ネットワークセキュリティ演習

7回 マルウェア

演習レポートのURL

https://goo.gl/forms/1JonUVMRaq9wECkt1

ASLR の無効化

```
sudo sysctl -w kernel.randomize_va_space=0
```

★必ず最後に有効化する

有効化の方法

```
sudo sysctl -w kernel.randomize_va_space=2
```

C言語コンパイラのインストール

```
sudo apt install gcc
```

```
sudo apt install clang
```

C言語のプログラム入門

最初のCプログラム

hello.c

```
nano hello.c
```

```c
#include <stdio.h>
int main(void){
  puts("hello");
  return 0;
}
```

コンパイル

```
gcc hello.c
```

```
ls
a.out  hello.c
```

コンパイルした実行ファイルの実行

```
./a.out
ハロー
```

実行ファイル名付きでコンパイル

実行ファイル名を ‘hello’ とする

```
gcc -o hello hello.c
```

コンパイルした実行ファイルの実行

```
./hello
ハロー
```

コマンドライン引数

kodame.c

```
nano kodama.c
```

```c
#include <stdio.h>
int main(int argc, char *argv[]){
  puts(argv[1]);
  return 0;
}
```

コンパイル

```
gcc kodama.c
```

実行

```
$ ./a.out hello
hello
```

アドレス演算子　&

addr.c

```
nano addr.c
```

```c
#include<stdio.h>
int main(){
  int a = 0;
  printf("address=%p\n",&a);
  return 0;
}
```

コンパイル

```
gcc addr.c
```

実行 (実行結果のアドレスは異なる)

```
./a.out

address=0x7fffffffe634
```

間接演算子 *

```
nano pointer.c
```

C

```c
include<stdio.h>
int main(){
    int a = 0;
    int *ip; /*整数へのポインタ*/
    ip=&a;  /*変数ipにはアドレスが格納される*/
    printf("a=%d\n", *ip);
    printf("ip=%p\n", ip);
    return 0;
}
```

```
gcc pointer.c
```

実行結果

```
$ ./a.out
a=0
ip=0x7fffffffe62c
```

間接演算子によるデータへのアクセス

```
nano pointer2.c
```

C

```c
#include<stdio.h>
int main(){
  int a = 0;
  int b = 1;
  int *ip; /*整数へのポインタ*/
  ip=&a;
  printf("*ip=%d\n", *ip);
  ip=&b;
  printf("*ip=%d\n", *ip);
  return 0;
}
```

コンパイル

```
gcc pointer2.c
```

実行

```
./a.out
*ip=0
*ip=1
```

ポインタのポインタ

```
nano pp.c
```

```C
#include<stdio.h>
int main(){
  int a = 0;
  int *ip; /*整数へのポインタ*/
  int **ipp; /*整数へのポインタのポインタ*/
  ip=&a;
  ipp=&ip;
  printf("**ipp=%d\n", **ipp);
  return 0;
}
```

コンパイル

```
gcc pp.c
```

実行

```
./a.out
**ipp=0
```

C言語の配列

```
nano array.c
```

```c
#include <stdio.h>
int main(){
  int a[3];
  a[0]=10;
  a[1]=20;
  a[2]=30;
  printf("a[0]=%d,a[1]=%d,a[2]=%d\n",a[0],a[1],a[2]);
  return 0;
}
```

コンパイル

```
gcc array.c
```

実行

```
./a.out
a[0]=10,a[1]=20,a[2]=30
```

ポインタとしての配列

```
nano parray.c
```

```c
#include <stdio.h>
int main(){
  int a[3];
  int *pa;
  pa=&a[0];
  a[0]=10;
  a[1]=20;
  a[2]=30;
  printf("*pa=%d\n",*pa);
  return 0;
}
```

コンパイル

```
gcc parray.c
```

実行

```
./a.out
*pa=10
```

```
nano parray2.c
```

```C
#include <stdio.h>
int main(){
  int a[3];
  int *pa;
  pa=a;
  a[0]=10;
  a[1]=20;
  a[2]=30;
  printf("*pa=%d\n",*pa);
  return 0;
}
```

