Lecture 8 : Type Confusion

Alexandre Bartel

2019

Previously...

Previously... in Lecture 1 (Introduction)

- Software Development Life-cycle
- Vulnerability Life-cycle
- Vulnerability Disclosure

Previously... in Lecture 2 (Buffer Overflow)

- A buffer on the stack
- Return address on the stack
- Overwrite return address
- Jump to shellcode on the stack

Previously... in Lecture 3 (ROP)

- NX bit (stack non-executable)
- Gadgets in already loaded code
- Chain gadgets (addresses of gadgets and data on the stack)
- Only data on the stack

Previously... in Lecture 4 (ASLR)

- Randomize code segment at program start
- Breaks gadget chains
- Bypass with information leak (e.g, vulnerability)

Previously... in Lecture 6 (CFI)

- Mecanism to allow only "intended" paths
- Binary instrumentation to add IDs
- Indirect jumps, call, returns check if ID of "destination" is correct
- Pure software implementation have 20% overhead

Previously... in Lecture 7 (Heap-Overflow)

- How a heap-overflow can be attacked depends on the heap management implementation
- The "unlink" attack present in early versions of glibc provides a "write anywhere" primitive to the attacker
- Recent implementations performs more check to prevent "unlink" based attacks

Type Confusion

Type Confusion

- Type: Concept in Object-oriented langages
- Confusion: Type A is though of as being type B

Type

- In C++, an object contains a number of data fields, and a number of functions to modify them
- In Java, an object contains a number of data fields, and a number of methods to modify them
- Each object represents an abstract or a concrete concept and has a **type** (e.g., Vehicle, Car, Truck)
- Types can be classified in a structure called **class hierarchy** or **inheritance tree**

Exercise: Hierarchy Example for Vehicle, Car, Truck.

Type Confusion in C++

In C++

- C++ is statically typed
- The compiler makes sure type usage is safe in the program
- However, the programmer can bypass the compiler by using one of the following operation performing a type conversion:
    1. **static_cast**: statically checks that there is a relationship between the current type and the target type → potential type confusion
    2. **reinterpret_cast**: no verification at all done by the compiler → potential type confusion
    3. **dynamic_cast**: dynamically checks that the current type is compatible with the target type. If not an exception is raised → no type confusion possible

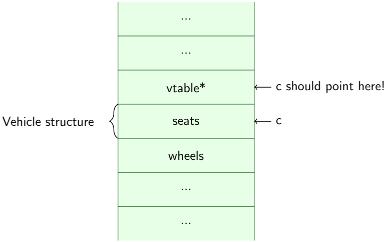
Objects In C++

- An object is a sequence of bytes in memory
- Every field is localize at some fixed offset (computed at compilation time) from the start of this sequence
- An object having at least one virutal method has a memory structure called the `vtable` (or `vftable`)
- Every virtual method is represented by an index in the `vtable`
- The vtable is placed before the sequence of bytes representing fields

C++ Type Confusion Example

```
class Vehicle {
  int seats;
}
class Car : Vehicle {
  int wheels;
  virtual void drive();
}

Vehicle *v = new Vehicle;
Car *c = static_cast <Car*>v; // type confusion
c->wheels = 0x33 ; // undefined behavior
c->drive(); // undefined behavior
```



When can an Attacker Trigger a Type Confusion in C++?

- Environments which enable attacker to execute code (Browsers with javascript, JVM, etc.)
- Huge class hierarchy
- Enables attacker to execute his own code

Type Confusion in Java

Type Confusion in Java

- Same principle for Java
- Breaks encapsulation

Context For the Attacker

- Context: attacker can execute arbitrary code in a sandboxed JVM.

Security Manager and Permissions

- The sandbox in Java is activated when there is a Security Manager.
- Before every protected operation, the untrusted code is checked against one ore multiple permissions (e.g., READ_FILE)
- If the code does not have the permission, an exeception is thrown

Security Manager and Permissions

Example of a permission check for the method changing the value of a property (ex of property: "java.class.path").

```java
public class System {
  [...]
    public static String setProperty(String key, String value) {
        checkKey(key);
        SecurityManager sm = getSecurityManager();
        if (sm != null) { // no check if no security manager
            // permission check
            sm.checkPermission(new PropertyPermission(key,
                SecurityConstants.PROPERTY_WRITE_ACTION));
        }
        // if permission check fails, an exception is thrown
        // so this code is not executed
        return (String) props.setProperty(key, value);
    }
}
```

Goal for an attacker in Java

- If there is a sandbox, the security manager is NOT null.
- Goal for an attacker: disable the sandbox.
- If the field `System.securityManager` is set to `null`, the security manager is disabled and no security check is performed

```
public final class System {
  [...]
    /* The security manager for the system.
     */
    private static volatile SecurityManager security = null;
  [...]
    public static SecurityManager getSecurityManager() {
        return security;
    }
  [...]
}
```

Conclusion

- What is it? Manipulation of an object through another object.
- Consequences? Undefined behavior, hijack control flow.
- Why it works? No verification at runtime.

Question?

Projets

- Groups of 2
- Suggested topics:
  1. Heap exploitation on Debian 3.1
  2. Patch for CVE-2018-20343
  3. Complete exploit for CVE-2018-20343
  4. Study and PoC for CVE-2013-0912 (Chrome type confusion)
  5. Stable code injection through `/proc/self/mem`
  6. Explanation of a recent exploit targeting webbrowsers (Chrome, Firefox, etc.)
  7. Exploitation of a PoC type confusion in C++
- Deliverables: Presentation + Code (PoC)

Projet: Heap exploitation on Debian 3.1

- Explain the differences in the heap management code from debian 2.2 (lab 7) and debian 3.1
- Explain and develop a proof-of-concept to exploit a heap overflow on debian 3.1

Projet: Patch for CVE-2018-20343

- Understand CVE-2018-20343, a buffer overflow vulnerability
- You have to identify all instances of buffer overflow in the code (the code is not very big)
- You have to patch the vulnerable code

Projet: Complete exploit for CVE-2018-20343

- The current proof-of-concept only changes the value of EIP.
- You have to improve the PoC to enable an attacker to execute arbitrary code

Projet: Study and PoC for CVE-2013-0912 (Chrome type confusion)

- Reproduce the SVG code for the exploit based on information you find on the internet
- You should create a VM with a distribution from 2013 and have the vulnerable version of Chrome

Projet: Stable code injection through `/proc/self/mem`

- DosBox enables untrusted code to mount the host filesystem in the guest
- Thus untrusted code can write to `/proc/self/mem`
- You develop code to inject a shellcode into the virtual process of dosbox to execute arbitrary code
- You do this by writing to `/proc/self/mem`

Projet: Explanation of a recent exploit targeting webbrowsers (Chrome, Firefox, etc.)

- Contact me when you have found a CVE you want to explain.

Projet: Exploitation of a PoC type confusion in C++

- Write a proof-of-concept showing how to exploit a type confusion in C++ in a x86_64 architecture (latest debian)