When securing C++ code, use C++ solutions

YAIR FRIEDMAN

Pro

Buffer overflow - Problem

- CWE-121

```
#define BUFSIZE 256
int main(int argc, char **argv) {
  char buf[BUFSIZE];
  strcpy(buf, argv[1]);
}
```

Buffer overflow – C way

```
#define BUFSIZE 256
int main(int argc, char **argv) {
  char buf[BUFSIZE];
  strcpy_s(buf, sizeof(buf), argv[1]);
}
```

Buffer overflow – C++ way

```
int main(int argc, char **argv) {
  std::string buf{argv[1]};
}
```

- Or even

```
int main(int argc, char **argv) {
  std::string_view view{argv[1]};  // Non-owning “view”
}
```

Format strings and off-by-one – Problem

- CWE-193

```
char lastname[20];
char firstname[20];
char name[40];
char fullname[40];

strncat(name, firstname, sizeof(name));
strncat(name, lastname, sizeof(name));
snprintf(fullname, sizeof(fullname), "%s", name);
```

Format strings and off-by-one – C way

```
char lastname[20];
char firstname[20];
char name[40];
char fullname[40];

strncat_s(name, sizeof(name), firstname, sizeof(name)-1);
strncat_s(name, sizeof(name), lastname, sizeof(name)-1);
snprintf_s(fullname, sizeof(fullname), _TRUNCATE, "%s", name);
```

Format strings and off-by-one – C++ way

```
std::string lastname;
std::string firstname;
std::string name;
std::string fullname;
name = firstname + lastname;
stringstream ss(firstname); ss << lastname; name = ss.str();
name = absl::StrCat(firstname, lastname);
fullname = name.substr(40);
```

Integer Overflow - Problem

- CWE-190 (Real OpenSSH example)

```
nresp = packet_get_int();
if (nresp > 0) {
  response = malloc(nresp*sizeof(char*));
  for (i = 0; i < nresp; i++)
    response[i] = packet_get_string(NULL);
}
```

Integer Overflow – C way

- From CERT INT32-C

```
int nresp = packet_get_int();
if (nresp > 0) {
  long long tmp = (long long)nresp * (long long)sizeof(char*);
  if ((tmp > INT_MAX) || (tmp < INT_MIN)) {
    /* Handle error */
  }
  response = malloc((int)tmp);
  …
}
```

Integer Overflow – C way

- From CERT INT32-C

```
void f(int si_a, int si_b) {
  int sum;
  if (((si_b > 0) && (si_a > (INT_MAX - si_b))) ||
      ((si_b < 0) && (si_a < (INT_MIN - si_b)))) {
    /* Handle error */
  } else {
    sum = si_a + si_b;
  }
  /* ... */
}
```

Integer Overflow – C++ way

```
safe<int> nresp = packet_get_int();
if (nresp > 0) {
  safe<size_t> tmp = nresp * sizeof(char*);
  response = new unsigned char*[tmp];// but don’t do that
  for (i = 0; i < nresp; i++)
    response[i] = packet_get_string(NULL);
}
```

Resource Handling - Problem

```
void my_resource_hungry_function()
{
  resource1_t r1 = get_resource1();
  resource2_t r2;
  r2 = get_resource2(r1);
  if (r2.ok())          // let's check if we got resource 2
  {
    // do something resource consuming
    ...
```

Resource Handling - Problem

```
// now we need resource 3
resource3_t r3 = get_resource3();
if (!r3.ok())// if we didn't get r3 we must exit prematurely
{
  // cannot continue, lets remember to release r1 and r2
  r2.release();    // r2 needs r1 so we release it first
  r1.release();
  return;
}
```

Resource Handling - Problem

```
// do even more stuff
...
if (some_weird_condition)
{
  // do something here
  // end early, must free resources
  r3.release(); r2.release(); r1.release(); return;
}
```

Resource Handling - Problem

```
        // more stuff here
        ...
        r3.release(); r2.release();
    }
    // do we have valid r1 here?
    r1.release();
}
```

Resource Handling – Bird's-eye view

```
void my_resource_hungry_function()
{
  resource1_t r1 = get_resource1();
  resource2_t r2;
  r2 = get_resource2(r1);
  if (r2.ok())            // let's check if we got resource 2
  {
    // do something resource consuming
    ...
    // now we need resource 3
    resource3_t r3 = get_resource3();
    if (!r3.ok())      // if we didn't get r3 we must exit prematurel
    {
      // cannot continue, lets remember to release r1 and
      r2.release();    // r2 needs r1 so we release it
      r1.release();
      return;
    }
    // do even more stuff
    ...
    if (some_weird_condition)
    {
      // do something here
      // end early, must free resources
      r3.release(); r2.release(); r1.release()
    }

    // more stuff here
    ...
    r3.release(); r2.release();
  }
  // do we have valid r1 here?
  r1.release();
}
```