コンパイル

```
gcc parray2.c
```

実行

```
./a.out
*pa=10
```

C言語の文字列は文字の配列

```
nano chars.c
```

```C
#include <stdio.h>
int main(){
  char *aisatu;
  aisatu="hello";
  puts(aisatu);
  return 0;
}
```

コンパイル

```
gcc chars.c
```

実行

```
./a.out
hello
```

strcpy関数による文字列のコピー

```
nano stringcpy.c
```

```C
#include <string.h>
int main(){
  char *aisatu;
  char greeting[8];
  aisatu="hello";
  strcpy(greeting, aisatu);
  puts(greeting);
  return 0;
}
```

コンパイル

```
gcc stringcpy.c
```

実行

```
./a.out
hello
```

strcpy関数のコピー先サイズが小さい場合

```
nano stringcpy2.c
```

```c
#include <stdio.h>
#include <string.h>
int main(){
  char *aisatu;
  char greeting[3];
  aisatu="helloween";
  strcpy(greeting, aisatu);
  puts(greeting);
  return 0;
}
```

コンパイル

```
gcc stringcpy2.c
```

実行

```
./a.out
helloween
*** stack smashing detected ***: <unknown> terminated
中止 (コアダンプ)
```

コマンドライン引数

```
nano kodama.c
```

```c
#include <stdio.h>
int main(int argc, char *argv[]){
  puts(argv[1]);
  return 0;
}
```

コンパイル

```
gcc kodama.c
```

実行

```
./a.out hello
hello
```

コマンドライン引数の配列

```
nano args.c
```

```C
#include <stdio.h>
int main(int argc, char *argv[]){
  printf("argc: %d\n",argc);
  for(int i; i<argc ; ++i){
    printf("argv: %s\n",argv[i]);
  }
}
```

コンパイル

```
gcc args.c
```

実行

```
./a.out aaa bbb
argc: 3
argv: ./a.out
argv: aaa
argv: bbb
```

コマンドライン引数で入力した文字列のコピー

```
nano kodama2.c
```

```C
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]){
  char buffer[6];
  strcpy(buffer, argv[1]);
  puts(buffer);
  return 0;
}
```

コンパイル

```
gcc kodama2.c
```

実行

```
./a.out hello
hello

./a.out helloween
helloween
*** stack smashing detected ***: <unknown> terminated
中止 (コアダンプ)
```

関数呼び出し先で文字列をコピー

```
nano funccpy.c
```

C

```c
#include <stdio.h>
#include <string.h>
void kansu1(char *argv) {
  char buffer[6];
  strcpy(buffer, argv);
  puts(buffer);
}
int main(int argc, char *argv[]){
  kansu1(argv[1]);
  return 0;
}
```

コンパイル

```
gcc funccpy.c
```

実行

```
./a.out hello
hello
```

バッファーオーバーフロー攻撃

```
nano bof.c
```

```c
#include <stdio.h>
#include <string.h>
void kansu1(char *argv) {
  char buffer[6];
  strcpy(buffer, argv);
  puts(buffer);
}
int main(int argc, char *argv[]){
  kansu1(argv[1]);
  return 0;
}
```

コンパイル

```
gcc bof.c
```

実行

```
./a.out hello
hello

./a.out helloween
helloween
*** stack smashing detected ***: <unknown> terminated
中止 (コアダンプ)
```

書式文字列攻撃

```
nano format.c
```

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[]){
  char buffer[100];
  strcpy(buffer, argv[1]);
  printf(buffer);
  return 0;
}
```

コンパイル

★ウォーニングが出るが無視する

```
gcc format.c

format.c: In function ‘main’:
format.c:6:10: warning: format not a string literal and no format arguments [-Wformat-
   printf(buffer);
          ^~~~~~
```

実行

```
./a.out 'AAAA %x %x %x %x %x %x %x %x %x'
AAAA ffffe9d0 78252078 78252078 f7dd0d80 f7dd0d80 ffffe6f8 0 41414141 25207825
```

Aの文字コードは 41 なので、AAAA の格納位置が　41414141 と出てくる

ASLR の有効化

★必ず最後に有効化する

有効化の方法

```
sudo sysctl -w kernel.randomize_va_space=2
```