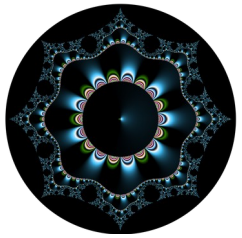
Sécurité des applications

Gestion de la Mémoire

Thibaut et Corinne HENIN



www.arsouyes.org
@arsouyes

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

Sommaire

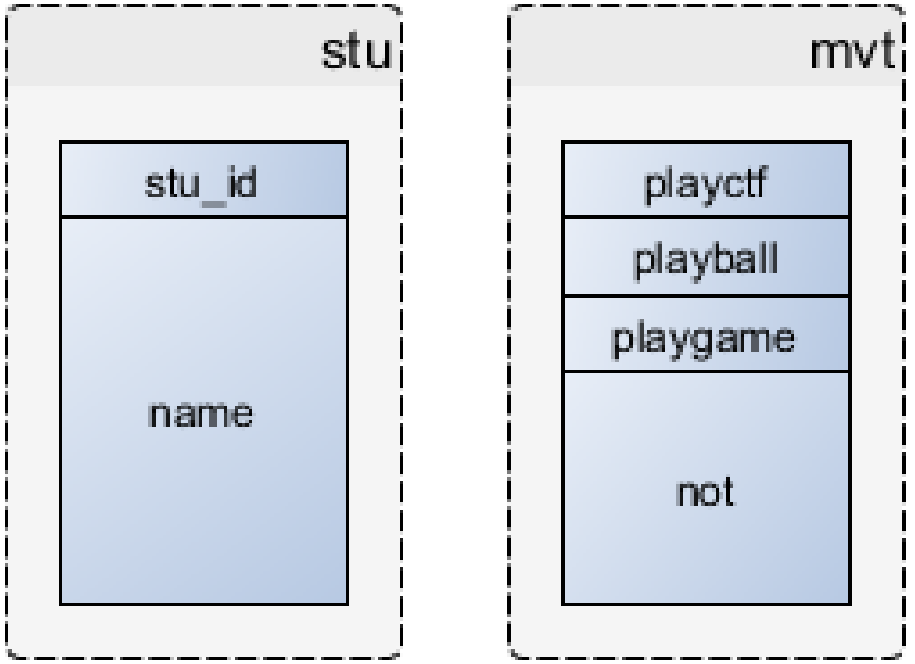
- **Mémoire**
  - Use after free
  - Double free
  - Null pointer dereference
  - Uninitialize Local Variable
- **Objets**
  - Deserialization
  - Finalize attack
  - Bad Cast

Mémoire

« Use after free »

