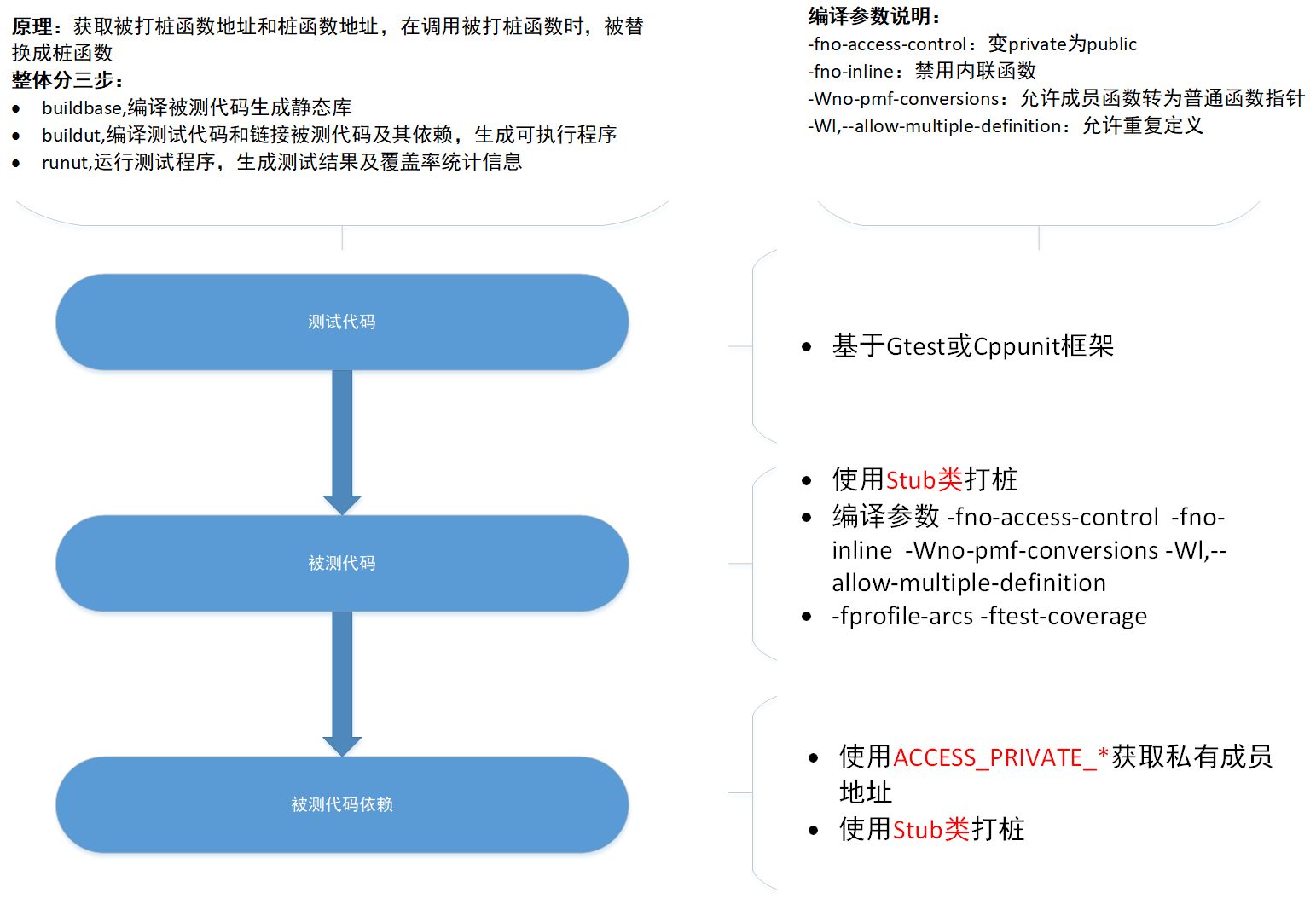


**注意：**

* 只适用linux的x86、x64架构
* ACCESS\_PRIVATE\_\*相关方法基于C++11
* Stub类可以基于C++03
* windows上也可以实现，替换掉linux的api，使用时注意函数的调用约定和linux不同



## 各种打桩的场景：

### 普通函数打桩（非static）

#include<iostream>

#include "stub.h"

using namespace std;

int foo(int a)

{

cout<<"I am foo"<<endl;

return 0;

}

int foo\_stub(int a)

