CMPS 122 Winter 2019

Lab #2: Remote buffer overflow exploit

Due: 23:59 Thursday, February 14, 2019
This lab is 15% of your final grade

In this lab you will dictionary attack a remote web server to crack the password of a known user account, then attempt to subvert the site contents using buffer overflow exploits.

WARNING (1 of 2) Running a network or transport level port scanner `https://en.wikipedia.org/wiki/Port_scanner` on the SOE network is in direct contravention of UCSC regulations. If you are detected using such a tool, you may be expelled from the university. You do not need a port scanner for this lab; so do not be tempted to use one.

Background

Over the recent holiday weekend, you travelled to San Diego for a literary festival.

Whilst there you met a middle-aged, mildly eccentric Englishman who happily confessed his favorite novel was not by one of the many great authors from his own country, but is in fact `Around the World in Eighty Days` by the acclaimed French writer, Jules Vern. Your new friend seemed particularly taken with the manservant the protagonist hires after firing his valet for bringing him shaving water one degree centigrade too cold.

As the festival progressed, you kept bumping into this British admirer of Continental literature. At a buffet lunch you mentioned your current situation as an impoverished student of Computer Science at UCSC. In response the gentleman enthusiastically informed you he owned a software company, presented you with his business card, and suggested you keep in touch in case you fancied working for him after graduation.

From then on, you sought him out at every opportunity. He was never hard to find in a crowd as he was always wearing a replica shirt of his favorite English football team. That's association football, of course, not the peculiar game enjoyed on this side of the Atlantic were players are forced to don what look like motorcycle helmets, and spend little more than a few minutes on the field during a three hour game.

On the last night of the event, you found your potential future employer in a quiet neighborhood restaurant, enjoying a glass or several of full-bodied Californian red. You declined his offer to share the bottle, but joined him for dinner and listened intently to his "back in the day" stories about how he and a handful of college friends would construct entire systems from scratch in a matter of days.

As the evening wore on, another bottle of wine was ordered. Suitably lubricated, the software entrepreneur told you of a system he and his friends had written at the very dawn of the World Wide Web. He went on to claim this thirty-year-old system was still in use at a number of large public universities.

Seeing your incredulity and glancing around to make sure he was not overheard, the now distinctly tipsy Englishman breathed wine fumes in your direction and whispered...

"Don't tell anyone, but there's a back door in it. Piece of cake getting in so long as you know the skeleton key and can guess my password."

Thinking quickly, you asked, "So you hard-coded your account details?"

The confirmation was swift. "You know it, mate. I can get into those servers any time I like!"

On your return to Santa Cruz, you made discrete enquiries and discovered UCSC had once used web servers from the Englishman's company, but no one could remember when they had last been operational. One engineer you spoke to observed...

"For all I know, it could be running somewhere on campus right now."

Scenario

It turns out the web server with the backdoor is running 100's of websites for UCSC, all on the same machine, the IPv4 address of which is disclosed in the Canvas assignment for this lab. The servers are listening on different network ports but all you have been able to discover about them is that they are outside the IPv4 reserved range.

As the servers in question support HTTP, you could use a browser to connect to each potential port in turn, but this could prove time consuming, as there are approximately 64,000 possibilities. A better approach would be to automate the search by writing some code or a script.

WARNING (2 of 2) You can develop code and experiment with ideas on you own machine(s), but all attacks against the compromised servers must originate from grunhilda.soe.ucsc.edu.

Setup

SSH in to the CMPS122 teaching server. Use Putty (`http://www.putty.org/`) if on Windows:

```
$ ssh <cruzid>@grunhilda.soe.ucsc.edu
```

Create a suitable place to work: (only do this the first time you log in)

```
$ mkdir -p CMPS122/Lab2
```

Requirements

Basic:

- Find and gain non-browser access to the web server that has your CruzID as the backdoor account username.

Advanced:

- Mount a buffer overflow attack against "your" server to demonstrably compromise its operation without crashing it.

Challenge:

- Automate the techniques you used for the Basic and Advanced requirements so they can be run in a parameterized fashion to compromise any identical instance of this old web sever.
- All steps undertaken for Basic and Advanced should be automated, including the dictionary attack, logging into the server, and downloading both the source code and the binary image.
- Your automation should discover the password "blackout" period and adjust accordingly.