Avant de commencer

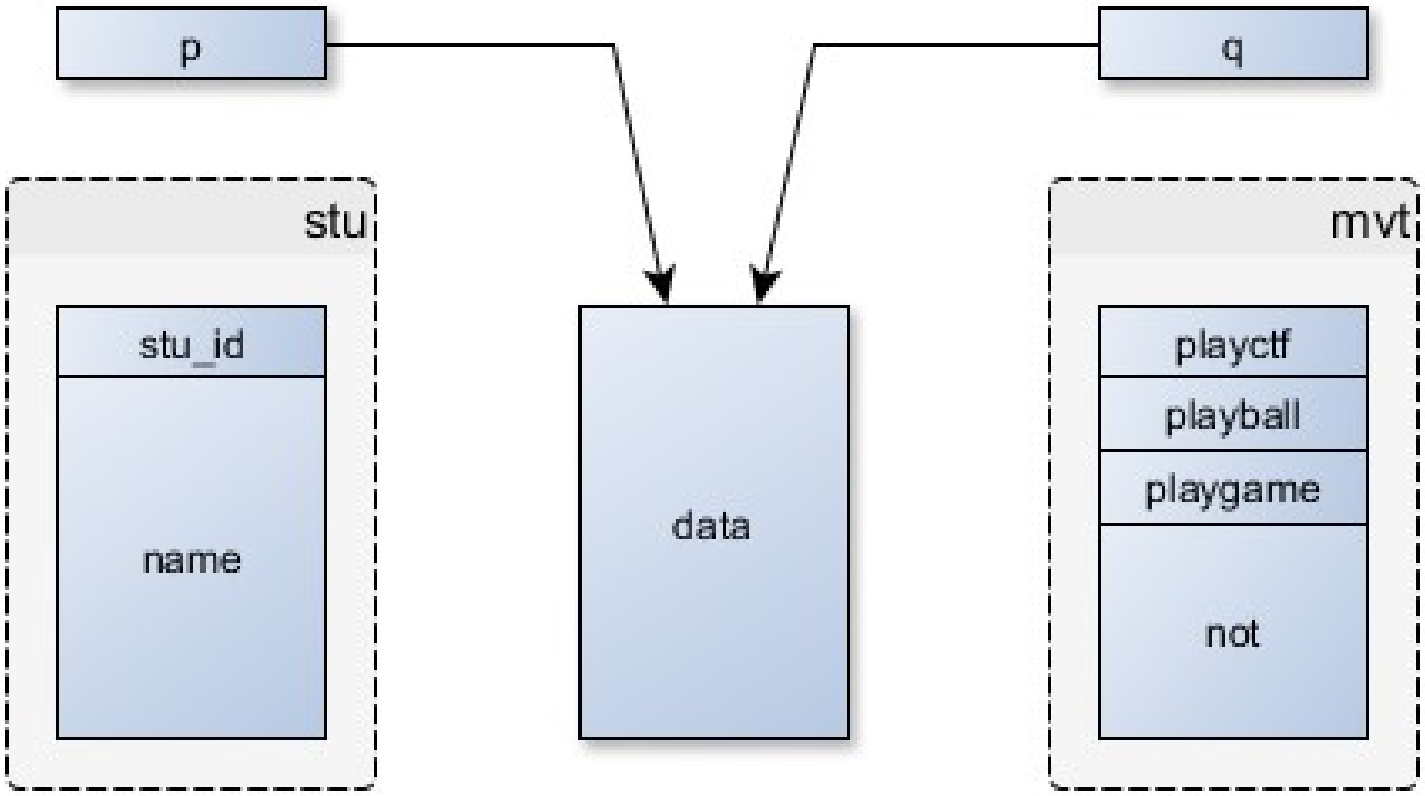
```
typedef struct stu {
   int stu_id ;
   char name[20] ;
} stu_t ;

typedef struct mvt {
   void (* playctf)() ;
   void (* playball)() ;
   void (* playgame)() ;
   char not[12] ;
} mvt_t ;
```



