Lecture 18

Subverting a Type System

turning a bit flip into an exploit

Ras Bodik
Ali and Mangpo

Hack Your Language!
CS164: Introduction to Programming Languages and Compilers, Spring 2013
UC Berkeley

Today

Type safety provides strong security guarantees.

But certain assumptions must hold first:

- banning some constructs of the language
- integrity of the hardware platform

These are critical. Failure to provide these permits type system subversion.

Today: type system subversion

- means and consequences

why static types?

review

What do compilers know thanks to static types?

Dynamically-typed languages (Python/Lua/JS):

```
function foo(arr) {
    return arr[1]+2
}
```

Interesting questions for the compiler:
- where an exception can(not) be thrown?
- what function does + refer to?
- how to represent data in memory?

Statically-typed languages (Java/C#/Scala):

type of argument ← programmer provided → return type of function

```
function foo(arr:int[]) : int {
    return arr[1]+2
}
```

Declared types lead to compile-time facts

Let's discuss our example.

The + operator/function:

In **Java**: we know at compile time that + is an integer addition, because type declarations tell the compiler the types of operands.

In **JS**: we know at compile time that + could be either int addition or string concatenation. Only at runtime, when we know the types of operand values, we know which of the two functions should be called.

Declared types lead to compile-time facts

Does a Python compiler know that variable arr will refer to a value of indexable type?

It looks like it should, because arr is used in the indexing expression arr[1].

But Python does not even know this fact for sure. After all, foo could be legally called with a float argument, say foo(3.14). Yes, foo will throw an exception in this case, at arr[1], but the point is that the compiler must generate code that checks (at runtime) whether the value in arr is an indexable type. If not, it will throw an exception.

How do compilers exploit compile-time knowledge?

dynamically typed languages (Python/Lua/JS):

```
function foo(arr) {
   return arr[1]+2
}
```

ask yourself:
- how can array of int be represented in Python, Java
- which one is more efficient?

statically typed languages (Java/C#/Scala):

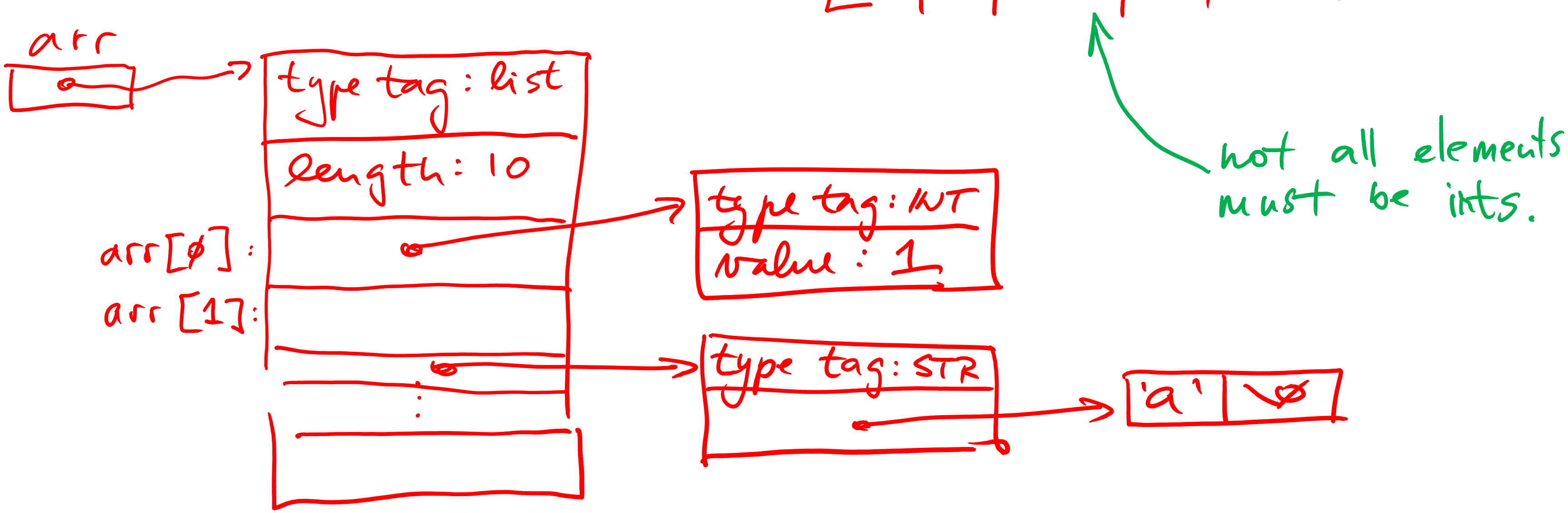
```
function foo(arr:int[]) : int {
   return arr[1]+2
}
```

