**CSE 5474 — Software Security** Spring 2022

# BabyHeap CTF

*Due April 21st, 2022*

## 1 Overview

In this CTF assignment, you are given 9 challenges with varied difficulties (from `babyheap_level1` to `babyheap_level9`). In those challenges, you need to write pwntools (no alternative, otherwise it will be very challenging) python scripts to send commands to the vulnerable program and exploit its heap vulnerabilities. All of the vulnerable program has the following user interface:

```
What do you want to do?
1. add a chunk
2. edit a chunk
3. delete a chunk
4. show the content of a chunk
5. check to win
6. exit
Choice:
```

- `babyheap_level1`. In this challenge, you need to exploit the use after free vulnerability. Basically, you need to allocate a small chunk, free it, edit the chunk metadata with the address of `check_variable` so that the system believes the free chunk has a successor free chunk with data portion in the address of `check_variable`. Then you need to allocate twice, which at the same time, overwrites the `check_variable` with a value. Then by calling check function, you catch the flag.

- `babyheap_level2`. In this challenge, you need to exploit the double free vulnerability. Similar to babyheap_level1, you need to allocate a small chunk, free it twice, allocate again to edit the chunk metadata with the address of `check_variable` so that the system believes the free chunk has a successor free chunk with data portion in the address of `check_variable`. Then you need to allocate twice, which at the same time, overwrites the `check_variable` with a value. Then by calling check function, you catch the flag.

- `babyheap_level3`. In this challenge, you need to exploit the heap overflow vulnerability. You need to allocate two small chunks, free them, allocate again to edit the first chunk with a payload that overflows to the second chunk, making the second chunk pointing to exit GOT entry. Then you need to allocate twice, which at the same time, overwrites the exit GOT entry with the win function address. Then by calling program option 5, which is the exit option, you catch the flag.

- `babyheap_level4`. In this challenge, you need to exploit the double free vulnerability. Similar to babyheap_level2, you need to allocate a small chunk, free it twice, allocate again to

edit the chunk metadata with the address of exit function so that the system believes the free chunk has a successor free chunk with the address of exit function. Then you need to allocate twice, which at the same time, overwrites the exit function with win function address. Then by calling program option 5, which is the exit option, you catch the flag.

- `babyheap_level5`. In this challenge, you need to exploit the heap overflow vulnerability. Specifically, you need to perform heap overflow three times to: 1) overwrite puts GOT entry with printf plt entry and use format string vulnerability to get libc base address; 2) overwrite exit GOT entry with libc setuid address; 3) overwrite free GOT entry with libc system function address and with /bin/sh parameter; 4) trigger the attack by selecting corresponding options (which will call system function with parameter /bin/sh); 5) finally, use cat /flag in your shell.

- `babyheap_level6`. In this challenge, you need to exploit the use after free vulnerability in fast bin. Particularly, you need to: 1) prepare the tcache bins and fast bins by malloc(s) and free(s); 2) perform use-after-free attack on fast bins to let it point to the got table; 3) overwrite the whole got table with the function's real address in libc, with exceptions that exit GOT entry is replaced with setuid function address and free GOT entry with system function address; 4) trigger the attack by selecting corresponding options (which will call system function with parameter /bin/sh); 5) finally, use cat /flag in your shell.

- `babyheap_level7`. In this challenge, you need to exploit the double free vulnerability in fast bin. The program will print out a "Christmas gift" which is the runtime libc base address. You need to: 1) prepare the tcache bins and fast bins by malloc(s) and free(s); 2) perform double free attack on fast bins to let it point to the got table; 3) overwrite the whole got table with the function's real address in libc, with exceptions that exit GOT entry is replaced with setuid function address and free GOT entry with system function address; 4) trigger the attack by selecting corresponding options (which will call system function with parameter /bin/sh); 5) finally, use cat /flag in your shell.

- `babyheap_level8`. In this challenge, you need to exploit the double free vulnerability in fast bin. You need to: 1) calculate libc base address chunks in unsorted bins; 2) prepare the tcache bins and fast bins by malloc(s) and free(s); 3) perform double free attack on fast bins to let it point to the got table; 4) overwrite the whole got table with the function's real address in libc, with exceptions that exit GOT entry is replaced with setuid function address and free GOT entry with system function address; 5) trigger the attack by selecting corresponding options (which will call system function with parameter /bin/sh); 6) finally, use cat /flag in your shell.