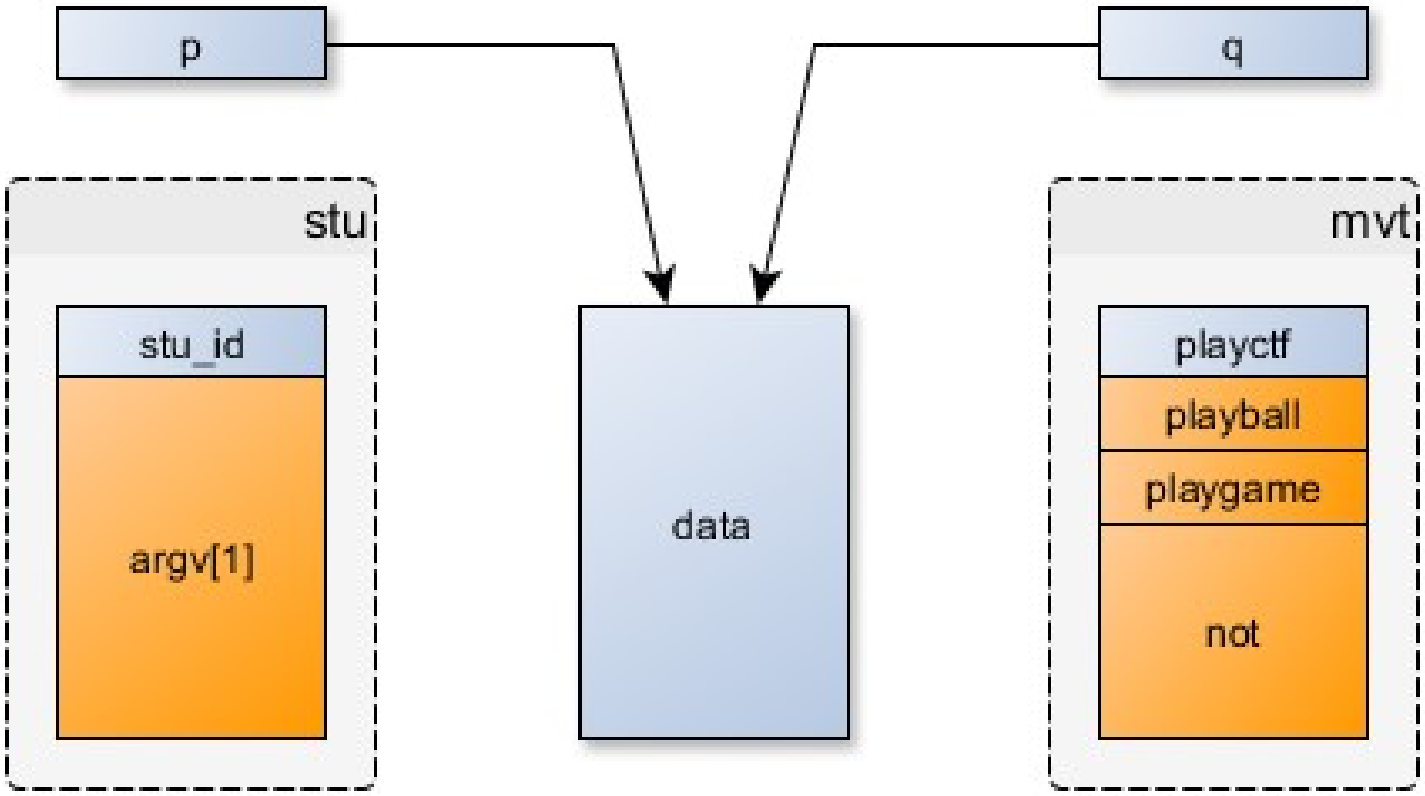
Code problématique

```
int main(int argc, char ** argv) {

  stu_t * p = (stu_t *) malloc(sizeof(stu_t)) ;
  free(p) ;
  mvt_q * q = (mvt_t *) malloc(sizeof(mvt_t)) ;
  initMvt(q) ;

  strncpy(p->name, argv[1], 20) ;

  q->playball() ;

  free(q) ;
}
```

État de la mémoire



État de la mémoire



Protection

- Good Design
  - *i.e.* RAII Pattern
- Analyse statique
- Analyse dynamique
  - *i.e.* valgrind

« Double Free »

Code problématique

```
int function() {
  stu_t *a, *b, *c ;
  int res = 0 ;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  /*
   * Do stuffs
   */

  free(a) ;  free(b) ; free(a) ;
  return res ;
}
```

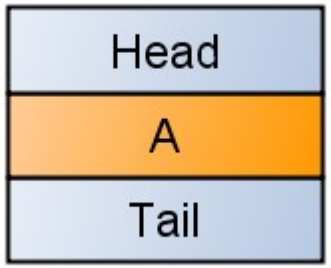
```
int isAdmin(int id) {
  stu_t *a, *b, *c ;
  int res = 0;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  a->id = 0 ;
  c->id = id ;
  res = a->id == c->id ;

  free(a) ; free(b) ; free(c) ;
  return res ;
}
```

Structure « fastbin »

```
int function() {
  stu_t *a, *b, *c ;
  int res = 0 ;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  /*
   * Do stuffs
   */

  free(a) ;  free(b) ; free(a) ;
  return res ;
}
```

