Spring 2018 Cryptography and Network Security

Homework 3

Release Date: 5/17/2018

Due Date: 6/9/2018, 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: DDoS (feat. Web security)
  - TA 2: Symbolic Execution, IoT Security
  - TA 3: Capture the Ether

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

Fun Hands-on Projects

1. IoT Security (25%)

Most of the IoT devices use ARM architecture. In this lab you are asked to exploit all these vulnerable programs compiled in the RaspberryPi device. Note that these challenges are non-trivial and really hard for newcomer. You may experience something you never learn before, like `buffer overflow`, `gdb` and so on. However, we will give you some hints for all of the challenges. If you are stuck, please refer to the tutorial for problem1.

All of these challenges are from Azeria Labs. So there will be no flag in these challenges. However, you are asked to answer all the questions below and write in the report to prove that you successfully exploit all the challenges.

Note that you only have to solve stack0, stack1, stack3 and stack4, feel free to solve other questions by yourself.

(a) stack0 (5 points)

1. (5 points) At least how many characters you should input to trigger the `buffer overflow`?

(b) stack1 (6 points)

1. (3 points) What is the `specific value` (4 characters)?
2. (3 points) How you find it?

(c) stack3 (7 points)

1. (3 points) What is the address for the `win()` function?
2. (4 points) Explain how you are able to call the `win()` function.

(d) stack4 (7 points)

1. (3 points) What is the address for the `win()` function?
2. (4 points) Explain how you are able to call the `win()` function.

Your input should contain your student ID. Please screenshot your result and paste it in the report. Note that you cannot use tools other than `GDB` or `objdump`, otherwise you will not receive any credit. For more specific detail, please refer to the tutorial for problem1.

2. Symbolic Execution (25%)

(a) aMAZEing (15 points)

Given a simple Maze game, you are asked to reach the "#" symbol in order to get the flag. However, it seems like this maze has no solution. Is this true?

This problem provides several files including:

1. `check.h`: A header file for the function *check_wall*().
2. `check.o`: A linker file for the function *check_wall*().
3. `check.bc`: A llvm linker file for the function *check_wall*().
4. `maze.c`: The Maze game source code.

Here is how the program works:

1. Command: 'w' = up, 'a' = left, 's' = down, 'd' = right.
2. You are asked to input a list of command as a string. For instance, input "wwasd" = up up left down right.

The maze is visually unsolvable. However, what if the program itself has bug which leads us to an unintended solution? It will be great if we can trace our program using some tools to find all these non-trivial (or careless) bugs.

Can you solve this Maze? If you submit the unintended (correct) solution to `nc 140.112.31.96 10132`, you will get the flag. Note that this problem has only **one unintended (correct) solution**. Please state all the bugs in the report.

Tutorial for KLEE and Problem 2

(b) Flag Verifier (10 points)

You already know the flag! Here is the Flag_Verifier, it will verify if you got the correct flag!

This problem provides one file:

1. `flag_verifier.c`: The Flag_Verifier source code.

Here is how the program works:

1. Given an input (a string), it will check if you got the correct flag.

If you already solved `Problem 2(a)`, we assume you aready know the basic idea of how to use the Symbolic Execution Engine `KLEE`.

In this problem, you will encounter one of the most challenging issues in Symbolic Execution called `Path Explosion`. If you use a similar method as `Problem 2(a)` to solve this problem, you are unlikely to get the solution because the search path is exponentially increased, so the engine can't reach the expectation target in feasible time. However, `Path Explosion` can be mitigated if you prune some unnecessary paths. Since you have the source code, can you try to prune those paths in order to let `KLEE` to find the solution? Please state all the paths you pruned in the report.

Tutorial for KLEE and Problem 2

3. DDoS (feat. Web security) (25%)

(a) Mirror Force (9 points)

In hw2, you have bypassed Cloud-based Security Providers (CBSPs) by finding the origin IP of the website. The next step is to launch a DDoS attack. However, as you have limited IoT bots, it would be reasonable to launch an amplification attack.

You have to find an amplification medium (amplifier), which meets the following requirements:

1. The amplification factor should be more than 20x.
2. It should be possible to spoof source IP address to the amplifier.

Please compute the amplification factor, and describe how to launch the amplification attack in detail.

Note: Please be careful when finding an amplifier. On the NTU campus and dormitory networks, suspicious network activities may result in being banned by NTUCC. (I think it's a violation of privacy...)

(b) The Revenge of Hash Table (8 points)

Another method to paralyze a website is Low Bandwidth DoS via algorithmic complexity attack. You are required to exploit the Python Hash Table to exhibit the worse case complexity. Instead of $\Theta(1)$, the crafted evil input would degenerate to $\Theta(n^2)$ for n operations.

Here is a small program **dos.py** written in Python 2.7 that runs fast in most cases. Please write a program that generate 50000 numbers in $[0, 2^{30}]$, such that this program runs slowly. Your evil input should make it at least 10x slower than random input. A simple input file **normal_input** is provided for reference of input format. Save your program as **dos.ext**, and your evil input as **dos.txt**. You must explain how you designed the evil input.

Hint: "The force is with those who read the source." - Isaac, founding member of Balsn CTF team.

So just read the source dictobject.c.

(c) 499 Chaos (8 points)

In fact, rather than taking down the website using DoS, some hackers try to exploit logic vulnerabilities in code and cause severe financial losses.

Recently, Balsn telecommunication, `http://140.112.31.96:10131`, annonces a plan with unlimited data, which costs only NT$ 499. You want to buy the 499 plan for your mother as a Mother's Day gift, but you have NT$ 0 dollars because of the GRE registration fee. Please try to buy the 499 plan for free and obtain the flag! The source code is available in `app.py` .

4. Capture the Ether (25%)

(a) (25 points) 2017 has been a crazy year for smart contract and decentralized application (DApp). Hundreds of startups have raised over 6 million dollars with it. With so much money stored on smart contracts, they obviously became hackers' targets. Here we prepared a `Capture the Ether` smart contract as an easy hands-on practice for hacking smart contract. Enjoy!
Reminders:

1. The `Capture the Ether` contract is deployed on `Ropsten` testnet, not Ethereum mainnet. Contract address: 0x9b56a7c72d9782503fa1684ae0fca835c0973638
2. Please pass in your student ID (lower case, ex: b02902000) as a parameter while solving the challenges. MaKe sure your score is increased and recorded on the smart contract. DO NOT use others' student ID. Using others' student ID will lead to serious consequences. If your student ID is taken by others, please contact CNS TAs.
3. Please describe your solutions in the report or submit your codes (if any) as code4-*.ext for any challenge you solved.
4. FAQ: `https://hackmd.io/s/SkACuaFRG`

(b) (5 points bonus for Capture the Ether only) One of the biggest hacking events by far was `the DAO hack`, in which the hacker took away 50 millions worth of Ether, causing Ethereum to hard fork. Please answer the following questions:

1. What is `hard fork`? What happened after Ethereum hard fork?
2. What was the vulnerability of `the DAO` contract? Explain how the hacker hacked it.