



(12)发明专利申请

(10)申请公布号 CN 110472425 A

(43)申请公布日 2019.11.19

(21)申请号 201910649077.4

(22)申请日 2019.07.18

(71)申请人 福建天晴在线互动科技有限公司

地址 350001 福建省福州市长乐区湖南镇
大鹤村

(72)发明人 刘德建 官泉 陈宏展

(74)专利代理机构 福州市博深专利事务所(普
通合伙) 35214

代理人 张明

(51)Int.Cl.

G06F 21/60(2013.01)

G06F 21/12(2013.01)

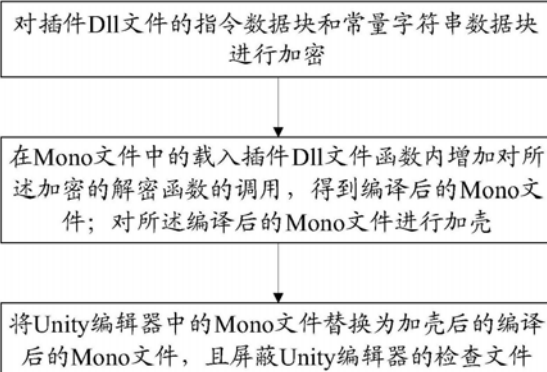
权利要求书1页 说明书8页 附图4页

(54)发明名称

基于Mono的Unity插件加密方法、存储介质

(57)摘要

本发明提供一种基于Mono的Unity插件加密方法、存储介质,方法包括:对插件D11文件的指令数据块和常量字符串数据块进行加密;在Mono文件中的载入插件D11文件函数内增加对所述加密的解密函数的调用,得到编译后的Mono文件;对所述编译后的Mono文件进行加壳;将Unity编辑器中的Mono文件替换为加壳后的编译后的Mono文件,且屏蔽Unity编辑器的检查文件。本发明能够使逆向工程无法自动还原出插件D11文件的源码,从而确保共享插件D11的安全;而且确保插件能正确加载和执行;进一步地,还具备较高的加密效率和广泛的运用范围。



1. 一种基于Mono的Unity插件加密方法,其特征在于,包括:
对插件D11文件的指令数据块和常量字符串数据块进行加密;
在Mono文件中的载入插件D11文件函数内增加对所述加密的解密函数的调用,得到编译后的Mono文件;对所述编译后的Mono文件进行加壳;
将Unity编辑器中的Mono文件替换为加壳后的编译后的Mono文件,且屏蔽Unity编辑器的检查文件。
2. 如权利要求1所述的一种基于Mono的Unity插件加密方法,其特征在于,所述对插件D11文件的指令数据块和常量字符串数据块进行加密,具体为:
获取插件D11文件中的.Net目录结构体的结束地址,将其作为指令数据块的起始位置;
获取插件D11文件中的MetaData数据块的起始地址,将其作为指令数据块的结束位置;
加密所述指令数据块的起始位置至所述指令数据块的结束位置之间的数据内容;
获取插件D11文件中常量字符串数据块的位置和数据块长度;
依据所述位置和数据块长度对所述常量字符串数据块进行加密;
在插件D11文件中置入加密标识。
3. 如权利要求2所述的一种基于Mono的Unity插件加密方法,其特征在于,所述在Mono文件中的载入插件D11文件函数内增加对所述加密的解密函数的调用,具体为:
在Mono文件中增加对载入的插件D11文件是否存在所述加密标识的判断函数;
在Mono文件中增加对应指令数据块的第一解密函数,所述第一解密函数包括指令数据块的起始位置和结束位置的获取,以及对应加密后指令数据块的解密函数;
在Mono文件中增加对应常量字符串数据块的第二解密函数,所述第二解密函数包括常量字符串数据块的位置和数据块长度的获取,以及对应加密后常量字符串数据块的解密函数;
设置在Mono载入插件D11文件函数时调用所述判断函数,若判断结果为是,则调用所述第一解密函数和第二解密函数。
4. 如权利要求1所述的一种基于Mono的Unity插件加密方法,其特征在于,所述将Unity编辑器中的原始Mono文件替换为加壳后的编译后的Mono文件,具体为:
通过补丁应用将Unity编辑器中EmbedRuntime目录里的Mono文件以及支持MonoBleedingEdge的mono-2.0-bdwc.dll文件和mono-2.0-sgen.dll文件替换为所述加壳后生成的Mono文件以及支持MonoBleedingEdge的mono-2.0-bdwc.dll文件和mono-2.0-sgen.dll文件。
5. 如权利要求1所述的一种基于Mono的Unity插件加密方法,其特征在于,所述屏蔽Unity编辑器的检查文件,具体为:
对Unity编辑器中的组件更新检查文件进行删除或者重命名文件名。
6. 一种计算机可读存储介质,其上存储有计算机程序,其特征在于,所述程序在被处理器执行时,能够实现上述权利要求1-5任意一项所述的一种基于Mono的Unity插件加密方法所包含的步骤。