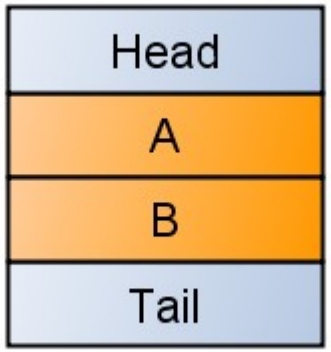
Structure « fastbin »

```
int function() {
  stu_t *a, *b, *c ;
  int res = 0 ;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  /*
   * Do stuffs
   */

  free(a) ;  free(b) ; free(a) ;
  return res ;
}
```



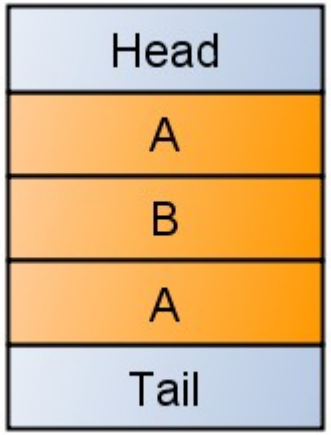
Structure « fastbin »

```
int function() {
  stu_t *a, *b, *c ;
  int res = 0 ;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  /*
   * Do stuffs
   */

  free(a) ;  free(b) ; free(a) ;
  return res ;
}
```

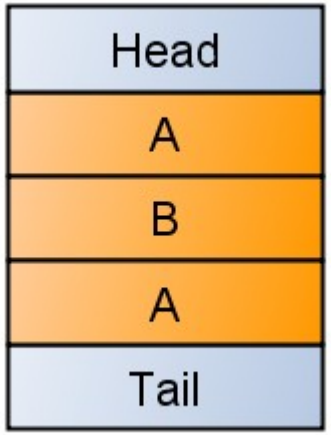


Structure « fastbin »

```
int function() {
  stu_t *a, *b, *c ;
  int res = 0 ;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  /*
   * Do stuffs
   */

  free(a) ;  free(b) ; free(a) ;
  return res ;
}
```

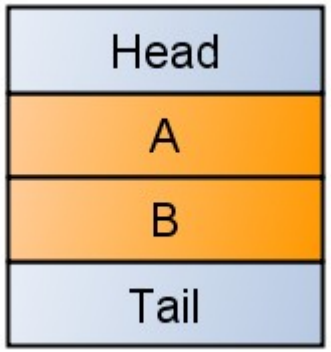


Recyclage pour malloc



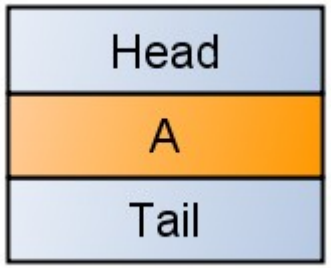

```
int isAdmin(int id) {
  stu_t *a, *b, *c ;
  int res = 0;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  a->id = 0 ;
  c->id = id ;
  res = a->id == c->id ;

  free(a) ; free(b) ; free(c) ;
  return res ;
}
```

Recyclage pour malloc




```
int isAdmin(int id) {
  stu_t *a, *b, *c ;
  int res = 0;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  a->id = 0 ;
  c->id = id ;
  res = a->id == c->id ;

  free(a) ; free(b) ; free(c) ;
  return res ;
}
```

Recyclage pour malloc




```
int isAdmin(int id) {
  stu_t *a, *b, *c ;
  int res = 0;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  a->id = 0 ;
  c->id = id ;
  res = a->id == c->id ;

  free(a) ; free(b) ; free(c) ;
  return res ;
}
```