- `babyheap_level9`. In this challenge, you need to exploit the double free vulnerability in fast bin. You need to: 1) prepare the tcache bins and fast bins by malloc(s) and free(s); 2) perform double free attack on fast bins to let it point to the got table; 3) overwrite the whole got table with the function's real address in libc, with exceptions that put GOT entry is replaced with printf function address; 5) trigger the format string attack by selecting corresponding options to leak libc base address; 6) prepare the tcache bins and fast bins by malloc(s) and free(s); 7) perform double free attack on fast bins to let it point to the got table; 8) overwrite the whole got table with the function's real address in libc, with exceptions that exit GOT entry is replaced with setuid function address and free GOT entry with system function address; 9)

trigger the attack by selecting corresponding options (which will call system function with parameter /bin/sh); 10) finally, use cat /flag in your shell.

## 2 Requirements

There are two requirements for heap overflow. 1) the specific version of libc library. 2) ipdb. First, please go to `https://drive.google.com/file/d/1FM6mBGqr-t5rTd5jCmfd32BgHcAIEd4v/view?usp=sharing` and download the file. Then, follow the bash commands.

```
mv ~/Downloads
unzip 2.27-3ubuntu1.2_amd64.zip
sudo mv 2.27-3ubuntu1.2_amd64 /lib
```

For the debug purpose, you can use ipdb. Here is the bash command to install ipdb.

```
pip3 install ipdb
```

There are two steps in using ipdb. 1) Make a breakpoint using ipdb.set_trace(). 2) print (p.pid) (p comes from process function). When the program stops at the breakpoint, you can open new terminal and execute bash coommand 'gdb –pid=number of p.pid'.

## 3 Script template

This section gives you a script template and you can develop your attacks based on its utilities. This code can also be downloaded from `http://web.cse.ohio-state.edu/~lin.3021/file/s21a/babyheap_exploit_template.py`

```
#!/usr/bin/python3
from pwn import *
import ipdb

binary = './babyheap9'

gs = """
break main
continue
"""

e = ELF(binary)

END_OF_MENU = "e.g, 1\n"

def add(p, content, size=None):
        """
        scaffolding for communitcathing with the program
        """
```

```
        if not size:
                size = len(content)
        p.sendlineafter("Choice:", '1')
        p.sendlineafter("Size of the chunk?:", str(size))
        p.sendafter("Content:", content)
        line = p.recvregex("Chunk (\d+) is created")
        res = re.match(b"Chunk (\d+) is created", line)
        idx = int(res.group(1))
        return idx

def edit(p, idx, content):
        """
        scaffolding for communitcathing with the program
        """
        p.sendlineafter("Choice:", '2')
        p.sendlineafter("Index:", str(idx))
        p.sendafter("New content:", content)
        p.recvuntil("updated successfully!")

def delete(p, idx):
        """
        scaffolding for communitcathing with the program
        """
        p.sendlineafter("Choice:", '3')
        p.sendlineafter("Index:", str(idx))
        p.recvuntil("deleted successfully!")

def show(p, idx):
        """
        scaffolding for communitcathing with the program
        """
        p.sendlineafter("Choice:", '4')
        p.sendlineafter("Index:", str(idx))
        p.recvuntil("is:")
        line = p.recvuntil("\n\n---")
        return line[:-5]

def check(p):
        """
        scaffolding for communitcathing with the program
        """
        p.sendlineafter("Choice:", '5')

def exit(p):
        """
```

```
        scaffolding for communitcathing with the program
        """
        p.sendlineafter("Choice:", '6')

def launch_attack(p):
        e = p.elf
        libc = e.libc

        ################## write your attacks here. ##################

def main():
        if args.GDB:
                p = gdb.debug(binary, gdbscript=gs)
        else:
                p = process(binary)

        #p.recvuntil(END_OF_MENU)
        launch_attack(p)

        # Make sure that this is kept for gdb
        # p.interactive()

if __name__ == "__main__":
        main()
```

## 4 Memory Layout of Different Bins

The following figures illustrate the memory layout (64 bits system) of TCache bins, fast bins, unsorted bins, small bins, and large bins.