基于Mono的Unity插件加密方法、存储介质

技术领域

[0001] 本发明涉及Unity领域,具体涉及基于Mono的Unity插件加密方法、存储介质。

背景技术

[0002] 目前,在Unity二次开发人员与其他协作部门,如美术、策划、任务等部门进行协作开发时,开发人员需要提供Unity插件给相关协作人员。协作人员获取到插件后,在Unity编辑器内载入插件,进行使用;而插件一般由C#语言编写,被Mono或者.Net编译器编译成D11文件。由于D11文件容易被逆向工程直接查看源代码,因此具有代码泄漏的风险。

[0003] 外界虽然有Unity加固应用包的方案,但是并不适用于运行在Unity编辑器里的D11插件加密,也不适用在部门协作中这一具体应用场景。

[0004] 现有技术可参考的方案,在实际项目实践中发现存在以下缺点:

[0005] 1、采用全文加密,会使Unity编辑器加载插件D11文件时,D11文件被认作非托管D11而跳过Mono的载入和解析过程,导致加密方案失效。

[0006] 2、采用删除或者加密D11特征值,则会使Unity编辑器加载插件D11文件时,先行载入未解密的D11导入表,导致无法载入关联的D11,进而Unity编辑器程序崩溃。

[0007] 因此,有必要提供一种基于Mono的Unity插件加密方法、存储介质,以实现Unity插件的应用包进行保护。

[0008] 另外,申请号为CN201711260537.1,公开的《一种安卓应用的加固方法和装置》,其涉及对安卓应用的加固,与本申请的加密方式类似,但是至少存在以下本质区别:1、其只能适用于安卓平台,而无法适用于Unity的windows操作系统平台;2、并且,其声明的so文件为libmono.so;而本申请提供的方法为Mono.dll以及支持MonoBleedingEdge的mono-2.0-bdwgc.dll和mono-2.0-sgen.dll;3、其加密方式为对各个函数进行指令变换;而本申请无需去查找各个函数,而是直接找到指令数据块进行加密,速度更快,更高效;4、所述d11文件为Assembly-CSharp.dll文件;本申请的DLL文件为C#编译生成的D11文件,本申请的适应范围更广;5、其替换的libmono.so是在应用包打包完成后,进行加固操作时进行替换;而本申请替换的是unity编辑器里的,是开发状态的时候进行加固,在打包之前;6、其修改了用于安装到手机上的APK包;而本申请修改了WINDOWS电脑里美术、策划里安装的UNITY编辑器。

发明内容

[0009] 本发明所要解决的技术问题是:针对windows平台提供一种基于Mono的Unity插件加密方法、存储介质,且加密效率高。

[0010] 为了解决上述技术问题,本发明采用的技术方案为:

[0011] 一种基于Mono的Unity插件加密方法,包括:

[0012] 对插件D11文件的指令数据块和常量字符串数据块进行加密;

[0013] 在Mono文件中的载入插件D11文件函数内增加对所述加密的解密函数的调用,得到编译后的Mono文件;对所述编译后的Mono文件进行加壳;

[0014] 将Unity编辑器中的Mono文件替换为加壳后的编译后的Mono文件,且屏蔽Unity编辑器的检查文件。

[0015] 本发明提供的另一个技术方案为:

[0016] 一种计算机可读存储介质,其上存储有计算机程序,所述程序在被处理器执行时,能够实现上述基于Mono的Unity插件加密方法所包含的步骤。

[0017] 本发明的有益效果在于:本发明通过对插件D11文件的指令数据块和常量字符串数据块进行加密,使逆向工程无法自动还原出源码,从而起到保护插件D11文件安全的作用,且具有加密效率更高的特点;同时,还对Mono文件进行编译,通过增加自动解密函数,并确保其安全,以及替换原Mono文件和屏蔽检查,确保插件能正确加载和执行。本发明能够很好的运用在Unity项目中的部门间协作开发中,在共享插件时,有效抵御各类逆向应用的静态分析和还原源码,预防人事变动或其他原因引起的代码泄漏风险,保证共享插件D11文件的安全。