Data structure rerepsentation

In Python, `arr[i]` must check at runtime:

- is (the value of) arr an indexable object (list or a dictionary)?
- what is type (of value in) of arr[1]?

The representation of array of ints must facilitate these runtime questions:



Compare this with array of ints in Java

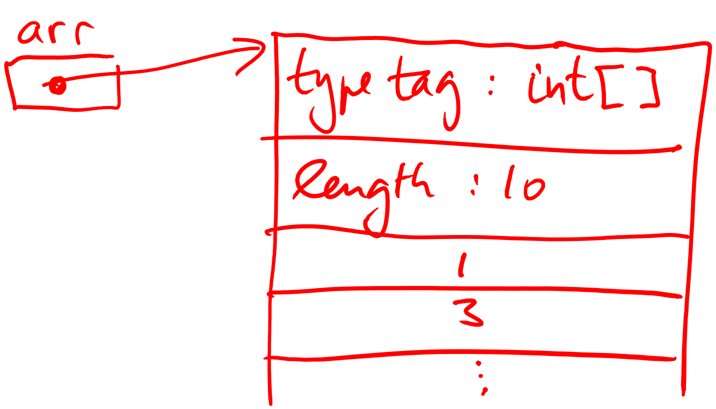
Java arrays must be homogeneous

- All elements are of the same type (or subtype)

We know these types at compile time

- So the two questions that Python asks at runtime can be skipped at Java runtime, because they are answered from static type declarations at compile time

Hence Java representation of arrays of ints can be:



[1, 3, 7, 5, ..., 19]

Private fields

Private object fields

Recall the lecture on embedding OO into Lua

We can create an object with a private field

- the private field stores a password that can be checked against a guessed password for equality but the stored password cannot be leaked

Next slide shows the code

Object with a private field

```
// Usage of an object with private field
def safeKeeper = SafeKeeper(“164rocks”)
print safeKeeper.checkPassword(“164stinks”)  -->  False

// Implementation of an object with private field
function SafeKeeper (password)
    def pass_private = password

    def checkPassword (pass_guess) {
        pass_private == pass_guess
    }
    // return the object, which is a table
    { checkPassword = checkPassword }
}
```

Let’s try to read out the private field!

Assume I agree to execute any code you give me.
Can you print the password (without trying all passwords)?

```
def safeKeeper = SafeKeeper(“164rocks”)
def yourFun = <paste any code here>
// I am even giving you a ref to keeper
yourFun(safeKeeper)
```

This privacy works great, under certain assumptions. Which features of the 164 language do we need to disallow to prevent reading of pass_private?

1. overriding == with our own method that prints its arguments
2. access to the environment of a function and printing the content of the environment

(such access could be allowed to facilitate debugging, but it destroys privacy)

Same in Java, using private fields

```
class SafeKeeper {
    private long pass_private;
    SafeKeeper(password) { pass_private = password }

    Boolean checkPassword (long pass_guess) {
        return pass_private == pass_guess
}   }

SafeKeeper safeKeeper = new SafeKeeper(920342094223942)
print safeKeeper.checkPassword(1000000000001)  -->  False
```

Redoing the exercise in Java illustrates that the issues exist in a statically typed language, too.

Challenge: how to read out the private field?

Different language. Same challenge.

```
SafeKeeper safeKeeper = new SafeKeeper(19238423094820)
<paste your code here; it can refer to ‘safeKeeper’>
```

Compiler rejects program that attemps to read the private field

That is, p.private_field will not compile to machine code

But some features of Java need to be disallowed to prevent reading of pass_private.