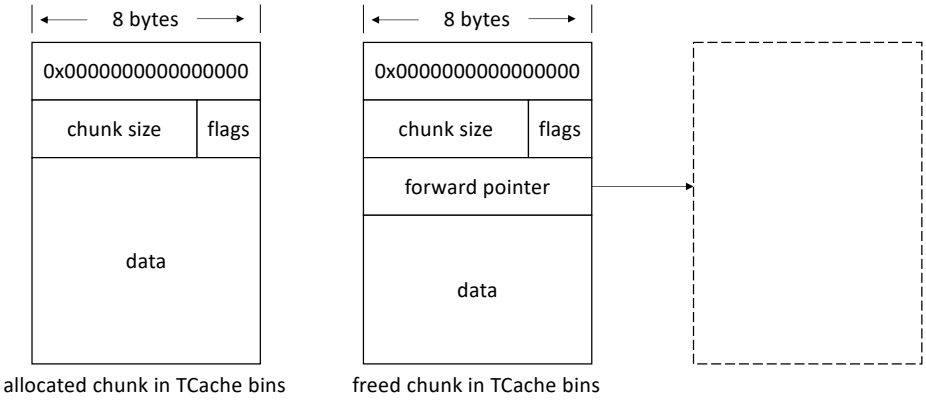


Figure 1: Allocated chunks and freed chunks in TCache bins

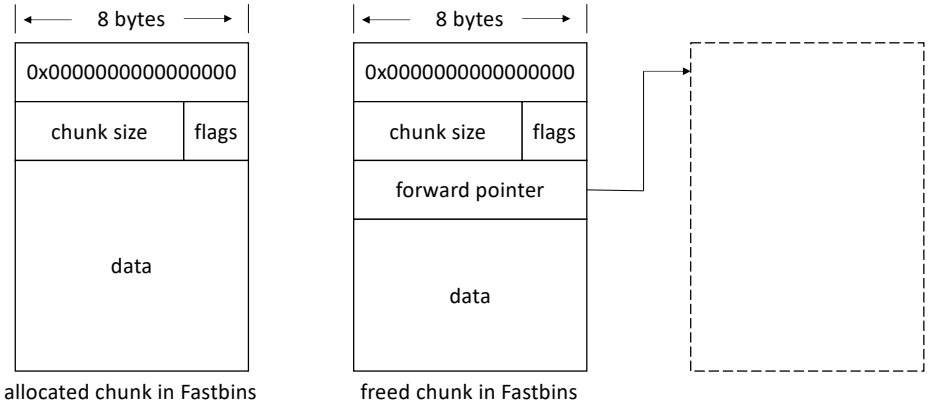


Figure 2: Allocated chunks and freed chunks in fast bins

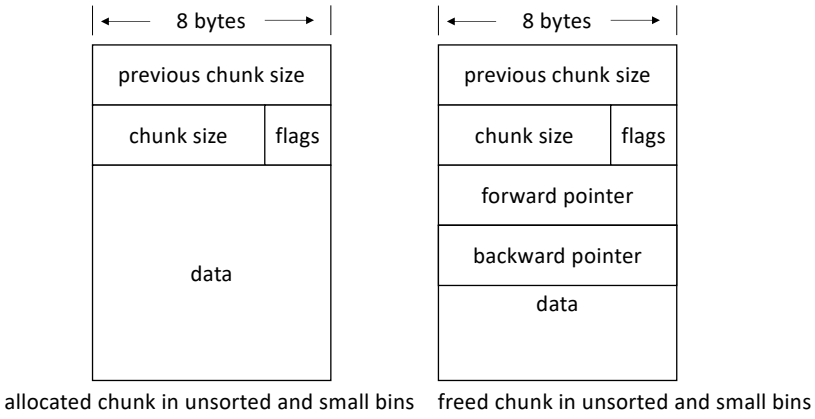


Figure 3: Allocated chunks and freed chunks in unsorted bins and small bins

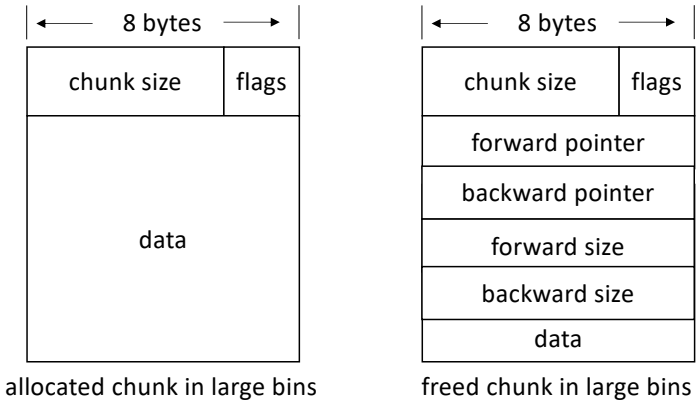


Figure 4: Allocated chunks and freed chunks in large bins

## 5 Deliverables

The flags you captured, and please submit them in `https://cse5474.osuseclab.com`, and the writeup describing how you solve these challenges. **Please also attach your code in your write up, and package them together**.