Recyclage pour malloc


```
int isAdmin(int id) {
  stu_t *a, *b, *c ;
  int res = 0;

  a = (stu_t *) malloc(sizeof(stu_t)) ;
  b = (stu_t *) malloc(sizeof(stu_t)) ;
  c = (stu_t *) malloc(sizeof(stu_t)) ;

  a->id = 0 ;
  c->id = id ;
  res = a->id == c->id ;

  free(a) ; free(b) ; free(c) ;
  return res ;
}
```

Protection

- Good Design
  - *i.e.* RAII Pattern
- Analyse statique
- Analyse dynamique
  - *i.e.* valgrind

Null Pointer dereferences

Null pointers

```
int tick(handler * h, void * data) {
  cb_t * cb = h->cb ;
  return cb(data) ;
}
```

- cb = NULL ?

Null pointers

```
int tick(handler * h, void * data) {
  cb_t * cb = h->cb ;
  return cb(data) ;
}
```

```
#include <sys/mman.h>

void fillNULL()
{
    mmap(0, 4096,
        PROT_READ | PROT_WRITE,
        MAP_PRIVATE | MAP_ANONYMOUS |
MAP_FIXED,
        -1, 0);
    // ...
}
```

Null pointers

- Attaquant alloue la page 0
  - Et y met ses données
- Cible utilise un pointeur NULL
  - Lit / utilise les données
- Exploiter le noyau
  - Sinon, pas/peu d’intérêt

Protections

- Good Code & Design
  - Always check given pointers
  - *i.e.* RAII Pattern
- Analyse statique de code
- Défense en profondeur :
  - Noyaux refusent d’allouer @ NULL
  - *i.e.* `/proc/sys/vm/mmap_min_addr` sur 2.6.23

Uninitialized local variable

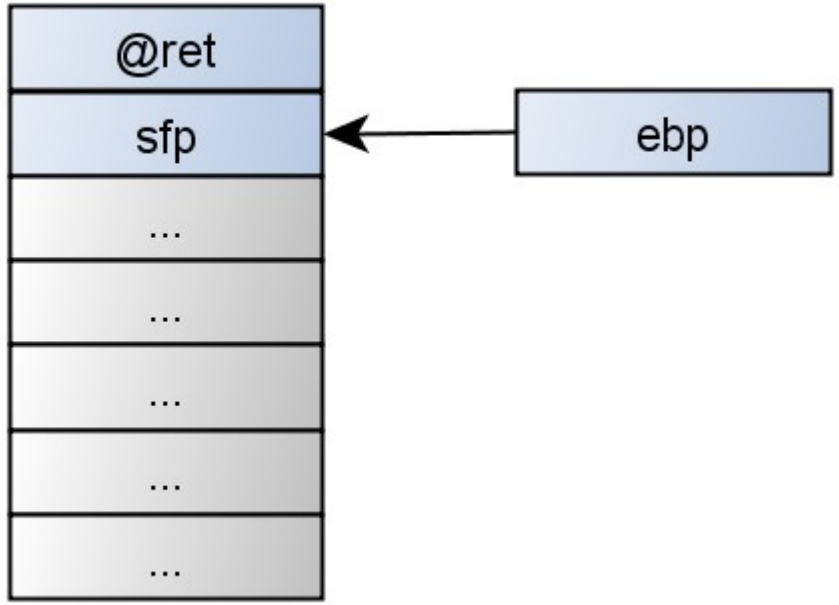
Exemple

```
// CVE 2016-8385 alike code

void function_a(char * buf) {
  size_t len = strlen(buf) ;
  // do stuffs
}

void function_b(char * buf) {
  size_t len ;
  char buffer[512] ;
  strncpy(buffer, buf, len) ;
}
```

```
int main(int argc, char ** argv) {
  if (argc < 3) {
    return 1 ;
  }

  function_a(argv[1]) ;
  function_b(argv[2]) ;
  return 0 ;
}
```

