Final for CS 419— Computer Security

1 Heap overflow

Alice wrote a heap management library, and she uses the following data structures to manage heap:

```
struct  heap_sector {
    int size;                   // size of the heap sector
    char *sector;               // pointer to the first char
    struct heap_sector *prev;   // pointer to previous sector
    struct heap_sector *next;   // pointer to next sector
};

struct heap_sector *heap_free;  // pointer to the list of free heap memory
struct heap_sector *heap_used;  // pointer to the list of used heap memory
```

And use the following logic to manage memory:

- `heap_used` is initialized as `NULL`, and `heap_free` is initialised to the first available byte;
- `malloc(int n)` function always allocates the first a few memory sectors whose size is larger than `sizeof(struct heap_sector) + 1` (until the total size equals to `n`) to user process, and moves the memory from `heap_free` to `heap_used`; if there is no enough space to allocate, it returns `NULL`;
- `free(char* h)` will free the corresponding heap and move the memory from `heap_used` to `heap_free`.

Try to answer the following questions:

- Please write the pseudo code for `malloc(int n)` and `free(char* h)`; (10 points)
- The design is potentially vulnerable to integer overflow. Please explain how this may happen, and how your pseudo code avoids this problem. (10 points)
- The design is potentially vulnerable to heap overflow. Please explain how it may happen and how your pseudo code avoids such problems. (10 points)
- The design is potentially vulnerable to off-by-one attack. Please explain how this attack may happen and how your pseudo code avoids such problems. (10 points)
- The design is potentially vulnerable to TOCTTOU problem. Please explain how it may happen and how your design avoids such problems. (10 points)

- Users may free the allocated memory twice or multiple times. Please explain how your design can prevent this. (5 points)

2 Linux audit

A lot of problems in Linux auditing are caused by the lack of application semantics in the log. For example, Firefox will not tell the kernel that which tab(s) or website(s) it is processing. Thus, Linux auditing can benefit a lot if applications can do this, i.e., actively let the kernel or auditing system know its executing status or semantics. You are asked to design a set of library calls, which 1) applications can use to actively tell the kernel its semantics; 2) kernel can capture such semantics and log them; 3) such semantics can be used to improve attack forensics (e.g., solving the dependence explosion problem).

- A list of library calls and their pseudo code (7 points);
- An example of how the added library can achieve the three mentioned goals. (8 points)

3 Adversarial examples

We know that adversarial examples can transfer from one model to another, but there is no guarantee on that. Suppose we have two target models, $\mathcal{M}$ and $\mathcal{F}$, which work on the same image classification task. For a normal image x, we want to generate an adversarial example, x' that can attack both models:

- If we can use FGSM method, how can we guarantee that x' can attack two models instead of only one? (5 points)
- Suppose we want to perform black box attack, meaning that we can query the model to get an output label but do not know the model architecture, trained weights, training data and not even the number of output labels etc. How can we generate x'? (10 points))
- Going beyond models, if the system will first apply a set of unknown but commonly seen image pre-processing (e.g., white balancing, enhancement), how can we effectively generate adversarial examples in the black box setting? (15 points)