{

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

Stub stub;

stub.set(foo, foo\_stub);

foo(1);

return 0;

}

### 实例成员函数打桩

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int i;

public:

int foo(int a){

cout<<"I am A\_foo"<<endl;

return 0;

}

};

int foo\_stub(void\* obj, int a)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

Stub stub;

stub.set(ADDR(A,foo), foo\_stub);

A a;

a.foo(1);

return 0;

}

### 静态成员函数打桩

### 模板函数打桩

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int i;

public:

static int foo(int a){

cout<<"I am A\_foo"<<endl;

return 0;

}

};

int foo\_stub(int a)

{

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

Stub stub;

stub.set(ADDR(A,foo), foo\_stub);

A::foo(1);

return 0;

}

#include<iostream>

#include "stub.h"

using namespace std;

class A{

public:

template<typename T>

int foo(T a)

{

cout<<"I am A\_foo"<<endl;

return 0;

}

};

int foo\_stub(void\* obj, int x)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

Stub stub;

stub.set((int(A::\*)(int))ADDR(A,foo), foo\_stub);

A a;

a.foo(5);

return 0;

}

### 重载函数打桩

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int i;

public:

int foo(int a){

cout<<"I am A\_foo\_int"<<endl;

return 0;

}

int foo(double a){

cout<<"I am A\_foo-double"<<endl;

return 0;

}

};

int foo\_stub\_int(void\* obj,int a)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub\_int"<< a << endl;

return 0;

}

int foo\_stub\_double(void\* obj,double a)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub\_double"<< a << endl;

return 0;

}

int main()

{

Stub stub;

stub.set((int(A::\*)(int))ADDR(A,foo), foo\_stub\_int);

stub.set((int(A::\*)(double))ADDR(A,foo), foo\_stub\_double);

A a;

a.foo(5);

a.foo(1.1);

return 0;

}

### 虚函数打桩

#include<iostream>

#include "stub.h"

using namespace std;

class A{

public:

virtual int foo(int a){

cout<<"I am A\_foo"<<endl;

return 0;

}

};

int foo\_stub(void\* obj,int a)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

typedef int (\*fptr)(A\*,int);

fptr A\_foo = (fptr)(&A::foo); //获取虚函数地址

Stub stub;

stub.set(A\_foo, foo\_stub);

A a;

a.foo();

return 0;

}

### 内联函数打桩

添加-fno-inline编译选项，禁止内联，能获取到函数地址，打桩参考上面。

### 第三方库私有成员函数打桩

已经存在的库，无法通过-fno-access-control编译选项去掉权限

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int a;

int foo(int x){

cout<<"I am A\_foo "<< a << endl;

return 0;

}

static int b;

static int bar(int x){

cout<<"I am A\_bar "<< b << endl;

return 0;

}

};

ACCESS\_PRIVATE\_FIELD(A, int, a);

ACCESS\_PRIVATE\_FUN(A, int(int), foo);

ACCESS\_PRIVATE\_STATIC\_FIELD(A, int, b);

ACCESS\_PRIVATE\_STATIC\_FUN(A, int(int), bar);

int foo\_stub(void\* obj, int x)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub"<<endl;

return 0;

}

int bar\_stub(int x)

{

cout<<"I am bar\_stub"<<endl;

return 0;

}

int main()

{

A a;

auto &A\_a = access\_private\_field::Aa(a);

auto &A\_b = access\_private\_static\_field::A::Ab();

A\_a = 1;

A\_b = 10;

call\_private\_fun::Afoo(a,1);

call\_private\_static\_fun::A::Abar(1);

auto A\_foo= get\_private\_fun::Afoo();

auto A\_bar = get\_private\_static\_fun::A::Abar();

Stub stub;

stub.set(A\_foo, foo\_stub);

stub.set(A\_bar, bar\_stub);

call\_private\_fun::Afoo(a,1);

call\_private\_static\_fun::A::Abar(1);

return 0;

}

## 不可以打桩的场景：

* 不可以对exit函数打桩，编译器做了特殊优化
* 不可以对纯虚函数打桩，纯虚函数没有地址
* static声明的普通内部函数不能打桩，内部函数地址不可见

## 总结

最终目的就是为了获取被打桩函数地址，只要能获取到地址就能被打桩；桩函数实现主要依赖函数调用约定（**\_\_cdecl、\_\_stdcall、\_\_thiscall等）**