Spring 2019 Cryptography and Network Security

Homework 3

Release Date: 5/28/2019

Due Date: 6/23/2019, 23:59

Instruction

- **Submission Guide:** Please submit all your codes and report to CEIBA. You need to put all of them in a folder named by your student id, compress it to hw3_{student_id}.zip. For example, hw3_r04922456.zip. The report must be in **PDF** format, and named report.pdf.
- You may encounter new concepts that haven't been taught in class, and thus you're encouraged to discuss with your classmates, search online, ask TAs, etc. However, you must write your own answer and code. Violation of this policy leads to serious consequence.
- You may need to write programs in the Capture The Flag (CTF) problems. Since you can use any programming language you like, we will use a pseudo extension code**.ext** (e.g., code.py, code.c) when referring to the file name in the problem descriptions.
- You are recommended to provide a brief usage of your code in **readme.txt** (e.g., how to compile, if needed, and execute). You may lose points if TAs can't run your code.
- In each of the Capture The Flag (CTF) problems, you need to find out a flag, which is in `BALSN{...}` format, to prove that you have succeeded in solving the problem.
- Besides the flag, you also need to submit the code you used and a short write-up in the report to get full points. The code should be named **code{problem_number}.ext**. For example, code3.py.
- In some CTF problems, you need to connect to a given service to get the flag. These services only allow connections from 140.112.0.0/16, 140.118.0.0/16 and 140.122.0.0/16.
- The TAs in charge of each problem are listed below.
  - TA 1: Problem 1, 2, 3, 4
  - TA 2: Problem 5, 6
  - TA 3: Problem 5, 6

  If you have problems, please come to the right TA hour. You can also mail to us.

Fun Hands-on Projects

1. My First Project (15%) (Bonus: 10%)

This is AngelBaby's first `C++` project. He finished this code in an hour. He wants to test the code and find if there are any **crucial bugs** because he knows that these bugs may cause serious damage. Can you point out at least **3 bugs** in this code?

Bugs that are counted as crucial bugs:

1. Cause the program crashed.
2. Access region that are not supposed to.

You have to explain what the bugs are, and how to fix the bugs in your report. The code is in `hw3/firstProject`.

Please do not modify `firstProject.cpp`. You need to modify `firstProject_patch.cpp` to fix the bugs you found. Then, use `diff -u firstProject.cpp firstProject_patch.cpp > firstProject.diff` to create the patch file.

Attach both `firstProject_patch.cpp` and `firstProject.diff` in your homework.

*Hint: You can use any **techniques** to find the bugs.*

Bonus: If you find more than 3 bugs, you will get additional marks.

2. Pokémon Master (15%)

This is a white-box code review challenge. The code contains some vulnerability. Please try to exploit them to get the flag.

1. Please get the flag, and explain how you solve this challenge in detail.
2. Explain the server's bug in detail. Why is it vulnerable?
3. Propose a fix for this vulnerability.

The challenge is running on http://140.112.31.97:10161. The challenge source is included in `hw3/pokemon_master`.

Hint: You don't need to be proficient at Python3 to solve this challenge. The vulnerability is language-independent.

3. Fuzz it! (20%)

Given an instrumented binary, you are asked to design a fuzzer that will repeatedly generate (or mutate) new input.

The service `nc 140.112.31.97 10160` is running the instrumented binary, it will output the coverage information for your input. The binary contains **5 flags**. Your task is to keep on fuzzing using your self-design fuzzer to get all the flags.

The instrumented program psuedo code:

```
unsigned char buf[0x30]; // global variable
void fuzz();    // target function (which include 5 flags)

int main() {
    while (1) {
        memset(buf, 0, sizeof(buf));
        read(0, buf, 20);
        fuzz();
    }
}
```

Sample code for `fuzz()` function:

```
void fuzz() {
    if(buf[3] == 190) {
        puts("1"); // coverage information
        if(buf[9] <= 200) {
            puts("1-1"); // coverage information
        }
        if(buf[6] != 19) {
            puts("1-2"); // coverage information
            puts("FLAG"); // print flag if you reach here
        }
    }
    if(buf[14] >= 248) {
        puts("2"); // coverage information
    }
}
```

Here is some information for the instrumented binary:

1. Each input size should be less than or equal to 20 bytes.
2. In `fuzz()`, there will be only `if-condition` (without `else`).
3. The depth of the `if-condition` is at most 5.
4. There are only 4 condition operators, which are ==, ! =, >=, <=.
5. Each condition will compare only one byte.
6. The coverage information for each branch is unique.

In your report, you have to include **5 flags** and explain how you design your fuzzer. You also need to submit your fuzzer code.

Important: Please do not (D)DoS the service!

Note: There is a 60s timeout for the service.

Hint: The tool introduces in guest lecture may help :)

4. Symbolic Execution (20%)

You are a secret agent, you notice that Eve hides his evidence (a **flag**) in this binary. You can't get the evidence because you don't know the secret code. Your assistant helps you to reverse the binary (`hw3/se/uuid`) and send the source code (`hw3/se/uuid.c`) to you. Can you get the secret code and receive the flag?