附图说明

[0018] 图1为本发明实施例一种基于Mono的Unity插件加密方法的流程示意图;

[0019] 图2为本发明实施例一和实施例二的基于Mono的Unity插件加密方法的分步骤流程示意图;

[0020] 图3为本发明实施例二中对插件D11文件加密过程中的修改示意图;

[0021] 图4为本发明实施例二中对插件D11文件加密前后在逆向应用中的分析结果对比图;

[0022] 图5为本发明实施例三中部门间协作流程示意图。

具体实施方式

[0023] 为详细说明本发明的技术内容、所实现目的及效果,以下结合实施方式并配合附图予以说明。

[0024] 本发明最关键的构思在于:对插件D11文件的指令数据块和常量字符串数据块进行加密,使逆向工程无法自动还原出源码;对Mono文件进行编译,确保插件能正确加载和执行。

[0025] 本发明涉及的技术术语解释:

[0026]

技术术语	解释
Mono	基于.NET 框架的开放源码开发平台，用其构建的应用程序可运行在不同硬件和操作系统上
Dll 文件	本文特指由.NET 语言的代码所编译生成的文件
插件	本文特指用于 Unity 编辑器在开发环境中运行的扩展程序
.Net Directory	Windows 的 PE 格式中的 .Net 目录数据
MetaData	Windows 的 PE 格式中的 Meta 数据
MetaData RVA	Meta 数据块的起始内存偏移地址
MetaData Steams	Meta 数据块的数据体

[0027]

Dos Stub	和 MS-DOS 2.0 兼容的可执行体
#US	常量字符串数据块的特征名称
MonoBleedingEdge	Unity 里支持.NET4.0 以上版本的 Mono 项目

[0028] 请参照图1以及图2,本发明提供一种基于Mono的Unity插件加密方法,包括:

[0029] 对插件Dll文件的指令数据块和常量字符串数据块进行加密;

[0030] 在Mono文件中的载入插件Dll文件函数内增加对所述加密的解密函数的调用,得到编译后的Mono文件;对所述编译后的Mono文件进行加壳;

[0031] 将Unity编辑器中的Mono文件替换为加壳后的编译后的Mono文件,且屏蔽Unity编辑器的检查文件。

[0032] 进一步地,所述对插件Dll文件的指令数据块和常量字符串数据块进行加密,具体为:

[0033] 获取插件Dll文件中的 .Net 目录结构体的结束地址,将其作为指令数据块的起始位置;

[0034] 获取插件Dll文件中的 MetaData 数据块的起始地址,将其作为指令数据块的结束位置;

[0035] 加密所述指令数据块的起始位置至所述指令数据块的结束位置之间的数据内容;

[0036] 获取插件Dll文件中常量字符串数据块的位置和数据块长度;

[0037] 依据所述位置和数据块长度对所述常量字符串数据块进行加密;

[0038] 在插件Dll文件中置入加密标识。

[0039] 由上述描述可知,能够准确且高效地加密指定数据块,加密后方法内的源代码将

无法正常显示,从而有效防止逆向工具直接还原源代码;同时,也能准确且高效的对常量字符串数据块进行加密,从而增加逆向难度,将无法通过关键字字符串搜索找到对应的数据块;进一步地,通过增加加密标识,既能用于插件运行时Mono载入时识别,也能防止重复加密。

[0040] 进一步地,所述在Mono文件中的载入插件D11文件函数内增加对所述加密的解密函数的调用,具体为:

[0041] 在Mono文件中增加对载入的插件D11文件是否存在所述加密标识的判断函数;

[0042] 在Mono文件中增加对应指令数据块的第一解密函数,所述第一解密函数包括指令数据块的起始位置和结束位置的获取,以及对应加密后指令数据块的解密函数;

[0043] 在Mono文件中增加对应常量字符串数据块的第二解密函数,所述第二解密函数包括常量字符串数据块的位置和数据块长度的获取,以及对应加密后常量字符串数据块的解密函数;

