克服 Java 代码篡改、逆向工程和盗窃漏洞

白皮书

如今，越来越多的开发人员选择 Java 平台作为首选的开源开发环境，通过这种平台可以直接使用可公开获得的.class 文件格式和简单的指令集。使用开放源码方法有许多优点，但这样编写的代码对于窥探是完全透明的。对于已开发软件中经过多年时间和投资形成的公司知识产权（IP），其屏障因此变得公开了。

目录

Sentinel 中文为“圣天诺”。

Java 开放源码风险和弱点

- JVM 是免费的 - 黑客可以使用商业反汇编程序查看代码，弄清 JVM 的工作方式 - 代码变得完全透明
- Java .class 是可公开获取的 - 黑客可使用简单的可用工具处理、修改或转换.class 文件
- 更少的本机代码指令 - 黑客现在使用更少的指令（200 而不是 400），可以更快地分析代码以进行逆向工程
- 更好控制 - 黑客不需要深入到特定的硬件级别，可以更容易地查看和分析数据

执行摘要

由于具有平台独立性，Java 环境越来越受到开发人员的欢迎。然而，Java 的开源代码、可公开获取的.class 文件格式和简单的指令集使其暴露于窥探和敌对分析之下。随着各机构将它们的商业应用程序转向 Java 技术，其关键知识产权变得易于遭受逆向工程、篡改和盗窃。目前，源代码和字节码级别的.class 文件加密和混淆是开发人员用于阻止逆向工程的两个主要措施，但漏洞仍然存在。

本白皮书探讨了 Java 代码漏洞的性质，现有解决方案的局限性，以及结合混淆与加密的 Java 代码全面封装如何成为保护关键算法和其它知识产权的理想解决方案。

Java 应用程序正变得越来越普遍

如今，全世界的 IT 公司都在一定程度上接受了 Java。现在有无数的产品使用 Java 编写并用于 Java 平台，而不是 Windows 或 UNIX 平台。与其它编程语言相比，Java 最重要的优点是平台独立性 - 这意味着它几乎可以在任何体系结构和操作系统上运行。Java 并不将应用程序绑定到特定的硬件体系结构（如 Intel 或 PowerPC），而是对所有体系结构都使用单一的代码库。软件开发人员先以使用预定义 Java API 软件包的 Java 语言编写程序， 然后使用 Java 编译器来编译这些程序。其成果是可在 Java 虚拟机（JVM）上执行的已编译字节码，Java 虚拟机是用于执行 Java 程序的软件环境（如 Java Runtime Environment）。用户仅在拥有 JVM 时才能运行 Java 程序；但许多平台都有 JVM，这使得 Java 成为一种高度可移植语言。

Java 极易遭受逆向工程

虽然能够“编写一次，随处运行”（Write Once, Run Anywhere）是一个巨大的优势，但这种环境的架构方式使其远比本机应用程序更容易被黑客进行逆向工程。这意味着开发人员面临着失去知识产权的非常真实的危险。基于应用程序的虚拟机比本机应用程序更容易逆向工程的原因有很多：

JVM 是开源的

Sun 已经免费提供 JVM 的源代码。这使得黑客只需查看代码即可弄清虚拟机的工作方式。

Java .class 文件格式是可公开获取的

如前所述，Java 源代码被编译成字节码，而字节码存储在 Java .class 文件中。Java .class 文件格式的规范是可公开获取的，因此有技术背景的任何人都能容易地编写可以处理、修改或转换.class 文件的工具。

JVM 是软件，而不是硬件

与需要理解特定处理器的专家使用的标准编程语言不同，JVM 是一个应用程序，它如同微处理器一样运作，并使用操作系统和计算机硬件提供的内置功能。由于黑客不必深入到硬件级别，因此更容易取得对 JVM 的完全控制。

