基于DevSecOps的软件供应链安全治理技术简析

Analysis of Software Supply Chain Security Governance Technology Based on DevSecOps

张小梅，苏俐竹（中国联通研究院，北京 100048）
Zhang Xiaomei，Su Lizhu（China Unicom Research Institute，Beijing 100048，China）

摘　要：
研究了软件供应链的安全治理技术，解决软件供应链所面临的安全风险，构建软件供应链安全治理体系，形成业务应用软件设计、编码、测试、运营全生命周期的防护方案，实现对软件全流程链条的安全治理，提升应用系统的自身免疫力。

关键词：
DevSecOps；软件供应链；应用自保护
doi：10.12045/j.issn.1007-3043.2022.09.004
文章编号：1007-3043（2022）09-0013-06
中图分类号：TN915.08
文献标识码：A
开放科学（资源服务）标识码（OSID）：

Abstract：
It introduces the security governance technology of the software supply chain，solves the security risks faced by the software supply chain，constructs the security governance system of the software supply chain，which forms the protection scheme for the whole life cycle of software application design，coding，testing and operation，realizes the security governance of the whole software process chain，and improves the self immunity of the application system.

Keywords：
DevSecOps；Software supply chain；RASP

引用格式：张小梅，苏俐竹 . 基于DevSecOps的软件供应链安全治理技术简析[J]. 邮电设计技术，2022（9）：13-18.

0 前言

在数字化转型背景下，作为云原生技术不可或缺的持续交付、DevOps、微服务和容器技术正在被广泛使用，面对软件交付周期所带来的压力，DevSecOps理念被提出，DevSecOps是一种融合了开发、安全及运营理念的全新的安全管理模式，其核心理念是：业务应用生命周期的每个环节都需要为安全负责，安全是整个IT团队（包括开发、测试、运维及安全团队）所有成员的责任，并且需要贯穿到从研发至运营的全过程。

收稿日期：2022-07-29

传统研发运营安全侧重于在测试及运营阶段，进行安全威胁排除，漏洞修复，更多的是被动式安全防御。面对云原生时代业务需要频繁调整、上线，亟需进行安全左移，在需求、研发阶段便进行安全介入，从源头处降低安全风险，实现主动式安全防御，从而构建覆盖软件应用服务全生命周期的安全体系。本文基于DevSecOps理念，在明确业务系统安全需求的前提下，制定贯穿软件系统全生命周期的实践方案，通过在软件开发的不同阶段将安全工具或安全活动进行整合，将各个信息安全孤岛进行串联，协同作战，实现安全设计、安全编码、安全测试和安全运营的统一融合，在提升软件系统研发过程标准化与自动化的同

时，降低对效率的影响，也降低安全的成本。

1 软件供应链安全治理思路

依托DevSecOps理念，通过构建安全工具链、整合安全流程、嵌入安全自动化检查等方式，将安全贯穿设计、开发、测试、运营生命周期的每个环节，推动研发运营安全体系向敏捷化、自动化演进，应对新的云服务开发运维模型，提高业务系统安全问题修复效率、降低安全运营成本。图1所示为软件供应链安全治理体系。