[0044] 设置在Mono载入插件D11文件函数时调用所述判断函数,若判断结果为是,则调用所述第一解密函数和第二解密函数。

[0045] 由上述描述可知,通过第一解密函数的调用,能使Unity编辑器在加载D11文件时能正确解析指令;通过第二解密函数的调用,能使Unity编辑器在加载D11文件时能正确显示字符串数据;进而保证正确地载入和运行已加密的插件D11文件。

[0046] 进一步地,所述将Unity编辑器中的原始Mono文件替换为加壳后的编译后的Mono文件,具体为:

[0047] 通过补丁应用将Unity编辑器中EmbedRuntime目录里的Mono文件以及支持MonoBleedingEdge的mono-2.0-bdwc.dll文件和mono-2.0-sgen.dll文件替换为所述加壳后生成的Mono文件以及支持MonoBleedingEdge的mono-2.0-bdwc.dll文件和mono-2.0-sgen.dll文件。

[0048] 由上述描述可知,通过补丁应用的对Unity编辑器进行补丁,使Unity编辑器能正常载入和解析加密的D11文件。

[0049] 进一步地,所述屏蔽Unity编辑器的检查文件,具体为:

[0050] 对Unity编辑器中的组件更新检查文件进行删除或者重命名文件名。

[0051] 由上述描述可知,能够使Unity编辑器在载入已加密的插件D11文件时跳过指令数据块的检查,从而能进入Mono的加载过程进行解密。

[0052] 本发明提供的另一个技术方案为:

[0053] 一种计算机可读存储介质,其上存储有计算机程序,所述程序在被处理器执行时,能够实现上述任意一项所述的一种基于Mono的Unity插件加密方法所包含的步骤。

[0054] 从上述描述可知,对应本领域普通技术人员可以理解实现上述技术方案中的全部或部分流程,是可以通过计算机程序来指令相关的硬件来实现的,所述的程序可存储于一计算机可读的存储介质中,该程序在执行时,可包括如上述各方法的流程。所述程序在被执行后,同样能实现各方法对应的有益效果。

[0055] 其中,所述的存储介质可以是磁盘、光碟、只读存储记忆体(Read-Only Memory, ROM)或随机存储记忆体(Random Access Memory, RAM)等。

[0056] 实施例一

[0057] 本实施例提供一种基于Mono的Unity插件加密方法,能够对插件D11文件进行有效

保护,使逆向工程无法正确还原出源码;且能保证Unity编辑器正确地加载和执行插件;更进一步地,还具备加密高效且适用范围广等优点。

[0058] 如图2所示,本实施的方法包括:

[0059] 一、开发加密应用

[0060] 开发能够对C#编译生成的插件D11文件的指令数据块和常量字符串数据块进行加密,并植入自定义加密标记的加密应用。优选地,使用高级对称加密算法进行加密。

[0061] 该操作能使逆向应用无法还原出插件D11文件的源代码,也无法正确还原出常量字符串;但是,依然保留了关键信息用于Unity编辑器能正确识别该文件。

[0062] 二、对开源的Mono项目进行改造

[0063] 首先,编写对插件D11文件的上述指令数据块和常量字符串数据块进行解密的解密函数,并设置在Mono项目载入D11的函数内调用该函数。其次,是对编译后的Mono项目进行加壳。

[0064] 该操作能使Unity编辑器能解析并在内存中还原该D11文件的指令和常量字符串,使编辑器内的其他脚本能正确调用和运行。

[0065] 三、编写Unity补丁应用

[0066] 通过补丁应用将Unity编辑器中原始的Mono文件替换成改造后的Mono文件,并且屏蔽Unity编辑器的检查文件防止载入失败。

[0067] 该操作能使Unity编辑器能正常载入和解析加密的D11文件。

[0068] 另外需要注意的是,本实施例中,所述插件D11文件是C#编译生成的D11文件,相较于现有技术对Assembly-CSharp.dll文件这类D11文件进行操作而言,本实施例的适应范围更广。并且,本实施例的替换Mono文件的操作是在开发状态的时候进行加固,在打包之前,因此安全性更高。

[0069] 本实施例特别适用于部门间协作场景下需要对D11插件进行加密保护,并且还需要确保能在编辑器里正常加载和运行的运用场景。

[0070] 实施例二

[0071] 本实施例对实施例一的方法做进一步限定。

[0072] 请参阅图2,本实施例的方法包括:

[0073] 一、插件D11文件加密

[0074] 因为用于Unity的插件D11是由C#编写的,之后将由Mono解析并且载入。所以可以对插件D11进行修改,为此需要分析windows的PE文件结构。具体过程如下:

[0075] 1.1、加密指令数据块

[0076] 首先找到D11文件里的.Net Directory,即.Net目录结构体的结束地址,将其作为指令数据块的起始位置;

[0077] 其次找到D11文件里的.Net Directory,即.Net目录结构体的MetaData RVA,即Meta数据偏移地址字段,将其重定向偏移后的位置,即MetaData数据块的起始地址作为指令数据块的结束位置;

[0078] 之后加密指令数据块起始位置到结束位置的数据内容。可以使用简单异或,或者取反等自定义的加密算法,也可以使用公开的现有技术,如AES,DES,TDEA,RC5等高级对称加密算法进行加密或者多次加密。

[0079] 该步骤能防止逆向工具直接还原源代码,加密后方法内的源代码将无法正常显示。

[0080] 1.2、加密常量字符串数据块

[0081] 找到D11文件里的.Net Directory内的MetaData Steams内的#US数据块,即常量字符串数据块,获取其位置和数据块长度,据此对常量字符串数据块进行加密。

[0082] 该步骤将增加D11文件中常量字符串被逆向的难度,将无法通过关键字字符串搜索找到对应的数据块。

[0083] 1.3、植入已加密标记

[0084] 在加密后的D11文件内植入自定义的已加密标记,并将对称加密的密钥或者其他需要的内容填充于Dos Stub数据块或者其他用于对齐的空白数据块里。

[0085] 该步骤用于插件运行时Mono载入时识别,也可以防止重复加密。

[0086] 对原插件D11文件的修改见图3所示。

[0087] 二、改造Mono

[0088] 为了能够正确地载入已加密的插件D11文件,故需要对开源的Mono项目进行改造。具体包括:

[0089] 2.1、增加判断已加密标记

[0090] 根据之前在D11文件内植入的标记与那些未加密的D11文件进行区分,对于存在标记的D11将在后续执行解密操作。

[0091] 2.2、增加解密指令数据块函数

[0092] 将预先编写好的解密指令数据块函数置入Mono文件中,并设置在载入插件D11文件函数时调用。

[0093] 解密指令数据块函数将先后执行:

[0094] 首先在Mono项目内的mono_image_load_cli_header函数里获取.Net Directory即.Net目录结构体在内存中的结束地址,将其作为指令数据块的起始位置;

[0095] 其次获取.Net Directory即.Net目录结构体的MetaData RVA即Meta数据偏移地址作为指令数据块的结束位置;

[0096] 之后对应指令数据块的起始位置和结束位置进行解密。

[0097] 该步骤能使Unity编辑器在加载D11文件时能正确解析指令。

[0098] 2.3、增加解密常量字符串函数

[0099] 同理,将预先编写好的解密常量字符串函数置入Mono文件中,并设置在载入插件D11文件函数时调用。

[0100] 解密常量字符串函数,将先后执行:

[0101] 首先在Mono项目内的load_metadata_ptrs函数里获取#US数据块,即常量字符串数据块,获取其位置和数据块长度。

[0102] 之后依据常量字符串数据块的位置和数据块长度解密常量字符串数据块。

[0103] 该步骤能使Unity编辑器在加载D11文件时能正确显示字符串数据。

[0104] 2.4、Mono加壳

[0105] 采用外界现有的加壳或者加固方案,对改造后的Mono进行保护。

[0106] 该步骤防止解密函数被逆向应用分析,进而破解插件D11文件,还原出源代码。

[0107] 三、对Unity编辑器进行补丁

[0108] 补丁应用在找到需要加载加密插件D11文件的Unity编辑器路径后,需要对该Unity编辑器的文件进行替换和修改,以达到正常解密的目的。

[0109] 补丁应用具体执行包括:

[0110] 3.1、替换Mono

[0111] 将重新编译Mono工程生成全新的含有能解析被加密的插件D11文件的Mono.dll以及支持MonoBleedingEdge的mono-2.0-bdwc.dll和mono-2.0-sgen.dll。通过补丁应用将Unity编译器对应版本内的EmbedRuntime目录里的Mono.dll以及支持MonoBleedingEdge的mono-2.0-bdwc.dll和mono-2.0-sgen.dll替换掉。