- Reflection, also known as introspection

  read about the ability to read private fields with java reflection API)

Summary of privacy with static types?

It's frustrating to the attacker that

(1) he holds a pointer *a* to the Java object, and

(2) knows that password is at address *a*+16 bytes

yet he can't read out password_private from that memory location.

Why can't any program read that field?

0. Compiler will reject program with p.private_field
1. Type safety prevents variables from storing incorrectly-typed values.

   B b = new A() disallowed by compiler unless A extends B
2. Array-bounds checks prevent buffer overflows
3. Can't manipulate pointers (addresses) and hence cannot change where the reference points.

Together, these checks prevent execution of arbitrary user code…

Unless the computer breaks!

Manufacturing a Pointer in C

Attack in C language

Before we describe the attack in Java, how would one forge (manufacture) a pointer in C

```
union {  int i; char * s; } u;
```

Here, i and s are names for the same location.

u.i = 1000

u.s[0] --> reads the character at address 1000

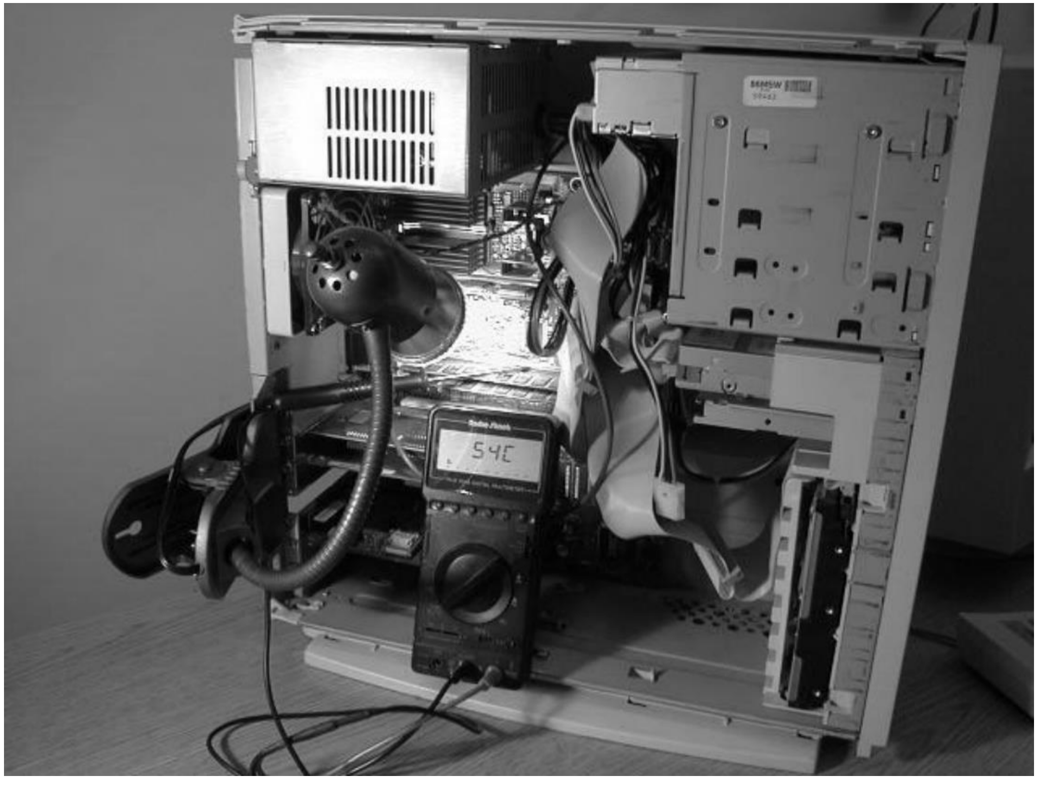
http://stackoverflow.com/questions/4748366/can-we-use-pointer-in-union

How to create a hardware error?