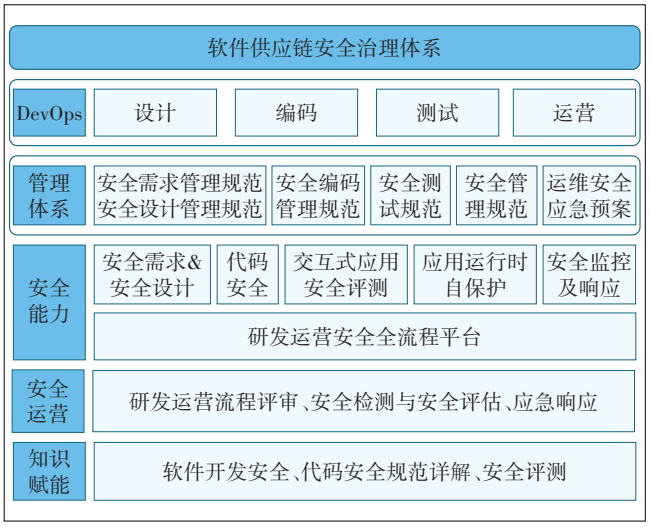

图1 软件供应链安全治理体系

2 软件供应链安全治理方案

2.1 设计安全

应用软件安全设计是保证软件安全的最根本前提，需要根据业务安全策略、合规需求、应用系统的特性等进行快速的分类分级，并根据系统分类分级情况分别定义不同的安全策略，进而选取合适的安全活动（见图2）。在计划阶段通常涉及的安全活动主要包含安全培训、威胁建模、安全开发衡量指标制定。

a）安全培训。DevSecOps流程方面需要让项目组成员清楚了解各个阶段需要承担的责任与义务，并进一步对DevSecOps体系整体过程进行学习。DevSecOps工具使用方面需要对项目组成员进行培训，如对开发人员、安全人员进行安全工具的使用培训，对运维管理人员进行安全基线工具、资产安全管理等工具的使用培训。

b）威胁建模。为满足DevOps快速敏捷要求，可采用轻量级威胁建模方式，主要包括如下几个步骤：

图2 安全设计活动

首先，通过动态调查问卷快速收集有关业务系统、部署环境和合规性要求等方面的重要且详细的信息，根据收集到的信息自动识别风险、威胁和潜在弱点；其次，根据预定义的安全和合规性策略对整体风险进行分类管理；最后，输出安全设计方案，并对安全问题进行跟踪。

c）衡量指标。根据业务应用系统的不同安全级别，制定对应的安全开发衡量指标，用于评估实施效果。

2.2 编码安全

应用软件编码阶段是业务安全问题修复成本最低的时期，依靠安全编码规范、安全基础库SDK的调用以及开源组件安全检测，最大程度地保证在编码阶段消除安全风险，保证软件的原生安全（见图3）。

首先，针对OWASP TOP10 2021和OWASP TOP10 2017涉及的漏洞类型制定安全编码规范，为应用系统的代码编写提供安全编码方法、安全编码用例、缺陷

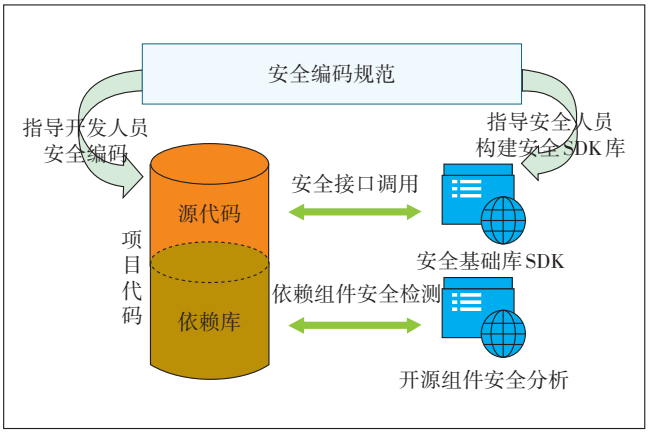

图3 安全编码方案

代码示例和安全接口函数等，保证应用系统的编码安全。下面给出几个SQL注入漏洞安全编码规范示例。

a）采用预编译方式，使用占位符接收数据替换为SQL语句（见图4）。

```
public class PreparedStatementTest {
    public static void main(String[] args)throws Throwable {
        //使用“?”占位符进行参数替换
        String sql=“select*from user where id=?”;
        try {
            //对SQL语句进行预翻译
            PreparedStatement statement=con.PreparedStatement(sql);
            //获取请求中的id字段传入SQL语句中，替换之前的占位符
            Statement.setInt(1,Integer.parseInt(req.getParameter(“id”)));
            //执行最终的查询SQL

            statement.executeQuery();
        } catch (SQLException throwables){
            Throwables.printStankTrace();
        }
}
```

