Name: Aishwarya Venkataraj My Id: av99869

CSCI8240 – Software Security & Cyber Forensics
Project 1: Dynamic Memory Overflow Detection

Part 1: Mark all user input data via stdin (gets and fgets) and command-line argument as "tainted".

Design: In order to mark all the user input data as tainted I created a separate function.

void markDataTainted(char *dest, int len)

The function is called inside the functions fgetsTail, getsTail and mainHead. The starting address of the data located in memory and the length of the input is passed as an argument to this function. In order to mark all the user data as tainted I created a TaintTable using Structure and used vector to add , propagate and modify the table. The structure of the Tainttable is as follows:

Tainted Address	Size
0xbffff3ec	20
0xbffff8ef	10

For any data input via fgets,gets and main its starting address and length is added into the table directly.

Reason: I used vector to modify the table because, vectors help in dynamic allocation of memory. As there is no fixed size for the table it is better to use vector to reduce the space complexity. The reason behind storing only the starting address of data and the size is also to reduce the space complexity. If we know the start address and length of the input we can find the contiguous memory addresses by using them.

Part 2 (30%): Monitor library calls to track how tainted data propagates (follow bytes through strcpy, strncpy, strcat, strncat, memcpy), and clear the mark when the byte is overwritten by non-tainted data (bzero, memset).

Design: In order to mark the input obtained from **strcpy** as tainted I designed a function.

void markDataTainted(**char** *dest,**char** *src,**int** len)

The destination address, source address and the length of the source is passed as argument. The length of the source will now become the length of the destination as well. In this function it first checks if the source address is tainted. If it is tainted the starting address of the destination and the length is added into the table. There is another case where the source data might not be tainted but the destination might already be in the tainted table. In that case the table is modified by changing the size of that destination and adding a new range if necessary. Another function is used to check the existence of the address in the table.

bool checktable(**char** *src)

The function returns true if the table has tainted data. The address to be checked is passed as the argument to the function. In this function with the source address it adds the size in order to get the final address and then check if the address we want to check lies in that range. In order to reduce the overhead I just check if the address lies withing that range

In order to mark the data obtained by **strncpy** and **memset** as tainted I used another function since it had many other cases to check.

void markDataTainted(**char** *dest,**char** *src, **char** *srcend, **int** len)

In this function we pass the destination , source starting and ending address and the size to be copied as the argument. In here each individual byte is checked to see if it is tainted based on the result the destination address is added into the table and the size is also altered based on the number of source bytes tainted.

In order to mark the data obtained by **strcat** and **strncat** as tainted I used another function since it had to change the size of the existing address in that table.

void catDataTainted(**char** *dest,**int** len)

The function takes as arguments the destination address and size to be concatanated.

It first checks the table for the destination address and if it exists it just adds the length and to size already in the table. Otherwise it adds the address and the length into the table.

In order to mark the data obtained by **bzero** and **memset** as tainted I used another function since it had to change the size of the existing address in that table.

void clearTaintedData(**char** *startaddr,**int** len)

This function changes the table to remove the address range from the table and also alters the size.

Part 3 : Detect when tainted data is used to change the program flow (return addresses, jump addresses, function pointers, etc.)

bool propagation(char *address)

This function checks if the return address is located in the tainted table and exits the program if the address is tainted.

It uses **PIN_ExitProcess(1)**; to exit the program.

Part 4: Store stack traces for each tainted byte to show how the byte is tainted and propagated

Design : In this part we get the stack traces for every byte using the IsCall() and InMainExecutable(). Using these functions we traverse each section in our image file, under each section we traverse each routine and under each routine we check every instruction. If the address is in our main executable file we add it to stack trace.

When a routine returns we use the IsRet() to pop back the address from the stack.

And while calling each function we add the stack trace of that function along with the entry of the memory address in our taint table.

This way it keeps the stack trace of each function and tainted byte.

Part 5: Monitor each instruction that can propagate tainted data

Design : In part 5, we monitor the adress flow between the memory and registers

```
if(INS_OperandCount(ins) > 1 && INS_OperandRead(ins, 1) &&
INS_OperandWritten(ins,0)){
```

```
if(INS_MemoryOperandIsRead(ins, 0) && INS_OperandIsReg(ins, 0))

else if(INS_MemoryOperandIsWritten(ins, 0) && INS_OperandIsReg(ins, 1))

else if(INS_OperandIsReg(ins, 0) && INS_OperandIsReg(ins, 1))
```

We use the following to instrument our function at instruction level to monitor the dataflow.
Case1(mem->reg): If memory is tainted we tainted the register, if not we check if reg is tainted and remove the register.
Case2(reg->mem): If register is tainted we tainted the memory, if not we check if momory is tainted and remove the memory.
Case 3(reg->reg: If reg1 is tainted we taint reg2, if not we remove the registerr from the taint table.

```
if (INS_OperandRead(ins, 0) && INS_OperandIsReg(ins,0))
        {
           REG reg = INS_OperandReg(ins,0);
           INS_InsertCall(ins,IPOINT_BEFORE, (AFUNPTR)branchreg,
                       IARG_INST_PTR,
                       IARG_BRANCH_TARGET_ADDR,
                       IARG_UINT32, reg,
                       IARG_END);
        }
```

I use the above function to check if each return register is tainted, when the return register is tainted The function exits using **PIN_ExitProcess(1)**;

Evaluation report:
Detection:I was not able to detect one of the attack testcases.
The stack trace of two attack programs was with little error.

False Positives: There was no false positives.

Name: Aishwarya Venkataraj My Id: av99869