[0112] 该步骤能使Unity编辑器能正常载入和解析加密的D11文件。

[0113] 3.2、屏蔽检查文件

[0114] 还需要对Unity编辑器里的AssemblyUpdater.exe即组件更新检查文件进行屏蔽,可以通过重命名文件名或者删除文件的方式。

[0115] 该步骤能使Unity编辑器在载入已加密的插件D11文件时会跳过指令数据块的检查,从而能进入Mono的加载过程进行解密。

[0116] 本实施例在实际项目操作中得到了很好实践检验。通过对插件D11文件的指令数据块和常量字符串进行加密,使逆向工程无法自动还原出源码。D11文件加密前后在逆向应用里的解析效果对比如图4所示,可见,加密后的D11文件逆向失败,无法正确还原出源代码。另外,通过使用补丁应用对Unity编辑器进行补丁,保证插件能被正确加载和执行。

[0117] 将本实施例的方法应用于Unity项目中的部门间协作开发,共享加密后的插件,有效抵御各类逆向应用的静态分析和还原源码,预防人事变动或其他原因引起的代码泄漏风险。

[0118] 实施例三

[0119] 本实施例对应实施例一或实施例二,提供一个具体运用场景:

[0120] 方案针对在Unity编辑器里运行的含有C#编译的D11插件进行加密的方法:

[0121] 首先,编写加密应用,具体为应用允许用户输入插件D11文件的路径,应用读取该路径的文件,判断文件是否由C#编写,是否无加密标记,如果是,则执行加密指令数据块函数和加密常量字符串数据块函数;对文件增加已加密标记。

[0122] 其次,针对每个Unity版本的Mono工程生成一份带有解密函数的mono.dll文件。具体是先获取Mono开源项目的源码,然后对Mono源码中添加判断载入的D11是否含有加密标记,如果是,则在后续解析函数中执行解密指令数据块和常量字符串数据块的函数;最后使用现有技术进行加固。

[0123] 最后,编写补丁应用,应用包含了对应Unity版本的带有解密函数的mono.dll文件。具体是应用会遍历用户电脑所安装的Unity版本与当前可进行补丁的版本进行比较,然后对可进行补丁的Unity进行替换Mono文件,然后屏蔽组件更新检查文件。

[0124] 如图5所示,在具体应用场景时,插件提供方如Unity编辑器的二次开发人员,使用加密应用对原始的未加密的插件D11文件进行加密,加密应用会生成加密后的插件D11文件。

[0125] 插件提供方将加密后的插件D11文件和补丁应用提供给插件使用方。

[0126] 插件使用方如美术,策划,任务等部门协作人员,收到补丁应用后,先对电脑里的Unity编辑器进行补丁操作,其会使Unity编辑器能正常加载和运行加密后的插件D11文件。之后插件使用方打开Unity编辑器,加载加密后的插件D11文件,即可正常运行。

[0127] 而插件使用方后续因人事变动或者其他原因导致插件D11文件和补丁应用外流,均无法用逆向应用直接还原源代码,有效保护插件提供方以及公司相关权益。

[0128] 实施例四

[0129] 本实施例对应实施例一至实施例三,提供一种计算机可读存储介质,其上存储有计算机程序,所述程序在被处理器执行时,能够实现上述实施例一至实施例三任意一个实施例所述的一种基于Mono的Unity插件加密方法所包含的步骤。具体的步骤内容在此不进行复述,详细请参阅实施例一至实施例三。

[0130] 综上所述,本发明提供一种基于Mono的Unity插件加密方法、存储介质,能够使逆向工程无法自动还原出插件D11文件的源码,从而确保共享插件D11的安全;而且确保插件能正确加载和执行;进一步地,还具备较高的加密效率和广泛的运用范围。

[0131] 以上所述仅为本发明的实施例,并非因此限制本发明的专利范围,凡是利用本发明说明书及附图内容所作的等同变换,或直接或间接运用在相关的技术领域,均同理包括在本发明的专利保护范围内。

