Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ronald L. Rivest

Handout 30
November 7, 2002

Problem Set 6

This last (no more!) problem set is due on *Thursday, November 14, 2002* at the beginning of class. Late homeworks will *not* be accepted.

At the end of each problem, tell us how each person contributed (writing, designing, coding, proof reading, moral support, etc).

Problem 6-1. Buffer overflows

In 1995, André DeHon "accidentally" left in one of Prof. Gerry Sussman's 6.001 lectures copies of a report[1] analyzing the MIT Card system.

A new university, TIM, is now about to deploy the TIM card system[2]. An 31773 TIM graduate student hax0r found a snippet of code (Figure 1) near a heavily frequented soda machine. The code also appears on `http://web.mit.edu/6.857/www/lecture.html#L18`.

```
...
int check_access () {
  char recipient[20];
  char query_string[128];

  ...

  if (getenv ("QUERY_STRING"))
    strcpy(query_string, getenv ("QUERY_STRING"));

  ...

  if (access_granted) {
    printf ("TRUE\n");
  } else {
    printf ("FALSE\n");

    ...

      execl ("/usr/bin/mhmail", "mhmail", recipient,
             "-subject", "TIM Card",
             "-body", message,
             (char *)0);
      return 0;
    ...
  }
}
```

Figure 1: A snippet of the program code on the TIM Card server. The complete code is available on the 6.857 lecture handouts Web page.

[1] `http://www.ai.mit.edu/people/andre/mit_card/security_assessment/security_assessment.html`
[2] not to be confused with the MIT Card system

Apparently the card readers talk to the card server with an HTTP-like protocol. When a student slides a TIM card, the card reader sends over a secure link an HTTP request of the form:

`http://857.lcs.mit.edu/cgi-bin/TIM?location=⟨loc⟩&id=⟨id⟩`

The card reader sends the ID number read from the card along with the building location of the card reader. If the card server decides (based on database lookups) that the student is authorized to enter the building, the server responds positively (TRUE). Otherwise, the card server responses negatively (FALSE) and sends an email a TIM Big Brother for police action. If the card reader sees a positive response, it unlocks the door to the building. For example,

`http://857.lcs.mit.edu/cgi-bin/TIM?location=SCL&id=123-45-6789`

returns TRUE because the card with ID number 123-45-6789 is permitted to enter the SCL building (the TIM analogue to LCS). On the other hand,

`http://857.lcs.mit.edu/cgi-bin/TIM?location=baLaideM&id=123-45-6789`

returns FALSE because the card does not have permission to open the door at the baLaideM building.

A team of curious TIM students decides to have fun with the TIM Card system.

(a) What's wrong with this picture?

Describe any security vulnerabilities present in the card server program in Figure 1. Order the vulnerabilities in descending severity.

(b) Buffer overflow baby steps

Construct an HTTP request that will cause the TIM Card server to divert the "alarm" messages to your group's email account. That is, trick the TIM Card server to alert you instead of Big Brother.

In your write up, explain how your attack works. Include the URL you constructed and a transcript or screen shot of the email you capture.

Make sure not to cause a denial of service attack (e.g., by querying the machine incessantly). Do not write any programs to automatically make lots of queries to the Web server (it's not necessary anyway). The best solution will take exactly one HTTP query of the Web server to capture the email. Your goal is to obtain the email without being noticed by the administrators!!

Hint: A Web browser should suffice as your only tool.

(c) The artful buffer overflow

Now that you have learned how to exploit a simple buffer overflow, it's time to simulate a sophisticated attack on your local machine.

Write a program that crafts special environment variables such that when the `TIM-stolen` program runs, it produces a new `/bin/sh` prompt. You can probably find some starter code online.

Download the `TIM-stolen.c` code, compile it on an x86 architecture for best results, then write a program that calls `system ("./TIM")` to obtain a shell.

Include a couple lines of text or a screen shot to demonstrate that you obtained a shell. Include any code you write, but no more than a couple pretty-printed pages. Make your code concise! The shorter the working program, the more points you get.

If you are unable to write code that gives a shell prompt, explain your methods for partial credit.

(d) The moment you've been waiting for

This is an optional problem part. It may involve a lot of grunge work, so only do this if you have time for fun. This problem part will not be graded.

Exploit a buffer overflow in the TIM Card server program to obtain the entire database of TIM ID numbers and names by executing `/bin/sh` as the Web server user ID on the TIM Card server. At the same time, make sure the administrators receive no emails notifying them of your break-in attempts. That is, you still need to divert the notification emails to your email address. We suggest that you try to inject shell code over a return address on the stack.

Include a couple lines of text or a screen shot to demonstrate that you obtained a shell. Include any code you write, but no more than a couple pretty-printed pages. Make your code concise! The shorter the working program, the more points you get.

Make sure **not to cause a denial of service attack**. If you do make any kind of automated queries of the Web server, insert a one second delay between requests. More specifically, do not test your exploit code on the TIM Card server before you successfully get it to work on your own machine in the previous problem part.

You may find tools such as netcat useful for sending arbitrary binary code inside HTTP requests. For Netcat help on Athena, type:

```
% add sipb
% nc -h
% nc 857.lcs.mit.edu 80 < REQUEST_FILE > OUTPUT_FILE
```

If you get a shell prompt, please be kind to the server machine. Play nice.

These URLs may help you in your endeavor:

- "Smashing the Stack for Fun and Profit":
  `http://www.phrack.com/phrack/49/P49-14`
- Stackguard (Google it)
- The GNU C Library – Users and Groups:
  `http://www.gnu.org/manual/glibc-2.0.6/html_chapter/libc_26.html`

(e) Stackguard

The TIM programmers are thinking about recompiling the TIM CGI program with Stackguard[3]. If this were to happen, how would it affect your attack? What would you have to do to make your attack work again?

(f) Breaking is easy. Cleaning up is the hard part.

Demonstrating that a security system is weak is fun, but fixing the weakness is often a much harder problem.

Rewrite the C code in Figure 1 to make the code safe from buffer overflows. If fixing the hole would require modification of code or libraries outside of TIM-stolen.c, then explain (rather than implement) what could be done to remedy the situation. Do not change the protocol the TIM Card reader and server use to communicate.

[3]`http://www.usenix.org/publications/library/proceedings/sec98/cowan.html`

Problem 6-2. Vulnerability reporting

(a) Full disclosure

The debate on "full disclosure" often appears on the bugtraq mailing list. Browse through some of the articles listed on `http://www.wild.lib.fl.us/bib/disclosure-by-date.html` and `http://www.counterpane.com/crypto-gram-0111.html`.

If you found a major security hole in a widely deployed product, how would you report the vulnerability and exploit code? What would your decision depend on? What steps would you go through? Write up a response on a single page.

(b) Case study

Browse the security vulnerabilities reported on `http://www.cert.org/advisories/`. Explain on a single page (1) how one vulnerability was handled well by the authors of any exploits (without major incident) and (2) how one vulnerability was handled poorly (causing major destruction or unnecessarily severe pain).