因此，例如在使用标准本机系统开发语言进行调试时，暂停处理器极为困难，需要具备处理器、调试功能及可用环调试器的专家知识。但是，由于 JVM 运行环境的源代码是可公开获取的，因此开发人员可以轻松地建立自己的虚拟机来完全控制虚拟处理器的各个方面。这样可以容易地分析运行环境中运行的每个应用程序。

Sentinel Envelope

特性和优点

- 文件自动包装器 - 通过文件加密和本机代码混淆提供强有力的保护来防止软件逆向工程
- 将应用程序重新关联到硬件 - 应用程序通过保护密钥与与硬件紧密结合
- 安全通信通道 - SENTINEL HASP 为受保护应用程序与保护密钥之间的通信提供安全通道，从而消除了中间人攻击。Java Envelope 使用这种能力来防止黑客通过截取通信访问保护密钥发回的数据。
- 运行时解密 - 由于 SENTINEL HASP 是按请求在运行时解密文件，而不是一次将所有.class 文件加载到虚拟机，因此可以防止黑客重建整个应用程序。

Java 的指令比本机代码少

然而，JVM 代码易于进行反向工程的另一个原因是它具有比本地应用程序更少的指令。这是出于性能考虑。JVM 的使用在应用程序和本机处理器之间增加了一个软件层，这会对性能产生负面影响。虽然现代处理器不断提高的执行速度最终将缓解这一问题，但这一问题仍然很明显。虚拟机开发人员提高执行速度的一种方法是使用比本机处理器汇编程序更小的字节码指令集。本机应用程序可能包含多达 400 条指令，而 Java 应用程序通常使用不超过 200 条的指令。更少的指令意味着黑客可以更快地分析代码以进行逆向工程。

这些特性使得虚拟机远比其它类型的应用程序更容易遭受逆向工程攻击。

第三方反汇编程序增加了漏洞

不仅是 JVM 本身容易遭受逆向工程攻击，商业和免费的 Java 字节码反汇编程序也越来越多，从而进一步简化了代码逆向工程的过程。

IDA 和 Eclipse 字节码插件是众多 Java 字节码反汇编程序中的两种。作为商业产品，IDA 是一种普遍的反汇编程序，可用于许多不同的处理器，包括 80x86 和 MIPS。Eclipse 字节码插件是免费软件。它能够反编译 Java .class 文件的字节码并以适当的顺序显示所有操作码指令。

尽管这些产品不大可能从字节码完美地恢复原始代码，但它们恢复的源代码将等同于原始代码，并且比字节码更具可读性。一旦恢复了源代码，攻击者可以容易地删除部分代码并将其非法地用于竞争对手的应用程序中，或在.class 文件中定位打补丁。

图 1 提供了黑客可能如何在.class 文件中打补丁的一个示例。屏幕的上半部分显示了一小段 Java 源代码。屏幕的下半部分显示了字节码反汇编的输出，也就是一个字节码指令列表。标记为红色的区域是源代码中 IF 结构的对应指令。字节码指令“LCMP”的十六进制表示为 0x94。该工具还指出了操作码在.class 文件中的位置。有了这些信息，黑客可以使用简单的十六进制编辑器来改变该 IF 分支，而这只需不到一分钟的时间。假设该 IF 条件用于许可证检查，黑客可倒置该条件，指示即使在许可证被验证为无效（如已过期）的情况下仍返回“True”，从而突破许可证检查。在这种情况下，黑客使用一个字节的补丁即可完成所有工作。虽然大多数应用程序都比这个示例更加复杂，但即使在复杂的应用程序中，字节码也非常简单并且容易理解。