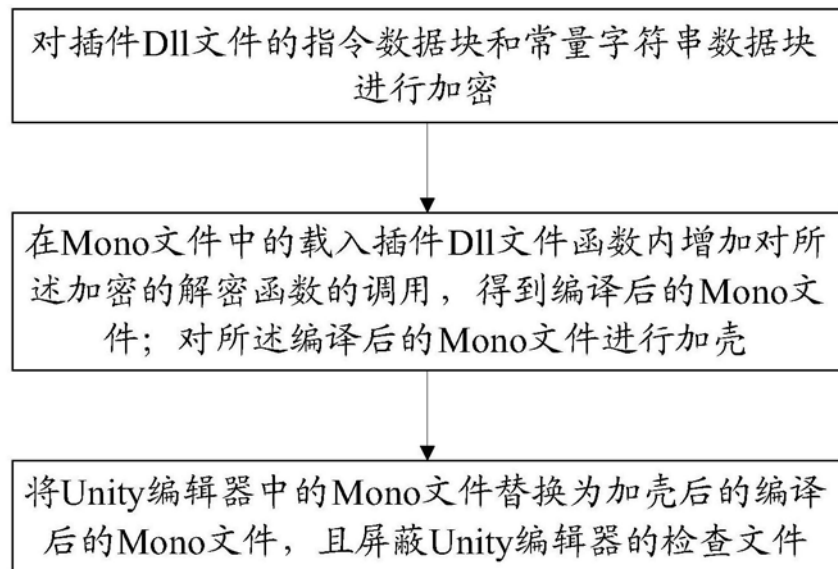


图1

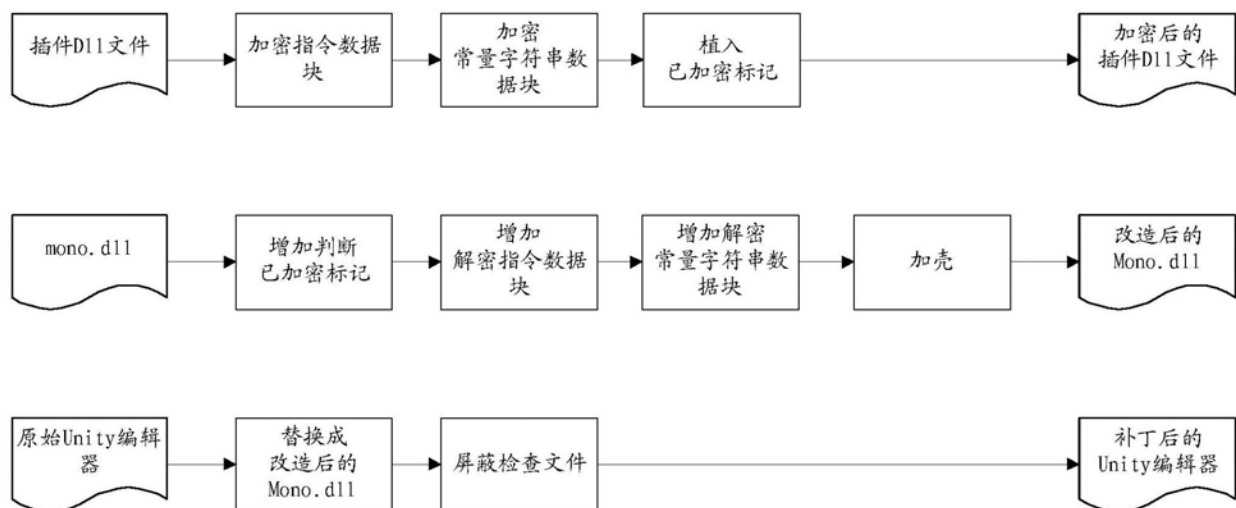


图2

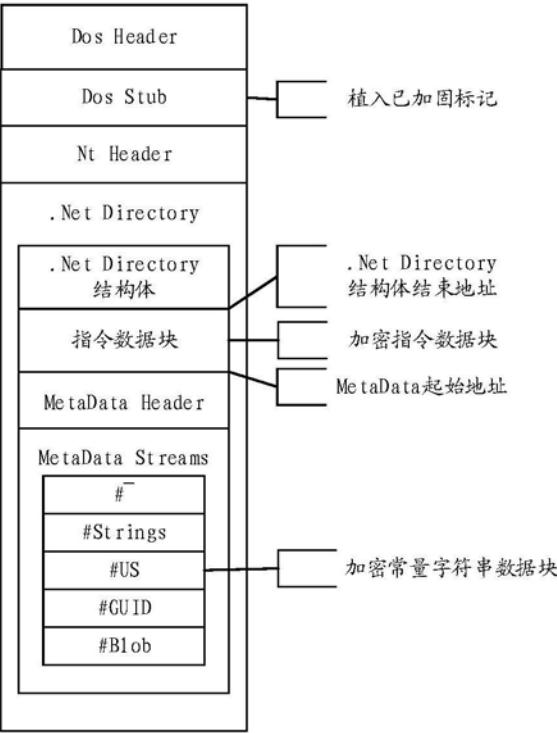


图3

Start() : void @06000001 X DLLTest.dll

```

1 // LogicSP
2 // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00002050
3 private void Start()
4 {
5     MonoBehaviour.print("script start");
6     int num = 0;
7     for (int i = 0; i < 100; i++)
8     {
9         num += i;
10    }
11    switch (1)
12    {
13        case 0:
14            num++;
15            break;
16        case 1:
17            num += 2;
18            break;
19        case 2:
20            num += 4;
21            break;
22        case 3:
23            num++;
24            break;
25        case 4:

```

加密前

Start() : void @06000001 X DLLTest.dll

```

1 // LogicSP
2 // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00002050
3 private void Start()
4 {
5     /*
6     An exception occurred when decompiling this method (06000001)
7
8     ICSharpCode.Decompiler.DecompilerException: Error decompiling System.Void LogicSP::Start()
9     ----> System.ArgumentOutOfRangeException: 索引超出范围。必须为非负值并小于集合大小。
10    参数名: index