图4　使用占位符接收数据替换为SQL语句示意

b）采用转义用户输入方式，将可能恶意执行的字符转义为无法执行的字符串（见图5）。

```
    public static String mysq1Encode(String str) {
        //创建一个StringBuilder对象，用于接收转移后字符
        StringBuilder sb=new StringBuilder();
        for(int i=0;i<str.length();i++) {
            //对原字符串进行逐字符处理
            char src = str.charAt(i);
            switch (src) {
            case '\'':
                    //单引号替换为两个单引号进行转义
                    sb.append("''");
                    break;
            case '\"':
            case '\\':
            case '%':
            case '\=':
                    //双引号、斜杠、百分号、等于号替换为空格
                    sb.append(' ');
             default:
                    //其他字符保持不变
                    sb.append(src);
                    break;
            }
        }
        //返回安全转义后的字符串
        return sb.toString();
    }
```

图5　将可能恶意执行的字符转义为无法执行的字符串示意

其次，开发安全基础库接口函数，打造安全基础库SDK即安全软件开发工具包，安全SDK集成了为软件工程师在开发过程中防御安全漏洞的一系列安全API，供软件项目直接引用。基于安全编码的SDK库，方便研发人员安全编码，提高开发效率，同时通过为代码漏洞修复提供安全接口，避免漏洞的二次引入。下面以防范SQL注入漏洞为例，介绍SDK包调研过程。

a）在项目中导入安全基础库SDK包（见图6）。

```
//导入安全基础库SDK包后，可以进行安全API调用
import com.tcsec.secsdk.sqli.Mysq1Encoder;
//创建mysq1Encode实例
Mysq1Encoder mysq1Encoder = Mysq1Encoder.getInstance();
//待转义的SQL查询语句，调用mysq1Encode方法对外部参数进行安全转义
String sq1=''select*from users where id=''+mysq1Encoder.mysq1Encode(id);
//执行SQL语句查询
ResultSet rs=s.executerQuery(encoder);
```

图6　项目中导入安全基础库SDK包示意

b）输入与输出结果对比，可以看到安全SDK的方法将用户输入的有害字符都做了相应的转义，可以防止SQL注入（见图7）。

```
输入：1 and 1=1
构造payload：select*from users where id= 1 and 1=1
未加入SDK输出sql语句：select*from users where id= 1 and 1=1
加入SDK输出sql语句：select null from users where id= 1 and 1=1
```

图7　输入与输出结果对比示意

最后，对编译后的二进制包和源码包进行分析，识别出项目包中引用的开源组件，检测软件系统的开源软件组成成分、开源组件安全漏洞、开源组件许可证风险等，提高软件开源组件风险整改效率。

2.3 测试安全

应用软件测试阶段利用智能动态污点跟踪技术，实现安全测试工作左移，将安全赋能至开发测试阶段，在软件的功能测试阶段同步开展安全测试，提高安全漏洞的发现效率和准确性，使交付更安全的产品成为可能（见图8）。

首先，智能动态污点跟踪技术依赖于字节码插桩技术，将安全检测Agent融入到应用程序中，代理程序Agent通过ASM框架等方式可以对Class字节码加载至JVM中时进行修改，从而插入跟踪的逻辑内容，达到污点跟踪的目的。图9为插桩后的StringBuilder.append()方法，插桩程序植入了Void.dance的代码作为埋点程序，后续对此方法的调用都会经由插桩程序接管。

