

セキュアコーディングノススメ

2008年1月24日(木) @
JANOG21 Kumamoto

JPCERT コーディネーションセンター
情報セキュリティアナリスト
戸田 洋三
久保 正樹

セキュアコーディング・クイズ！ の回答と解説

```
size_t strlen(const char *s) {
    const char *p = s;
    while (*p != '\0')
        p++;
    return p - s;
}

char source[10];
strcpy(source, "0123456789");
char *dest = (char *)malloc(strlen(source));
for (int i=1, j=0; i<=11; i++) {
    dest[i] = source[j];
    j++;
}
dest[i] = '\0';
printf("dest = %s", dest);
}

char input[] = "bogus@addr.com; cat /etc/passwd";
string email;
stringIterator loc = email.begin();
// /* \0 * \0
if (input[i] != '\0') {
    email.insert(loc++, input[i]);
}
else {
    email.insert(loc++, '\0');
}
}

/* NTFS size_t strcpy(char *dst,
size_t strlen(const char *src,
const char *src, size_t
size_t strlen(const char *src,
size_t strlen(const char *src,
const char *src, size_t

ar Char_str[100];
unsigned char uc_str[] = "hello world";
wchar_t wc_str[20] = L"hi";
void func(char *s) {
    size_t size = sizeof(s) / sizeof(s[0]);
}

int main(void) {
    char str[] = "Bring on the dancing queen";
    size_t size = sizeof(str) / sizeof(char);
    printf("%s\n", str);
}

int main(int argc, char *argv[]) {
    char *buff = (char *)malloc(strlen(argv[1]));
    if (buff != NULL) {
        strcpy(buff, argv[1]);
        printf("argv[1] = %s\n", buff);
    }
}
```

皆さん事前資料は見てきましたよね？！

脆弱性のトレンド、いまなぜセキユアコーディングか、
については、事前資料のスライド11, 20, 21あたりを
見といてね！

クイズの答えを解説します！

全問正解の人にはすてきな景品が！

問題1 sizeof() の不適切な使用はどこ？

```
// ...
if (input[i] != '\0') {
    email.insert(loc++, input[i]);
}
else {
    email.insert(loc++, '\0');
}
} // NTBS_size_t_strlen(char *str, ...)
```

```
void func(char *s) {
    size_t size = sizeof(s) / sizeof(s[0]);
}
```

```
int main(void) {
    char str[] = "Bring on the dancing horses";
    size_t size = sizeof(str) / sizeof(str[0]);
    func(str);
}
```

回答1 sizeof() の不適切な使用はどこ？

```
// ...
if (input[i] != '\n') {
    email.insert(loc++, input[i]);
}
else {
    email.insert(loc++, '\n');
}
} // NTBS_size_t_strlen(char *dst, ...)
```

```
void func(char **s) {
    size_t size = sizeof(s) / sizeof(s[0]);
}
```

サイズは 4

サイズは 1

```
int main(void) {
    char str[] = "Bring on the dancing horses";
    size_t size = sizeof(str) / sizeof(str[0]);
    func(str);
}
```

サイズは 28

サイズは 1

問題2 オフバイワンエラーが発生するのはどこ？

```
int main(void) {
    char source[10];
    strcpy(source, "0123456789");
    char *dest = malloc(strlen(source));
    for (int i=1; i <= 11; i++) {
        dest[i] = source[i];
    }
    dest[i] = '¥0';
    printf("dest = %s", dest);
}
```

回答2 オフバイワンエラーが発生するのはどこ？

```
int main(void) {  
    char source[10];  
    strcpy(source, "0123456789");  
    char *dest = malloc(strlen(source));  
    for (int i=1; i <= 11; i++) {  
        dest[i] = source[i];  
    }  
    dest[i] = '¥0';  
    printf("dest = %s", dest);  
}
```

サイズが10の配列に対して、10文字+NULL
終端の11文字をコピーしようとしている！

カウンタが1からスタート
するのがおかしい！

i が 11 のときに境界外参
照が発生！

更に、境界外書き込みも行っ
てしまう！

問題3 境界外へ書き込みが行われる可能性があるのはどこ？

Kerberos 5 Version 1.0.6 に実際にあった脆弱性

「

```
if (auth_sys == KRB5_RECVAUTH_V4) {
    strcat(cmdbuf, "/v4rcp");
} else {
    strcat(cmdbuf, "/rcp");
}
if (stat((char *)cmdbuf + offst, &s) >= 0)
    strcat(cmdbuf, cp);
else
    strcpy(cmdbuf, copy);
```