JVM 为开发人员提供了编写一次应用程序即可在几乎任何平台上运行的能力，但具有使黑客易于对源代码进行逆向工程、篡改或盗窃的重大缺陷。

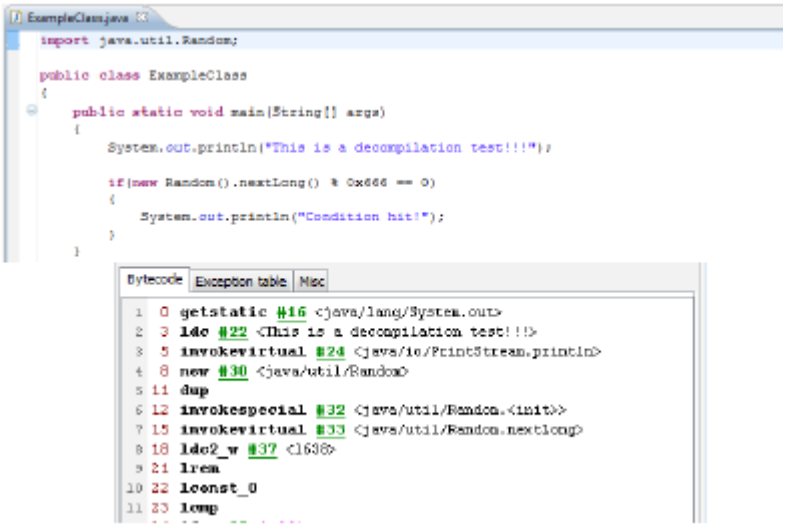


图 1 - 用于字节码反汇编的 Eclipse 字节码插件

为什么板载措施不足以防止逆向工程

大多数虚拟机都包含一些使逆向工程复杂化的功能。Java 允许用户在 JAR 存档中提供的每个类上设置一个数字证书，以确保原始文件没有被更改。虽然这样做并无害处，但该功能相当容易清除，并且仅针对静态补丁方法提供保护，而静态补丁只是攻击场景中的一小部分。而且，这种方法并不能针对运行时应用于内存的补丁提供保护。

Java 还通过虚拟机执行字节码验证器，该验证器在执行通过的字节码之前对其进行自动分析。这可以防止执行“奇怪”的代码，也使字节码注入变得更加困难。

然而，尽管这些措施给攻击者造成了困难，但对于充分保护知识产权还远远不够。

防止逆向工程攻击

开发人员通常用以防止.class 文件静态分析和字节码反汇编的一种方法是封装，这种方法通过应用加密/解密完整文件来防止对类文件的分析。通过封装，开发人员将受保护文件的原始加载器更换为处理加密/解密的自定义加载器。加密使用将.class 文件从标准 Java .class 格式更改为仅“密钥”所有者可读格式的算法来防止对这些文件的分析。然而，.class 文件的字节码在一个内存位置中仍保持可读，在系统加载器尝试加载该类之前的时刻，通常可从该位置访问字节码。如果黑客能够找到那个内存位置，就可以访问原始状态的该类。

防止黑客攻击该内存位置需要第二种技术，称为混淆。混淆可产生一个更加复杂、难于理解并且与原始代码具有相同行为方式的代码版本。下面是一个简单的 80x86 汇编程序代码段，取自 Windows 二进制代码。

虽然 Java 提供了一些内在的安全措施，但这些功能不足以完全防止攻击。加密和代码混淆等技术通常用于减缓攻击，但仍然留有漏洞。Java 封装（Java Enveloping）将加密与本机代码混淆相结合来提供最强的保护。通过使用 SENTINEL HASP 解决方案，您可以获得封装的诸多好处，而不必花费宝贵的时间和精力来自行开发解决方案。

```
text:0040C609 ; =============== S U B R O U T I N E =======================================
text:0040C609
text:0040C609
text:0040C609 sub_40C609          proc near          ; DATA XREF: .rdata:004561A4o
text:0040C609
text:0040C609 arg_0         = dword ptr 4
text:0040C609 Src                   = dword ptr 0Ch
text:0040C609 Size                  = dword ptr 10h
   ext:0040C609

   push  ebx
push     esi
push     [esp+8+Size] ; Size
mov esi, [esp+0Ch+arg_0]
push     [esp+0Ch+Src] ; Src
push     0 ; int

push     dword ptr [esi+0Ch] ; int
call      sub_43AF3E

   mov ecx,        [esi+4]
mov edx,           [ecx]
xor        ebx, ebx
cmp eax, 1000h
setnle    bl
push     ebx
push     ecx
call      dword ptr [edx+1Ch]
add  esp, 18h
xor        eax, eax
pop         esi
inc        eax
pop         ebx
retn
```