其次，当Agent完成Class插桩后，在被测应用运行时，会由埋点代码进行智能动态污点跟踪。一次污点跟踪过程是在一个HTTP请求中进行的，污点跟踪随

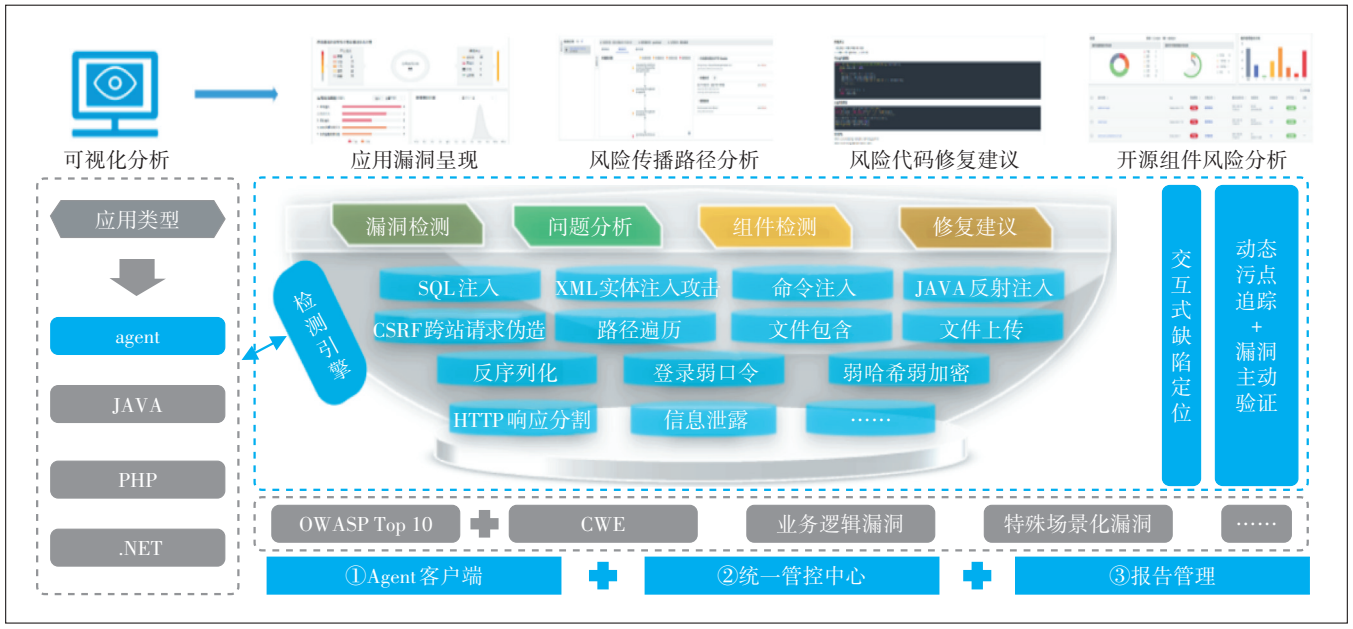

图8　交互式应用安全检测方案

```
public StringBuilder append(Object var1){
    try{
        Void.dance((Object)null,this,newObject[]{var1},''java.lang.
String Builder'',''append'',''(Ljava/lang/Object;)Ljava/lang/String-
Builder,'',124,false,true);
        StringBuilder var2=this.append(String.valueof(var1));
        Void.dance(var2,this,newObject[]{var1},'java.lang.String
Builder'',''append'',''(Ljava/lang/Object;)Ljava/lang/StringBuilder,
'',124,false,false);
        return var2;
    }catch(Throwable var3){
        throw var3
    }
}
```

图9　插桩后的StringBuilder.append()方法示意

着HTTP请求一起开始，HTTP请求的结束也标志着一次污点跟踪的结束（见图10）。智能动态污点跟踪的漏洞检测架构分为3个部分：污点源（Source）、传播过程（Propagator）和规则触发（Sink）。污点源是获取外部输入数据的地方，传播过程是污点数据进行传播的地方，规则触发是风险方法所在处。在应用程序运行过程中，Agent将会跟踪污点数据在应用程序中执行的程序上下文，并分析外部进入的数据在程序内部的传播过程，判断其是否经过了安全过滤或者安全验证机制，由此进行漏洞分析检测。

