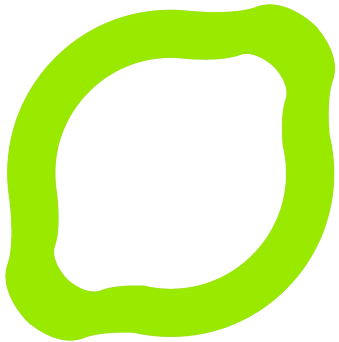
# C/C++ vs Security

Czyli jak strzelić sobie w stopę i nawet tego nie zauważyć

# O mnie

- Wicekapitan Dragon Sector
- Veles @ Codilime
- Analizuję losowe rzeczy, np: Reverse engineering Toshiba R100 BIOS



# Plan

- Skutki bugów
- Historyczne przykłady
- Ciekawe błędy:
  - C
  - C/C++
  - C++
- Przemyślenia

# Wstęp

- Programiści często nie są świadomi możliwych skutków błędów
- Naprawmy to ;)

# Przykład z życia nr 1: MySQL / MariaDB

```
my_bool
check_scramble(const char *scramble_arg, const char *message,
               const uint8 *hash_stage2)
{
  [...]
  return memcmp(hash_stage2, hash_stage2_reassured,
SHA1_HASH_SIZE);
}
```

# Przykład z życia nr 2: Ruby

```
int_pow(long x, unsigned long y)
{
    [...]
    long xz = x * z;
    if (!POSFIXABLE(xz) || xz / x != z) {
        goto bignum;
    }
    z = xz;
    [...]
}
```

# Przykład z życia nr 2: Ruby

```
>> 2 ** 63
=> -9223372036854775808
```

ups...

# Przykład z życia nr 2: Ruby

Jest błąd, trzeba naprawić!

```
* numeric.c (int_pow): make sure to assign the result of x * z.
  If xz is optimized out, the value won't overflow.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@31805 b2dd03c8-39d4-
```

# Przykład z życia nr 2: Ruby

```
int_pow(long x, unsigned long y)
{
    [...]
-   long xz = x * z;
+   volatile long xz = x * z;
    if (!POSFIXABLE(xz) || xz / x != z) {
        goto bignum;
    }
    z = xz;
    [...]
}
```

# Jak poprawnie sprawdzić OVF przy mnożeniu?

```
int a, b;
cin >> a >> b;
if (a > numeric_limits<decltype(b)>::max() / b)
   cout << "Overflow!" << endl;
else
   cout << a*b << endl;
```

# Przykład z życia nr 3: Linux kernel

```
static unsigned int tun_chr_poll(struct file *file, poll_table *
wait)
{
    struct tun_file *tfile = file->private_data;
    struct tun_struct *tun = __tun_get(tfile);
    struct sock *sk = tun->sk;
    unsigned int mask = 0;

    if (!tun)
        return POLLERR;
```

# Przykład z życia nr 4: OpenSSL

Heartbleed

# Ciekawe błędy: C

```c
char input[64];
scanf("%64s", input);
size_t len = strlen(input);
for (int i = 0; i < len/2; i++) {
    char tmp = input[i];
    input[i] = input[len - i - 1];
    input[len - i - 1] = tmp;
}
puts(input);
```

```c
char input[64];
fgets(input, 64, stdin);
if (input[strlen(input)-1] == '\n')
    input[strlen(input)-1] = 0;
size_t len = strlen(input);
for (int i = 0; i < len/2; i++) {
    char tmp = input[i];
    input[i] = input[len - i - 1];
    input[len - i - 1] = tmp;
}
puts(input);
```

# Ciekawe błędy: C/C++

```c
int num1, num2;
char op;
scanf("%d %c %d", &num1, &op, &num2);
switch (op)
{
   case '+': printf("%d\n", num1 + num2); break;
   case '-': printf("%d\n", num1 - num2); break;
   case '*': printf("%d\n", num1 * num2); break;
   case '/': printf("%d\n", num1 / num2); break;
}
```

```c
int num1, num2;
char op;
scanf("%d %c %d", &num1, &op, &num2);
switch (op)
{
   case '+': printf("%d\n", num1 + num2); break;
   case '-': printf("%d\n", num1 - num2); break;
   case '*': printf("%d\n", num1 * num2); break;
   case '/':
   if (num2) printf("%d\n", num1 / num2); break;
}
```

```
while (pos < input_len)
{
    size_t chunk_len =
       min(rand() % 8 + 2, input_len - pos);
    print_data(input + pos, chunk_len);
    printf(" ");
    pos += chunk_len;
}
```

```c
const int QUOTES_CNT = 5;
const char* quotes[QUOTES_CNT] = {"I am", "too lazy",
    "to put", "some interesting", "quotes here."};

int hash(const char* str);

int main()
{
    char input[32];
    printf("Your name: ");
    scanf("%31s", input);
    puts(quotes[abs(hash(input)) % QUOTES_CNT]);
    return 0;
}
```

```
unsigned input, rot;
cin >> input >> rot;

cout << ((input << rot) + (input >>
(sizeof(input)*8 - rot))) << endl;
```

# Ciekawe błędy: C++

```
std::vector<double> array;

[...] // Read user-controlled data into `array`

std::sort(array.begin(), array.end());
```

```
std::vector<double> array;

[...] // Read user-controlled data into `array`

for (auto x : array)
    if (x == NAN)
        exit(0);
std::sort(array.begin(), array.end());
```

```
size_t count;
cin >> count;
auto buf = new int[count];
for (int* it = buf; it < &buf[count]; it++)
   *it = rand();
sort(buf, buf+count);
for (int* it = buf; it < &buf[count]; it++)
   cout << *it << " ";
cout << endl;
```

# Czego nie było:

- Klasycznego buffer overflow
- Zarządzania pamięcią
- Bariery, reordering
- Wielowątkowości

# Przyczyny błędów

# Przyczyny błędów

- C i C++ są skomplikowane, łatwo o subtelne błędy
- Nawet zwykłe liczby są trudne
- Wiele rzeczy bardzo ciężko zaimplementować bezpiecznie w C/C++
- Brak możliwości pełnej weryfikacji kodu
- Ciężko automatycznie znaleźć potencjalnie groźne miejsca

# Co dalej?

# Co dalej?

- C++ Core Guidelines?
- Półśrodki:
  - analiza statyczna
  - testy + sanitizery
  - hardening przy kompilacji
- Inne języki?