回答3 境界外へ書き込みが行われる可能性があるのはどこ？

```
if (auth_sys == KRB5_RECVAUTH_V4) {
    strcat(cmdbuf, "/v4rcp");
} else {
    strcat(cmdbuf, "/rcp");
}
if (stat((char *)cmdbuf + offst, &s) >= 0)
    strcat(cmdbuf, cp);
else
    strcpy(cmdbuf, copy);
```

cp のサイズによっては cmdbuf が
オーバーフローする

copy のサイズによっては cmdbuf
がオーバーフローする

回答3つづき 実際に修正された Kerberos 5 のコード

```
cmdbuf[sizeof(cmdbuf) - 1] = '¥0'
if (auth_sys == KRB5_RECVAUTH_V4) {
    strncat(cmdbuf, "/v4rcp", sizeof(cmdbuf) - 1 -
strlen(cmdbuf));
} else {
    strncat(cmdbuf, "/rcp", sizeof(cmdbuf) - 1 -
strlen(cmdbuf));
}
if (stat((char *)cmdbuf + offst, &s) >= 0)
    strncat(cmdbuf, cp, sizeof(cmdbuf) - 1 - strlen(cmdbuf));
else
    strncpy(cmdbuf, copy, sizeof(cmdbuf) - 1 -
strlen(cmdbuf));
```

問題4 puts() が実行されるのはどの行？

```
signed char x, y;
```

```
x = -128;
```

```
y = -x;
```

```
if (x == y) puts("1");
```

```
if ((x - y) == 0) puts("2");
```

```
if ((x + y) == 2 * x) puts("3");
```

```
if (((char)(-x) + x) != 0) puts("4");
```

```
if (x != -y) puts("5");
```

回答4 puts() が実行されるのはどの行？

```
signed char x, y;
```

```
x = -128;
```

```
y = -x;
```

式 $-x$ は整数拡張され128 (0...0100) となるが、signed char の幅に代入されると -128 として解釈される！

```
if (x == y) puts("1");
```

```
if ((x - y) == 0) puts("2");
```

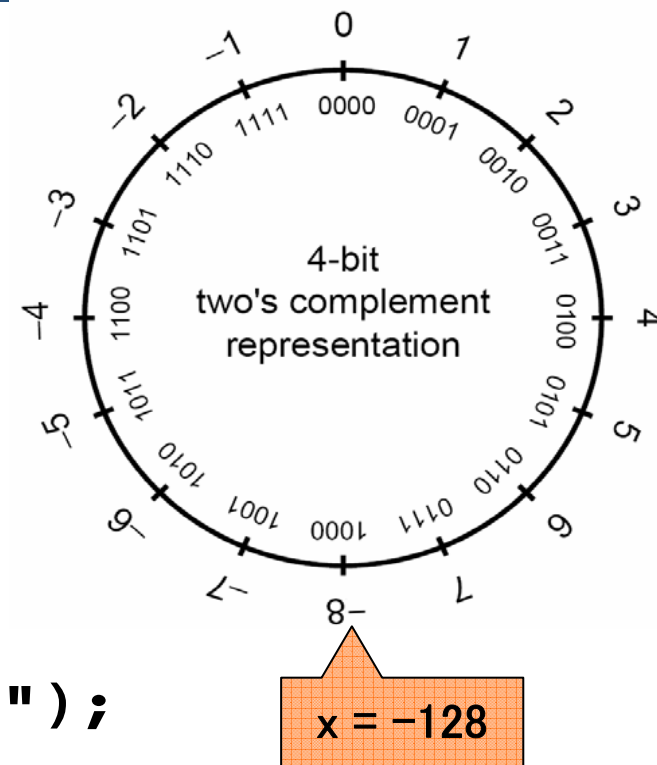
```
if ((x + y) == 2 * x) puts("3");
```

```
if (((char)(-x) + x) != 0) puts("4");
```

```
if (x != -y) puts("5");
```

演算 $!=$ を行うまえにオペランド x, y は整数拡張されるため、 $x = -128, y = 128$ として評価される。

char にキャストしているので、演算時に整数拡張(integer promotion)が発生しない！
つまり、 $-x$ は x と等しくなる。



まとめ

- **信頼できない入カソース**から入ってくる値は、以下の点をチェックする
 - 値のとりうる上限、下限は特定できるか
 - 可能な場合、入力インターフェイスから強制的にチェックを行う

- 整数の脆弱性を**回避する**には、

Cの仕様を**深〜く**理解しておかないとダメよ。

セキュアコーディングセミナー、トレーニングのご案内

出張セミナーやっています

- 企業・組織での研修にご利用ください

一般向けオープンセミナー

- 2008年に実施予定!

※写真は昨年 Robert C Seacord 氏を迎えたセミナーの様

詳しくはJPCERT/CCスタッフへお問合せください