现在我们将尝试在代码段中添加新的指令并更换其它指令以使代码难以理解，从而混淆代码段。

下面是已混淆的版本：

```
text:0040C609 ; =============== S U B R O U T I N E =======================================
text:0040C609
text:0040C609
text:0040C609 sub_40C609 proc near ; DATA XREF: .rdata:004561A4o
text:0040C609
text:0040C609 arg_0 = dword ptr 4
text:0040C609 Src = dword ptr 0Ch
text:0040C609 Size = dword ptr 10h
text:0040C609
push ebx
push esi
push [esp+8+Size] ; Size
mov esi, [esp+0Ch+arg_0]
push [esp+0Ch+Src] ; Src
mov ecx, 1024
```

由于 SENTINEL HASP 是按请求在运行时解密文件，而不是一次将所有.class 文件加载到虚拟机，因此可以防止黑客重建整个应用程序。

```
sub ecx, 1000
add ecx, 6
sub ecx, 30
push ecx

push dword ptr [esi+0Ch] ; int
call sub_43AF3E
push ecx
pop ecx

mov  ecx, [esi+4]
mov        edx, [ecx]
mov ebx, 2000
add ebx, 2000
add ebx, 2000
sub ebx, 6000
cmp  eax, 1000h
mov eax, 1000h
setnle bl
push ebx
push ecx
call   dword ptr [edx+1Ch]
add esp, 18h
xor eax, eax
pop esi
inc eax
pop ebx
retn
```

在此示例中，我们通过加入算术指令扩展了该操作码片段。该代码现在更加难以阅读。由于不能简单地从片段阅读代码，潜在的黑客必须执行计算才能理解其内容。

虽然这个特殊的示例仍然很容易理解，但我们可以根据需要的次数来重复该类型的混淆，将这个简单的代码段扩展为超过 1000 行代码，从而使代码内容变得难以理解。当使用了混淆时，攻击者必须制作特殊工具才能理解原始代码段，而这不是一件容易的事。

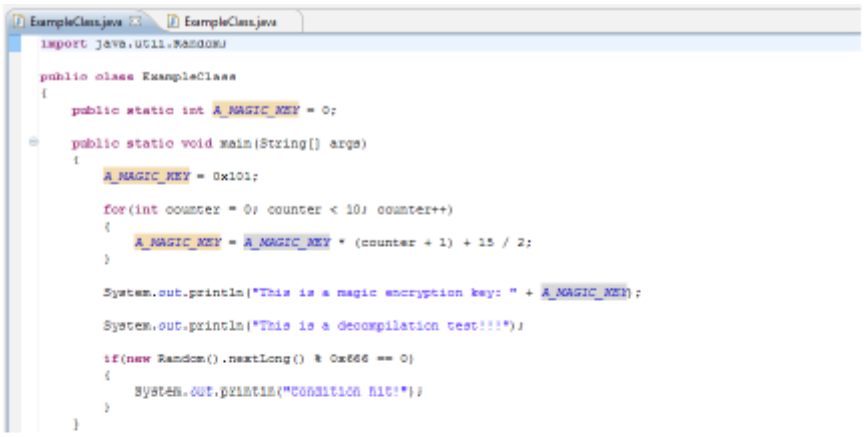
有几种不同类型的混淆方法可用于指令集：程序员可以通过替换二进制代码中的所有字符串来混淆代码，这样就更加难以找到一个好的切入点来开始逆向工程攻击。他们可以通过插入指向垃圾代码的跳转并返回来迷惑攻击者，或混淆源代码或字节码。