Memory Errors

A flip of some bit in memory

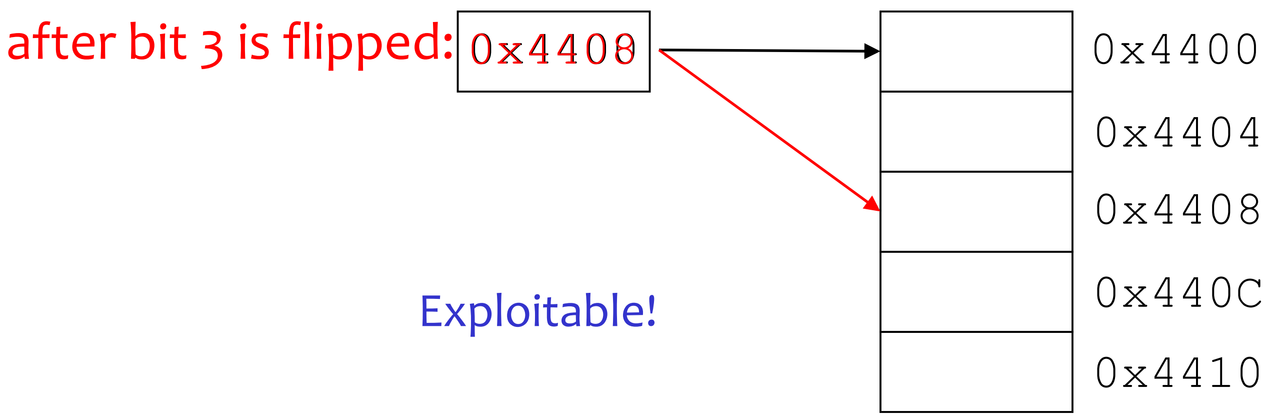
- Can be caused by cosmic ray, or deliberately through radiation (heat)



Effects of memory errors

Bitflip manufactures a pointer

except that we cannot control what pointer and in which memory location.



Manufacturing a Pointer in Java and Exploiting it

Overview of the Java Attack

Step 1: use a memory error to obtain two variables p and q, such that

1. p == q (i.e., p and q point to same memory loc) and
2. p and q have incompatible, custom static types

Cond (2) normally prevented by the Java type system.

Step 2: use p and q from Step 1 to write values into arbitrary memory addresses

- Fill a block of memory with desired machine code
- Overwrite dispatch table entry to point to block
- Do the virtual call corresponding to modified entry

We'll cover Step 2 first.

The two custom classes will form C-like union

```
class A {
  A a1;
  A a2;
  B b;    // for Step 1
  A a4;
  int i; // for address
         // in Step 2
}
```

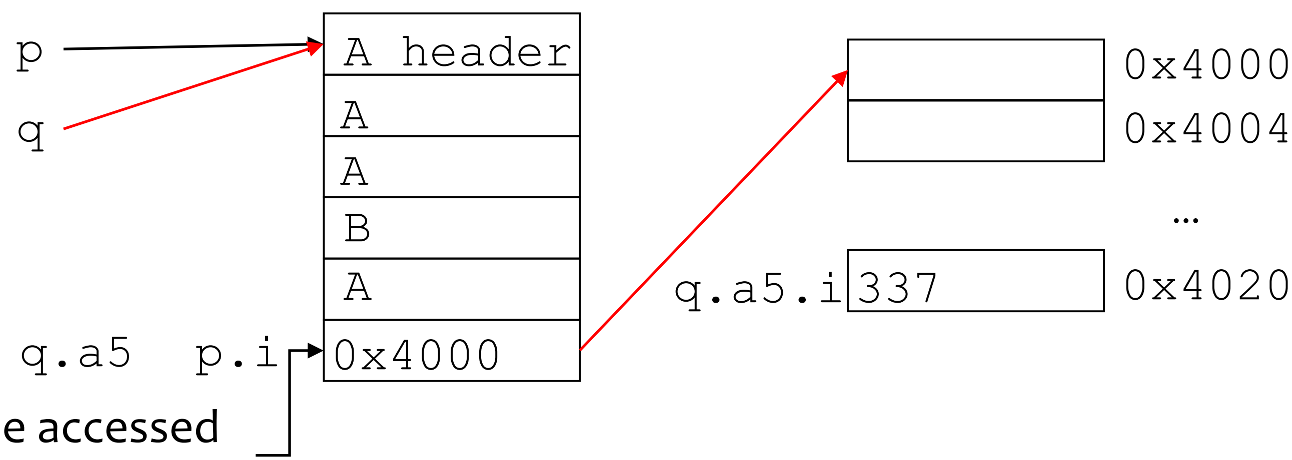
```
class B {
  A a1;
  A a2;
  A a3;
  A a4;
  A a5;
}
```