Resource Handling – C way

- go**

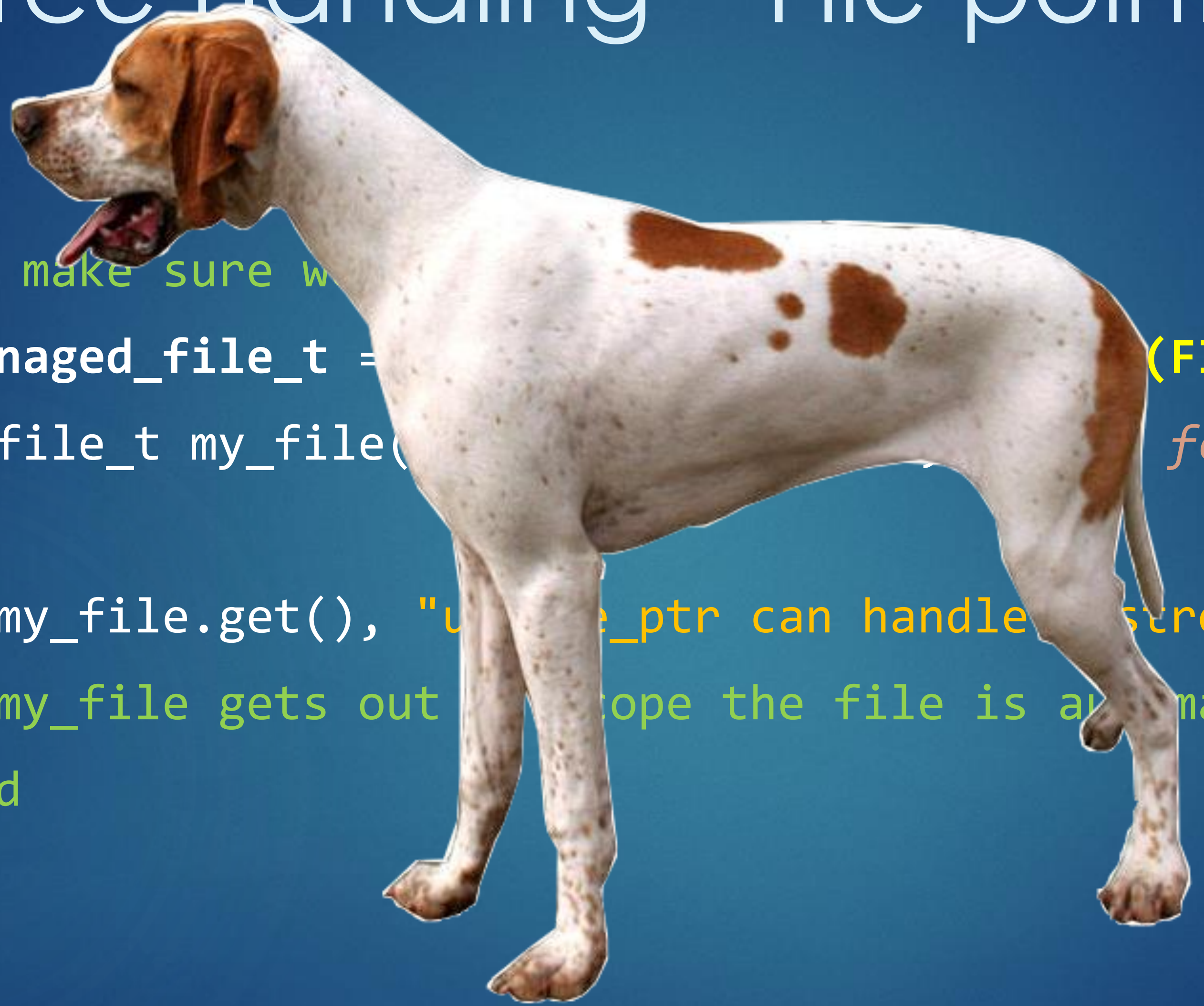
Resource Handling – C++ Way

```
void my_resource_hungry_function()
{
  unique_ptr<resource1_t> r1 = make_unique<resource1_t>();
  unique_ptr<resource2_t> r2 = make_unique<resource2_t>(r1.get());
  if (r2->ok())           // let's check if we got resource 2
  {
    // do something resource consuming
    ...
    // now we need resource 3
    unique_ptr<resource3_t> r3 = make_unique<resource3_t>();
    if (!r3->ok())       // if we didn't get r3 we must exit prematurely
      return;
    // do even more stuff
    ...
    if (some_weird_condition)
    {
      // do something here
      // end early, must free resources
      return;
    }
    // more stuff here
  }
}
```