开发人员也可以选择使用名为 Const2Code 转换的技术来混淆常量。例如，密钥有时在应用程序中存储为一组字节。如果黑客确定了这些字节的位置，他们就可以访问这些字节。为混淆一个常量（如密钥），Const2Code 算法将常量转换为可产生同一常量的多个不同的命令。

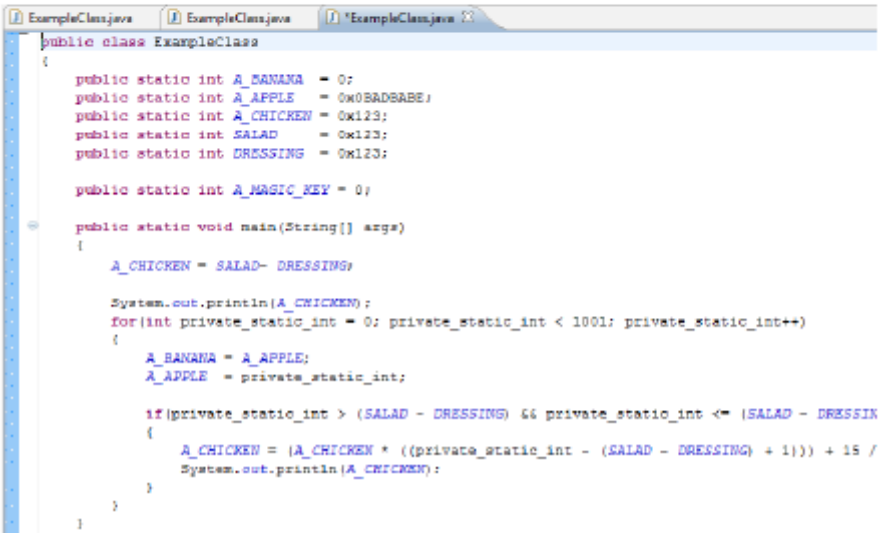
例如，为了在源代码段中隐藏常量 cst=0x12345678，可以将该常量简单地分为几个算术运算，如加、减等等。A = 0x9ABCDF00；B=0x2；C=0x135799E00。现在让我们使用这三个变量重新计算出常量 0x12345678。我们的原始常量为：cst = C / B + A – 0x88888888 – A + 1000= 0x12345678。如果应用程序仅使用该例程来计算常量，攻击者就必须理解其中的含义，而不是简单地获取该常量。

现在让我们来看一个源代码混淆的示例。

图 2 包含的原代码已使用 C2C 算法进行混淆，隐藏了密钥。



```
import java.util.Random;

public class ExampleClass
{
    public static int A_MAGIC_KEY = 0;

    public static void main(String[] args)
    {
        A_MAGIC_KEY = 0x101;

        for(int counter = 0; counter < 10; counter++)
        {
            A_MAGIC_KEY = A_MAGIC_KEY * (counter + 1) + 15 / 2;
        }

        System.out.println("This is a magic encryption key: " + A_MAGIC_KEY);

        System.out.println("This is a decompilation test!!!");

        if(new Random().nextLong() % 0x666 == 0)
        {
            System.out.println("Condition hit!");
        }
    }
```



```
public class ExampleClass
{
    public static int A_BANANA  = 0;
    public static int A_APPLE   = 0x0BADBABE;
    public static int A_CHICKEN = 0x123;
    public static int SALAD     = 0x123;
    public static int DRESSING  = 0x123;

    public static int A_MAGIC_KEY = 0;

    public static void main(String[] args)
    {
        A_CHICKEN = SALAD- DRESSING;

        System.out.println(A_CHICKEN);
        for(int private_static_int = 0; private_static_int < 1001; private_static_int++)
        {
            A_BANANA = A_APPLE;
            A_APPLE  = private_static_int;

            if(private_static_int > (SALAD - DRESSING) && private_static_int <= (SALAD - DRESSIN
            {
                A_CHICKEN = (A_CHICKEN * ((private_static_int - (SALAD - DRESSING) + 1))) + 15 /
                System.out.println(A_CHICKEN);
            }
        }
    }
```