Stretch:

- Mount another buffer overflow attack against "your" server to gain shell access.
- Use this shell to replace the homepage with one of your own.

Extreme:

- As for Challenge, but for a modified version of the web server. Do NOT assume the compromisable buffer is the same length and the unlock function is at the same address as it was in "your" server. i.e. you should automate the calculation of the sizes, addresses and offsets as well as their exploitation.

Time Frame and Schedule

Once you've found `your` server, you should expect the dictionary attack against it to take anything from a few seconds, if you get lucky, to an entire week. If you still haven't gained access after five days, ask for a password hint.

While the dictionary attack is in progress, you should use your time to work out how to undertake a buffer overflow attack against a vulnerable binary, There are many, many resources available online and you can also refer to the web cast of the live demonstration I did during the lecture on Thursday January 17.

What to submit

A UNIX format ASCII file named RESULTS providing evidence of basic and advanced requirements having been met. This file must have exactly three lines in it, with the following information:

- Line 1: Skeleton key required to gain backdoor access for the basic requirement
- Line 2: Password required to gain backdoor access for the basic requirement
- Line 3: The re-direct URL discovered when the advanced requirement is completed

For example:

```
SuperSecretKey
Pa$sw0rD
http://www.example.com/some/url.html
```

This file will be graded automatically. If the format is incorrect you will receive no points.

UPLOAD THE RESULTS FILE TO THE CANVAS ASSIGNMENT.

You should also submit a gzipped archive named `CMPS122-Lab2.tar.gz` containing all the code and scripts you used to complete this lab.

If you were able to complete the challenge requirement, the entry point for your automation should be a bash shell script named automate. sh in the first level directory of your archive. A call to this script would be:

```
$ ./automate.sh [ip address] [skeleton key] [user] [dictionary]
```


Where dictionary is a file with one password (or passphrase with spaces between words) per line. On completion, this script should display the automatically discovered URL for the advanced requirement as the last line of output. This will be graded automatically, if any other last line is found, you fail the test.

An example submission once unpacked might look something like this:

```
automate.sh
extreme.sh (see below)
src\something.c
src\Makefile
scripts\findport.sh
scripts\findpasswd.sh
python\foo.py
python\bar.py
```

UPLOAD THIS CODE ARCHIVE TO THE SAME CANVAS ASSIGNMENT AS YOUR RESULTS FILE.

In addition to submitting the results file and your code, you are required to keep a journal, which must be submitted as a PDF document named `journal.pdf`.

This journal should contain brief notes on:

- Everything you did
- URLS of places you found useful information from
- Details of failures and what investigations you undertook to try and find out why

The journal is mainly for your benefit so you can re-do steps later if they didn't quite work out the first time or you need to modify them in some way.

SUBMIT YOUR JOURNAL TO THE SAME CANVAS ASSIGNMENT AS YOUR ARCHIVE AND RESULTS.

Note that the journal WILL NOT BE READ unless plagiarism is detected in your submission.

For the extreme requirement, include in your submission a shell script named `extreme.sh` with the same signature as that for the challenge requirement, namely:

```
$ ./extreme.sh [ip address] [skeleton key] [user] [dictionary]
```

Where dictionary is a file with one password (or passphrase with spaces between words) per line.

What steps should I take to tackle this?

Come to the sections and ask.

How much code will I need to write and in what language?

Hard to say. Could be a few hundred lines, could be thousands. To a large extent it depends on the tools you choose to use. If you are familiar with python, you can certainty use that for some parts, if not all, of this. If you are an experienced C programmer, you may choose to use that. Better solutions will almost certainly use a combination of technologies.

Grading scheme

The following aspects will be assessed:

1. (100%) Does it work?
    a) Basic Requirements (40%)
    b) Advanced Requirements (30%)
    c) Challenge Requirements (10%)
    d) Stretch Requirements (10%)
    e) Extreme Requirements (10%)
2. (-100%) Did you give credit where credit is due?
    a) You submission is found to contain code segments copied from on-line resources, but you did not give clear and unambiguous credit to the original author(s) in your source code (-100%)
    b) You submission is found to be a copy of another CMPS122 student's submission (-100%)
    c) Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes a significant percentage of your submission:

$< 50\%$ copied code	No deduction
50% to 75% copied code	(-50%)
75% to 90% copied code	(-75%)
$> 90\%$	(-100%)