11    在 System.ThrowHelper.ThrowArgumentOutOfRangeException(ExceptionArgument argument, ExceptionResource resour
12    在 System.Collections.Generic.List`1.get_Item(Int32 index)
13    在 dnlib.DotNet.ParameterList.dnlib.Threading.Collections.IList<dnlib.DotNet.Parameter>.Get_NoLock(Int32 ir
14    在 dnlib.DotNet.ParameterList.get_Item(Int32 index)
15    在 ICSharpCode.Decompiler.ILAst.ILCodeUtil.ExpandMacro(ILCode& code, Object& operand, MethodDef method)
16    在 ICSharpCode.Decompiler.ILAst.ILAstBuilder.StackAnalysis(MethodDef methodDef)
17    在 ICSharpCode.Decompiler.ILAst.ILAstBuilder.Build(MethodDef methodDef, Boolean optimize, DecompilerContext
18    在 ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(IEnumerable`1 parameters, MemberMapping
19    在 ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(MethodDef methodDef, DecompilerContext
20    --- 内部异常堆栈跟踪的结尾 ---
21    在 ICSharpCode.Decompiler.Ast.AstMethodBodyBuilder.CreateMethodBody(MethodDef methodDef, DecompilerContext
22    在 ICSharpCode.Decompiler.Ast.AstBuilder.CreateMethodBody(MethodDef method, IEnumerable`1 parameters, Membe
23    */;
24 }

```

加密后

图4

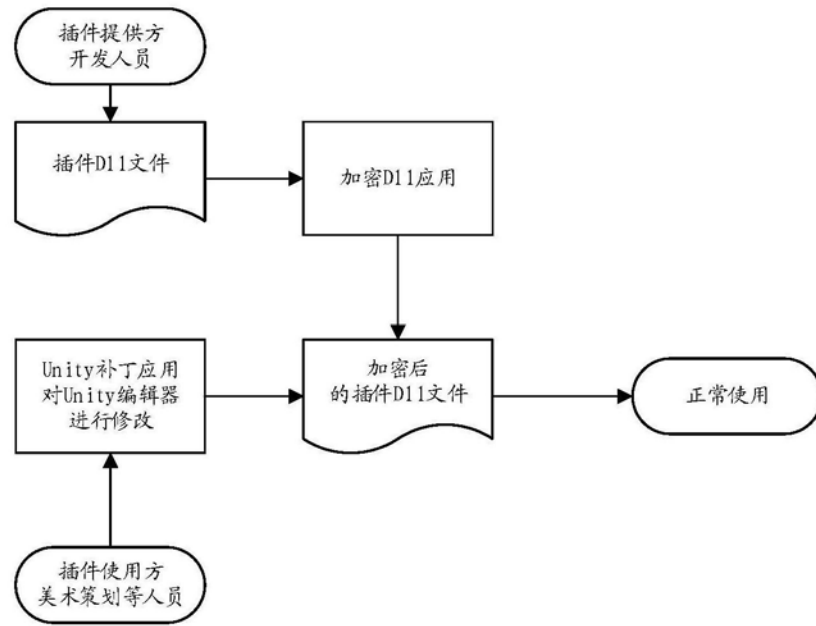


图5