Memory Handling – C++ way

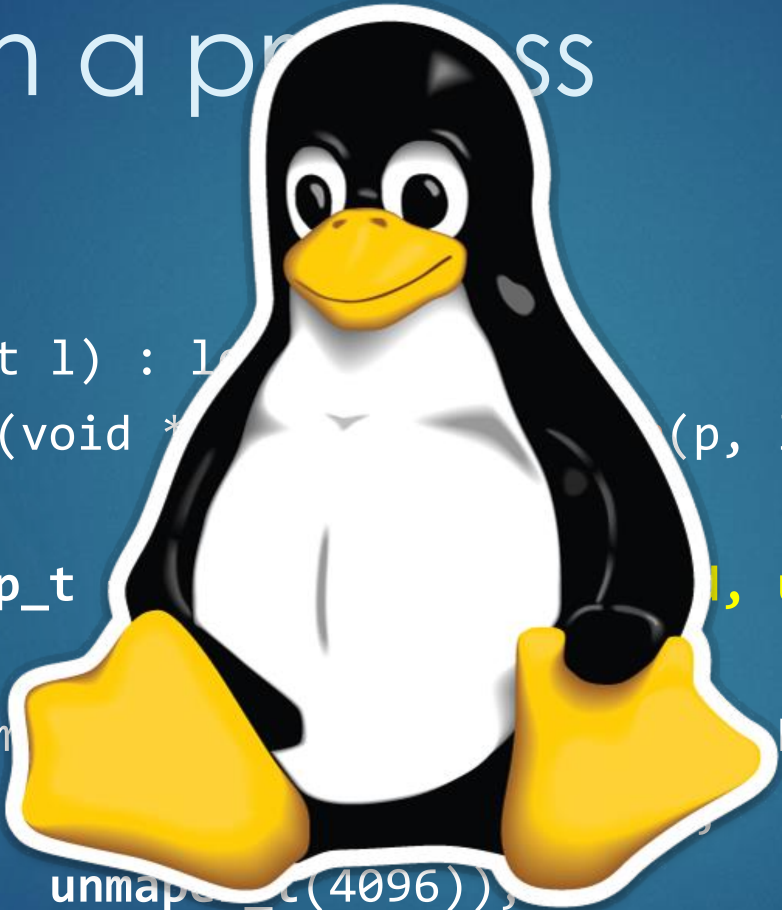
```
class MyClass {
 MyClass(int n) {…}
    …
}
…
{
  // instead of MyClass* myClassP = new MyClass(42); use
  unique_ptr<MyClass> myClassP = make_unique<MyClass>(42);
  // use myClassP as a normal pointer to MyClass object
} // No need to call delete(myClass) – done automatically!
```

Resource Handling – File pointers



```
// let's make sure w
using managed_file_t =                                 (FILE *)->int>;
managed_file_t my_file(                               fclose);
...
fprintf(my_file.get(), "u      _ptr can handle  streams!\n");
// When my_file gets out      cope the file is au  matically
// closed
```

Resource Handling – mapping pages from a pr ss



```
struct unmaper_t
{
  unmaper_t(size_t l) : l
  void operator()(void *                         (p, length); }
};
using managed_mmap_t                            , unmaper_t>;
{
  managed_mmap_t m                               PROT_WRITE,

                 unma    t(4096));
} // munmap called automagically from unique_ptr
```

Resource Handling – HMODULE handle

```
struct hmodule_deleter
{
  using pointer = HMOD                                   the pointer type
  void operator()(poin                                   rary(dllHandle); }
};

{
  std::unique_ptr<HMOD                                   dLibrary(libPath));
  ...
} // FreeLibrary called automagically from unique_ptr
```

Summary

- Don’t use `char[]` for st `:string_view`
- Don’t use format string dle strings. Use standard library opera ries to get the necessary effect
- Use libraries to do safe
- Use smart pointers to m and others