2.4 运营安全

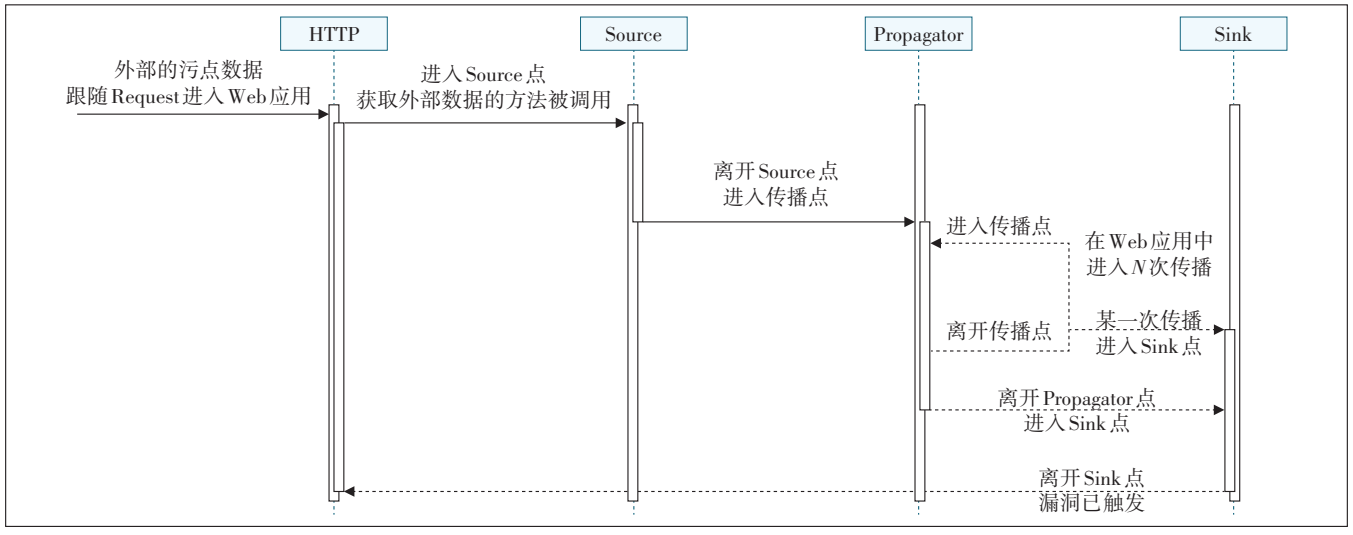

图10　污点跟踪过程时序图

应用发布运营阶段利用JavaAgent技术，在应用程序运行时动态编辑类字节码，将自身防御逻辑注入到Java底层API和Web应用程序当中，获取代码执行流程，监控敏感函数的调用，从而与应用程序融为一体，能实时分析和检测Web攻击，使应用程序具备自我保护能力。该技术能够利用请求的行为来判断是否为攻击请求，弥补WAF不能利用行为而仅用规则来判断请求及无法获取Web应用运行时环境的缺陷，并且不会受加密影响，可直接定位到漏洞代码文件和代码行号，补充了东西向防护能力，与边界防护能力联动，实现纵深安全防御，可实时监测Web攻击行为，增强0day漏洞应对能力。图11为SQL注入漏洞防御示例。