Assume 3-word object header

Step 2 (Writing arbitrary memory)

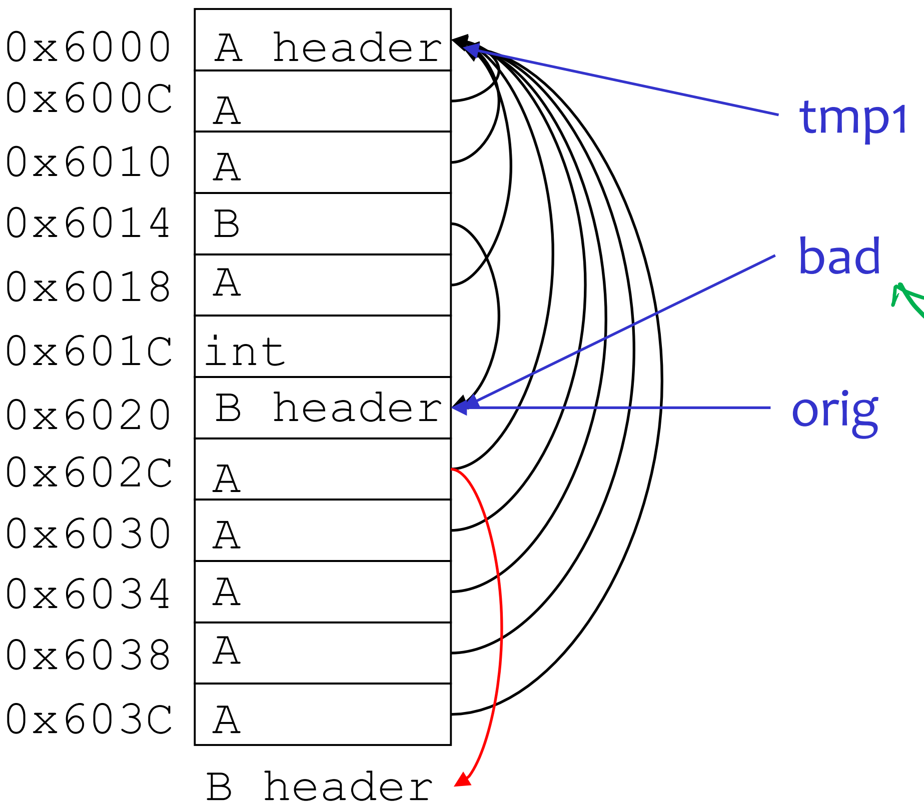
```
int offset = 8 * 4;       // Offset of i field in A
A p; B q;                 // Initialized in Step 1, p == q;
                          // assume both p and q point to an A

void write(int address, int value) {
  p.i = address – offset;
  q.a5.i = value; // q.a5 is an integer treated as a pointer
}
```

Example: write 337 to address 0x4020



this location can be accessed as both q.a5 and p.i

Step 1 (Exploiting The Memory Error)