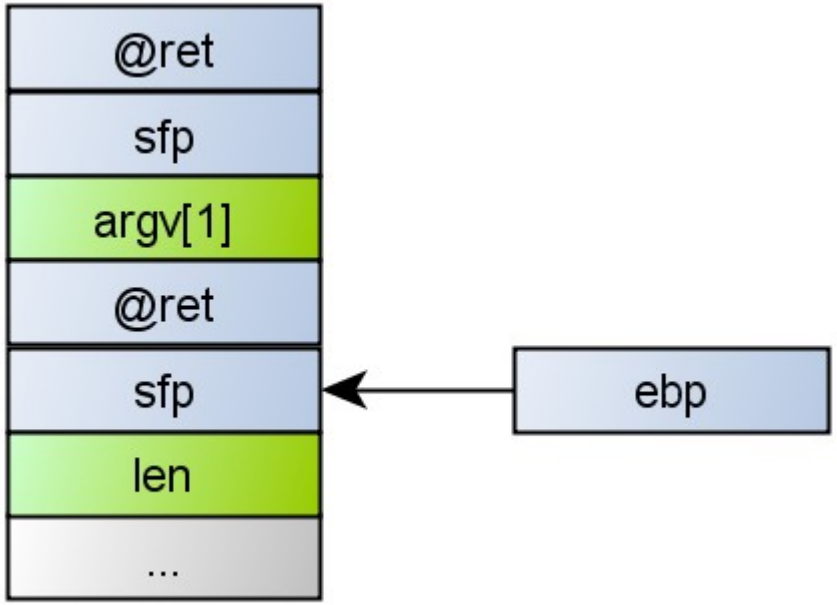
Pile

```
int main(int argc, char ** argv) {
  if (argc < 3) {
    return 1 ;
  }
  // ICI
  function_a(argv[1]) ;
  function_b(argv[2]) ;
  return 0 ;
}
```



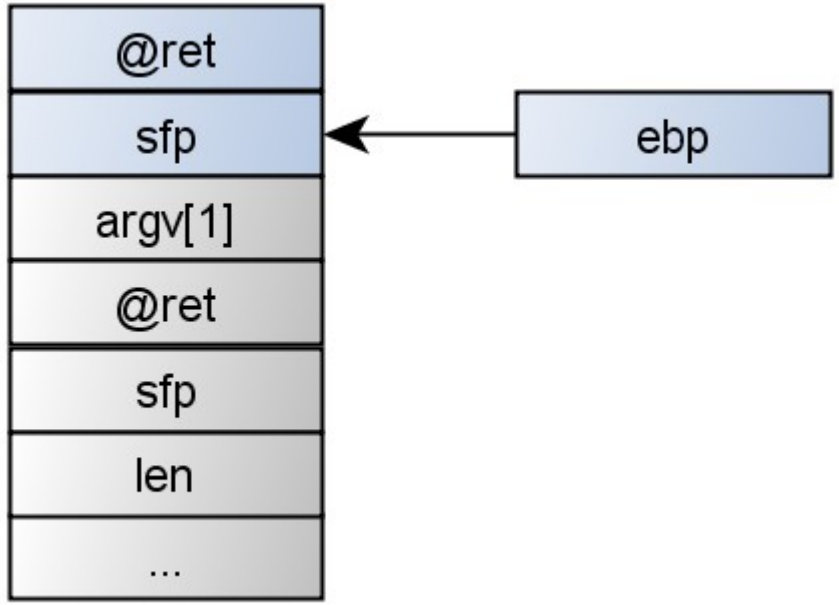
Pile

```
void function_a(char * buf) {
  size_t len = strlen(buf) ;
  // do stuffs
}
```