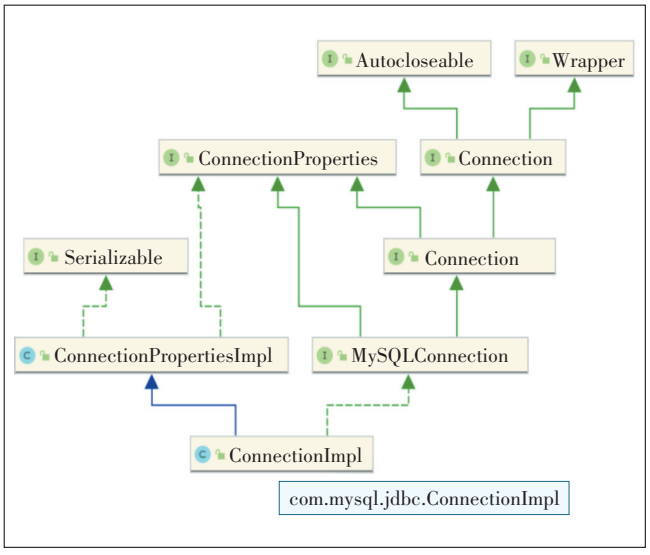

图11　com.mysql.jdbc.ConnectionImpl类继承关系图

在Java中，所有的数据库读写操作都需要使用JDBC驱动来实现，JDBC规范中定义了数据库查询的接口，例如Mysql驱动包实现数据库连接的实现类是：com.mysql.jdbc.ConnectionImpl，该类实现了com.mysql.jdbc.MySQLConnection接口，而com.mysql.jdbc.MySQLConnection类是java.sql.Connection的子类，也就是说com.mysql.jdbc.ConnectionImpl接口必须实现java.sql.Connection定义的数据库连接和查询方法。当com.mysql.jdbc.ConnectionImpl类被JVM加载后会因为配置了应用自保护的Agent，该类的字节码会传递到应用自保护的Agent处理，应用自保护经过分析后得出ConnectionImpl类符合应用自保护内置的ConnectionPrepareStatementHook类设置的Hook条件（父类名/方法名/方法参数完全匹配），那么应用自保护就会使用ASM动态生成防御代码并插入到被Hook的方法中。

图12给出了未修改的ConnectionImpl类代码片段示意，图13给出了经过应用自保护Agent修改后的代码片段示意。

```
package com.mysql.jabc;
public class ConnectionImpl extends ConnectionProperties Impl implements MySQLConntion {
        //省略其他业务代码
        public java.sql.PreparedStatement PrepareStatement(String sql) throws SQLExce ption {
        return preparedStatement(sql, DEFAULT_RESULT_SET_TYPE, DEFAULT_RESULT_SET_CONCURRENCY);
    }
}
```

图12　未修改的ConnectionImpl类代码片段示意

```
public class ConnectionImpl extends ConnectionProperties Impl implements MySQLConnection {
        //省略其他业务代码
        public PreparedStatement preparedStatement(string sql) throws SQLException {
        //生成Object数组对象，存储方法参数值
        Object[]parameters=new Object[]{sql};
        //生成try/catch
        try {
          //调用应用自保护方法进入时检测逻辑
          RASPHookResult enterResult=RASPHookproxy.onMethodEnter(parameters,…);
          String HandlerType=enterResultlt.getRaspHookHandlerType().toString();
          if(RASPHookproxy.REPLACE_OR_BLOCK.toString().equals(HandlerType)) {
          //如应用自保护检测结果需阻断或替换程序执行逻辑，return返回结果中设置的返回值
            return(PreparedStatement) enterResultlt.getReturn Value();
        } else if (RASPHook HandlerType.THROW.toString().equals(HandlerType)) {
          //如应用自保护检测结果需往外抛出异常，throw返回结果中设置的异常对象
            throw(Throwable)enterResult.getException();
        }
        //执行程序原逻辑，创建PreparedStatement对象
        PreparedStatement methodReturn=PreparedStatement(sql,1003,1007);
            return methodReturn;
        }catch (Throwable t) {
      }
    }
}
```

图13　经过应用自保护Agent修改后的代码片段示意

通过应用自保护Agent增强后的ConnectionImpl类执行任何SQL语句都会被应用自保护Agent捕获，并检测其合法性，从而实现了彻底的SQL注入攻击防御。应用自保护Agent除了会在开发语言底层重要的API（如文件读写、命令执行、SQL注入等API）中设置防御点（API Hook方式）以外，还能关联应用上下文及

请求的行为，对正常请求进行放行或不触发Hook直接正常通过，对恶意请求（包含但不限于OWASP TOP10涉及的攻击类型、后门攻击、反序列化攻击、内存马攻击等）进行清洗，同时会对有风险的请求进行记录并输出相关特定加密日志，以便安全监控平台进行分析、溯源，快速定位攻击信息及相关漏洞点。

