amazon.com`'s SSL/TLS web server was sent by the real company *Amazon.com, Inc.* (Assume that *Amazon.com, Inc.* has registered a public key with an SSL/TLS certification authority trusted by Alice's web browser, in the usual way.) Your derivation should conclude with the statement:

Amazon.com, Inc. **says** *RSP*

[7 marks]

(b) Note that the above concluding statement in the derivation does not include any statement about what exact software (and software version) is running on `amazon.com`'s web server. But in the derivation in the TAOS paper at the end of Section 2, the OS software is included in the concluding statement.

From Alice's point of view, what threat might exist on `amazon.com`'s web server because SSL does not include the web server's software in the security properties that Alice can formally derive? (Assume there are no exploitable vulnerabilities in the web server's software.)

[3 marks]

4. **Kerberos**

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**

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