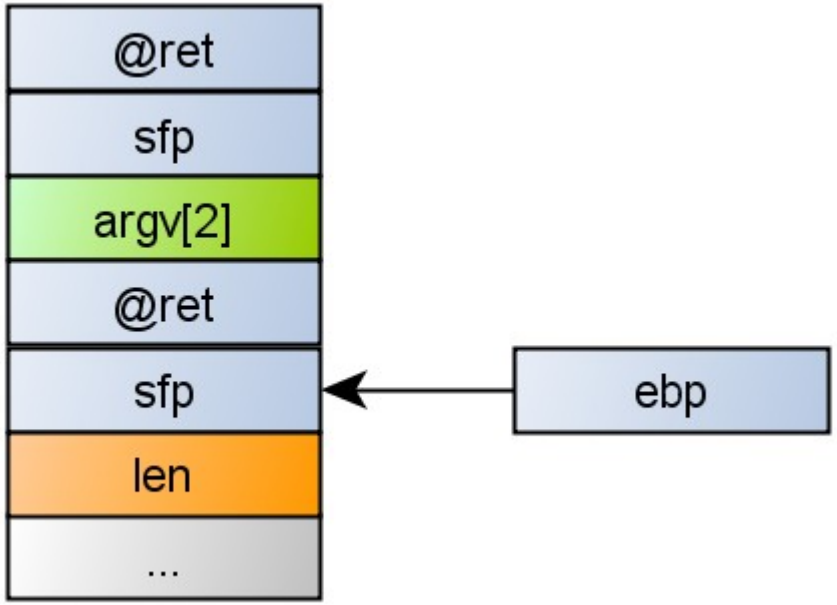
Pile

```
int main(int argc, char ** argv) {
  if (argc < 3) {
    return 1 ;
  }

  function_a(argv[1]) ; // ICI
  function_b(argv[2]) ;
  return 0 ;
}
```



Pile

```
void function_b(char * buf) {
  size_t len ;
  char buffer[512] ;
  strncpy(buffer, buf, len) ;
}
```



Protections

- Good Code & Design
    - Toujours initialiser lors de la déclaration
- Analyse statique de code
    - Gcc (voit pas tout)

Objets

Désérialization

Sérialisation

- **Serialize :**
  - Convertir la donnée en chaine
- **Unserialize :**
  - Retrouver la donnée à partir de la chaine
- **Languages objets :**
  - Java, PHP, C++, Python, Ruby, ...

Exemple 1/2

```
Class Example1
{
  public $cache_file;

  public function __construct() { … }

  public function __destruct() {
    @unlink($this->cache_file) ;
  }
}

$obj = unserialize(
  'O:8:"Exemple1":1:{s:10:"cache_file";s:15:"../../index.php";}'
  );
```

Exemple 2/2

```
class Example2
{
   private $hook;

   function __construct() { ... }

   function __wakeup()
   {
      if (isset($this->hook)) eval($this->hook);
   }
}



$obj = unserialize(
     'O:8:"Example2":1:{s:14:"Example2hook";s:10:"phpinfo();";}'
);
```

Conditions

- **Classe vulnérable**
  - Permet d’obtenir un effet
  - Peut nécessiter d’autres classes

- **Classe disponible**
  - Déjà chargée
  - Chargement dynamique (i.e. autoloader)

Protection

- Ne pas dé-sérialiser les données utilisateur
- Jamais, **S**ous **A**ucun **P**rétexte

Finalize attack

Ramasse miettes & finalize

- Objets non explicitement détruits
- Ramasse miette s’en charge
    - Appelle « finalize() » si définie

Finalize attack

Librarie

```
public class SecuredFile
{
  private string filename ;

  public SecuredFile(string filename, User me) {
    this.filename = filename ;
    if (! me.isAdmin()) {
      throw new Exception(« Not Allowed ») ;
    }
  }

  public string getContent() { … }

}
```

Finalize attack

Librarie

```
public class SecuredFile
{
  private string filename ;

  public SecuredFile(string filename, User me) {
    this.filename = filename ;
    if (! me.isAdmin()) {
      throw new Exception(« Not Allowed ») ;
    }
  }

  public string getContent() { … }

}
```