3 软件供应链安全治理方案价值

本文基于DevSecOps理念，构建了覆盖软件应用设计、编码、测试、运营全过程的防御方案，能够精准定位漏洞风险点，检测、防御和监控分析粒度深入代码级，不受业务加密与链路加密的影响。该方案主要有以下4个方面的价值。

a）主动防御：进行安全左移，研发阶段便进行安全介入，运营阶段植入原生安全能力，进行贴身防护，不受业务加密与链路加密的影响，与边界防护联动，实现主动式安全防御。

b）测试高效：覆盖加密应用场景，在软件的功能测试阶段同步开展安全测试，提升安全测试的覆盖率及准确性，动态发现应用软件组件漏洞，减少安全人员投入。

c）深度识别：不依赖于对攻击请求单一的规则判断，防御和监控分析粒度深入代码执行逻辑及触及的周边对象。

d）精准拦截：可对攻击进行精准捕获，输出建议处置方案，有效防护应用程序或其底层平台上的0Day漏洞，防止未知威胁爆发。

4 结束语

为夯实网络与信息安全技术，提高软件应用系统和网络环境的安全水平，抵御开源和云原生时代所带来的新型风险，本文基于DevSecOps理念，运用安全编码、开源组件安全分析、智能动态污点跟踪、应用运行时自保护等新技术，构建覆盖应用软件设计、编码、测试、运营全生命周期的安全防护体系，精准定位漏洞风险点，检测、防御和监控分析粒度深入代码级，且不依赖于单一规则，不受业务加密与链路加密的影响，能有效降低漏洞检测误报率，弥补网络云化后“边界安全防护”技术体系的不足，降低业务应用被攻陷的风险，推进安全能力原生化、服务化。

未来DevSecOps应用发展将呈现出自动化、智能化的趋势，DevSecOps的目标是在不影响效率的前提下提升软件系统安全性。随着人工智能在多个领域的应用，人工替代效能逐渐突出，这与DevSecOps在软件供应链安全治理领域的目标高度一致，人工智能在未来将是DevSecOps必不可缺少的基础能力。

参考文献：

[1] RED HAT. The state of container and kubernetes security[R/OL]. [2022-04-25]. https://www.redhat.com/en/blog/state-kubernetes-security.

[2] DoD. DoD enterprise DevSecOps reference design (v1.0)[R/OL]. [2022-04-25]. https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf?ver=2019-09-26-115824-583.

[3] Veracode. 2020年软件安全现状报告(v.11)[R/OL]. [2022-04-25]. https://www.163.com/dy/article/FQ1D556N0511ALHJ.html.

[4] 中国信息通信研究院，华为技术有限公司，深圳市腾讯计算机系统有限公司，等. 研发运营安全白皮书[R/OL]. [2022-04-25]. http://www.caict.ac.cn/kxyj/qwfb/bps/202007/P020200730529570390226.pdf.

[5] 中国信息通信研究院，云计算开源产业联盟，高效运维社区，等. 研发运营一体化(DevOps)能力成熟度模型[S/OL]. [2022-04-25]. https://blog.csdn.net/nikeylee/article/details/107495362.

[6] FreeBuf咨询. 2020 DevSecOps企业实践白皮书[R/OL]. [2022-04-25]. https://max.book118.com/html/2021/0816/8036014132003134.shtm.

[7] 悬镜安全，FreeBuf咨询. 2020 DevSecOps行业洞察报告[R/OL]. [2022-04-25]. https://max.book118.com/html/2021/1029/8100000123004026.shtm.

[8] 刘长建. DevSecOps的一些关于企业安全的思考[J]. 计算机与网络，2017，43(19)：54-55.

[9] 金泽锋，张佑文，叶文华，等. 面向完整价值交付的文档DevOps应用研究[J]. 软件学报，2019，30(10)：3127-3147.

[10] 王红蕾. 基于DevOps的轻量级持续交付方案[J]. 计算机系统应用，2020，29(9)：87-94.

[11] 车昕. 网络安全新思路 从DevOps到DevSecOps[J]. 通信世界，2019(25)：45-48.

[12] 王渭清，陈军，薄明霞. 电信运营商应用软件安全开发生命周期管理[J]. 电信科学，2011，27(S1)：284-287.

作者简介：
张小梅，工程师，硕士，主要从事网络安全技术研究工作；苏俐竹，工程师，硕士，主要从事网络与信息安全研究工作。