图 2 -可混淆的 Java 源代码示例 - 这是一个简单的代码段，使用原始的 C2C 算法计算密钥

图 3 - 手动源代码混淆的 Java 源代码示例

如图 3 所示，在进行手动源代码混淆后，该示例变得非常不同。乍看上去将无法看出手动混淆的代码是用于计算相同的密钥。

在这种情况下，不仅是源代码不同，生成的字节码也完全不同。由于可以生成大量字节码，源代码混淆的程序将变更加难以分析。

SafeNet Sentinel：一种更简便的封装方式

SENTINEL HASP Envelope 是一个自动文件包装器，通过文件加密和本机代码混淆提供针对软件逆向工程的强有力保护。这确保了嵌入软件中的算法、商业秘密和专业知识对于黑客是安全的。SENTINEL HASP 通过将 Java 应用程序重新关联到硬件平台，提供了高度安全的知识产权保护。这迫使攻击者在破解受保护的 Java 应用程序时不仅需要对 Java 应用程序进行逆向工程，还需要对本机代码进行逆向工程。因此，攻击者必须更加富有经验才能破解已封装的 Java 代码。

LicensingLive!™ (lahy'sun sing lahyv'), adj. n. [SAFENET, INTERACTIVE] 1. 立即访问与软件包装、定价、实现、交付和管理相关的最佳做法和新出现的挑战。2. 一个汇集软件供应商、行业分析师、许可顾问和技术供应商的论坛。

SENTINEL HASP 为受保护应用程序与保护密钥之间的通信提供安全通道，从而消除了中间人攻击。Java Envelope 使用这种能力来防止黑客通过截取通信访问保护密钥发回的数据。

由于 SENTINEL HASP 是按请求在运行时解密文件，而不是一次将所有.class 文件加载到虚拟机，因此可以防止黑客重建整个应用程序。

结论

虽然 JVM 为开发人员提供了编写一次应用程序即可在几乎任何平台上运行的能力，但这种体系结构具有使黑客易于对源代码进行逆向工程、篡改或盗窃的重大缺陷。众多的商业反汇编程序进一步简化了这一过程。而且虽然 Java 确实提供了一些安全措施，但包括的这些能力都不足以阻止攻击者。加密和混淆等技术通常用于减缓攻击，但仍然留有漏洞。封装将加密与本机代码混淆相结合来提供目前最强的保护，可以保护知识产权。通过使用 SENTINEL HASP 解决方案，您可以获得封装的诸多好处，而不必花费时间和精力来开发新的解决方案。

SafeNet Sentinel 软件货币化解决方案

SafeNet 在为全世界的软件和技术供应商提供创新和可靠的软件许可、授权和管理解决方案方面拥有 25 年的行业经验。我们的 Sentinel®软件货币化解决方案系列易于集成和使用，具有创新性并注重功能，对于任何规模、技术要求或组织结构的任何机构，都能满足其独特的许可证启用、执行和管理要求。客户只有使用 SafeNet 的产品才能够应对所有的反盗版、知识产权保护、许可证启用和许可证管理挑战，同时提高公司盈利能力、改进内部运营、维持竞争优势并巩固与客户及最终用户的关系。SafeNet 在适应新的需求和引入新技术以应对不断变化的市场环境方面具有丰富的成功经验，我们在全球的 25,000 多家客户已经认识到选择 Sentinel 就意味着选择了在现在和未来自由开展业务的主动权。

要下载免费的 SENTINEL HASP 开发工具包，请访问以下网址：
http://www3.safenet-inc.com/Special/hasp/safenet-hasp-srm-order/default.asp

加入谈话

沈阳云畅想科技有限公司
电话：024-3105 8958
网址：www.aladdin.ln.cn