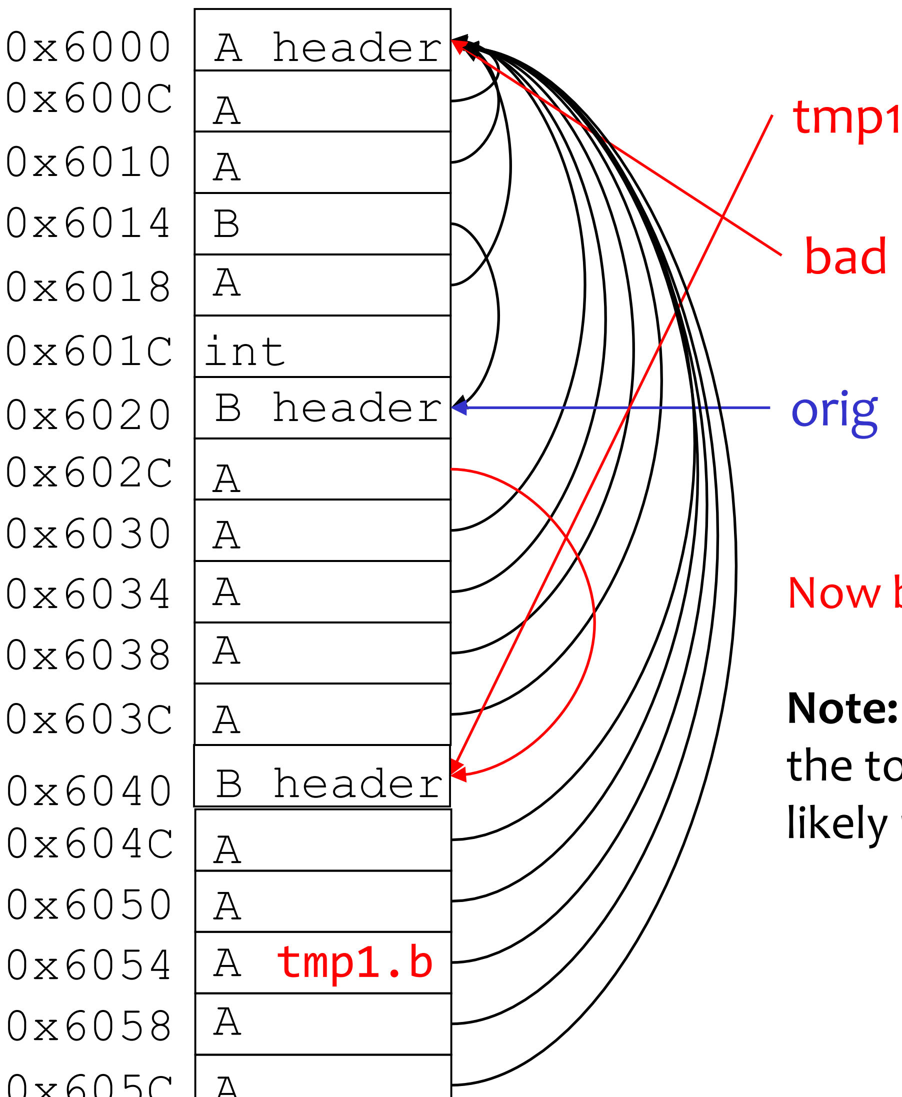


B orig;
A tmp1 = orig.a1;
B bad = tmp1.b;

without a bitflip, bad does not contain a 'bad' value.

The heap has one A object, many B objects. All fields of type A point to the only A object that we need here. Place this object close to the many B objects.

Step 1 (Exploiting The Memory Error)



B orig;
A tmp1 = orig.a1;
B bad = tmp1.b;

flip bit 0x40 in orig.a1

Now bad points to an A object!

Note: it is a coincidence that orig.a points to the top of the object header. It could equally likely point into an object of type B.

Step 1 (cont)

Iterate until you find that a flip happened and was exploited.

```
A p; // pointer to the single A object
while (true) {
  for (int i = 0; i < b_objs.length; i++) {
    // iterate over all B objects
    B orig = b_objs[i];

    A tmp1 = orig.a1; // Step 1, really check a1, a2, a3, …
    B q = tmp1.b;

    Object o1 = p; Object o2 = q; // check if we found a flip
    // must cast p,q to Object to allow comparison
    if (o1 == o2) {
      writeCode(p,q); // now we're ready to invoke Step 2
} } }
```

Results (paper by Govindavajhala and Appel)

With software-injected memory errors, took over both IBM and Sun JVMs with 70% success rate

think why not all bit flips lead to a successful exploit

Equally successful through heating DRAM with a lamp

Defense: memory with error-correcting codes

- ECC often not included to cut costs

Most serious domain of attack is smart cards

Paper: http://www.cs.princeton.edu/~sudhakar/papers/memerr.pdf