Attaquant

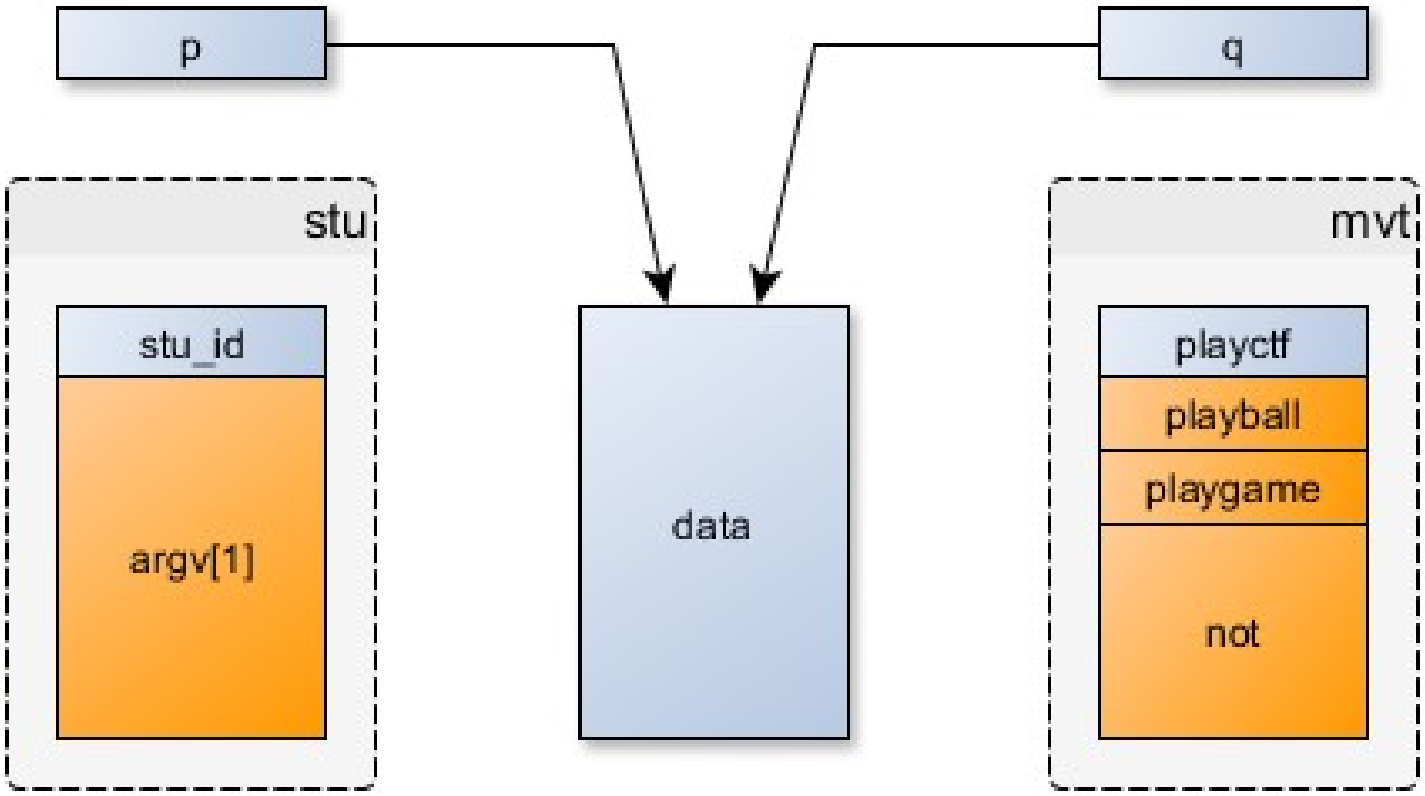
```
Public class MyEvilFile extends
SecuredFile
{


  public finalize() {
    string content = this.getContent() ;
    // do stuff
  }
}


SecuredFile mef =
    new MyEvilFile (« /etc/password ») ;
```

Protection

- Good Design
  - objets toujours valides
  - i.e. RAII

Bad Cast

Principe (photo non contractuelle)



Protection

- Ne pas « down caster »
    - Erreur de design
- Jamais, **S**ous **A**ucun **P**rétexte