You need to include the **flag** and explain how you get the secret code in your report.

You can access the service by `nc 140.112.31.97 10162`.

Useful tutorial:

1. https://ceiba.ntu.edu.tw/course/1a23b8/content/20190521_CNS.pdf
2. https://hackmd.io/s/S1olWZYTN

Note: Directly using symbolic execution won't help because of ***Path Explosion****.*

Hint: You only need to modify ***3 lines*** *of code.*

5. BGP and Network Model (22%)

Realistic simulation is needed to conduct BGP research without affecting the Internet. In this problem, you will learn to generate realistic topologies (containing ASes and the relationships between them) and perform simple measurement.

A. Topology Generation

1. (2%) Follow the instruction on this github repository to install *brite*, a topology generator, on your machine. Attach a screenshot of its GUI interface in your report to prove your work.
2. (5%) *brite* supports several network models, each of which has different characteristics. Between the *BarabsiAlbert (BA) model* and the *Waxman model*, which one do you think can better represent the AS-level topology? Please briefly explain your answer. Remember to acknowledge the source of your information.

For the rest of this problem set, please use the following configuration as default unless specified otherwise.

- Topology Type: 1 Level: AS ONLY
- HS: 1000
- LS: 100
- Model: BA
- Node Placement: Random
- Growth Type: Incremental

- Pref. Conn: None
- Bandwidth Distr: Constant
- alpha: NA
- beta: NA
- Max BW: 1024
- Min BW 10

B. Simple Measurement

3. (7%) We would like to know the usage of the m parameter in the BA model. For ease of observation, please draw a figure in which the x-axis represents the number of ASes, and the y-axis represents the number of edges. The figure should contain three lines, each of which corresponds to m to be 3, 4, 5, respectively. Your data point for the number of ASes should be from 100 to 400, and 50 as the interval. Each data point should be the average of 3 generated topologies. Your report should contain a screenshot of the figure, describe your findings, and infer the usage of m.

   Please also save your topologies as `[AS NUMBER]-[m]-[id].brite` under a folder named `topologies`. For example, the first generated topology with 100 ASes and $m = 3$ should be named as `topologies/100-3-1.brite`.

4. (8%) We have learned a DDoS defense proposal called Routing Around Congestion (RAC) in homework #2. When we prepend some ASes in the BGP update messages, those AS will not be able to become a part of the route due to loop protection. Such approach is called BGP Poisoning. It is known that if the BGP update messages for the reroute are leaked to some AS not on the reroute path, adversaries can learn the new path and conduct further attacks. Therefore, the RAC messages should be restricted inside the ASes on the reroute path. To do so, every neighbor of the path should be *poisoned*, so that these update messages will be dropped by the neighbors of the path and will not reach the adversary.

   Assume all ASes select their route based on the AS path length. Here we choose two random ASes, AS 2 and AS 100 for the following experiment. Suppose a server in AS 2 is suffering DDoS attack because of link congestion, AS 2 would like to announce a new AS path prepending some specific AS to reroute its critical traffic from AS 100 through a shortest path that is different from the original path between them. Thus, AS 2 should prepend the neighbor AS which is on the shortest path from AS 100 to AS 2 to its AS path advertisement. Please caculate how many AS should AS 2 prepend to isolate this path according the topologies you created in (3), and draw 3 graphs for $m = 3, 4, 5$ respectively with x-axis as the number of total ASes, and y-axis as the number of isolated ASes. Note that if there are more than one shortest path, just choose any one of it, and you can safely assume that the shortest path you select is not congested.

6. SSL Stripping (18%)

In this problem, you are asked to setup two virtual machines—an attacker VM and a victim VM, and the attacker VM will launch the SSL stripping attack against the victim VM. As learned in class, the attacker can see all the traffic content if the victim accidentally uses the HTTP protocol to connect to an HTTPS-required website.

You can use any OS on your virtual machines. However, Kali Linux is recommended.

1. (9%) For the purpose of this homework, you need to reroute all the traffic of your victim VM to the attacker VM. `arpspoof` is a common tool to perform *ARP spoofing attack*. Using ARP spoofing, the attacker can intercept all the victim's traffic and see the content of unencrypted packets. Demonstrate your work by recording the intercepted packets with `wireshark` on the attacker when the victim machine is visiting http://linux10.csie.org:15004/hw3.htm. Write down the steps and commands you use, and save the captured packets as `packet1.pcapng`. Remember to describe your environment settings, including the IP address and virtual machine network setting (e.g., NAT/Internal/etc.) of each VM you use. You might need to enable ip forwarding on the attacker VM so that the victim can still access the Internet as usual.

2. (9%) However, with only `arpspoof`, the attacker cannot get the content if the packets are encrypted, such as by HTTPS. `sslstrip` is another common tool to perform SSL Stripping attack. Demonstrate your work by recording the intercepted packets (of the victim) using `wireshark` on the attacker VM when the victim visits a login page protected by HTTPS but not HSTS (e.g., myNTU or CEIBA).

   Write down the steps and commands you use, and save the captured packets as `packet2.pcapng`. The packets captured should include the username and password entered by the victim. Please do not enter any real password when you are conducting the experiment.