通过实现高性能计算安全增强研究完整性

cloud security alliance®

高性能计算工作组的永久官方网址：
https://cloudsecurityalliance.org/research/working-groups/high-performance-computing-cloud-security

联盟简介

云安全联盟(Cloud Security Alliance,CSA)是中立、权威的全球性非营利产业组织,于2009年正式成立,致力于定义和提高业界对云计算和下一代数字技术安全最佳实践的认识,推动数字安全产业全面发展。

云安全联盟大中华区(Cloud Security Alliance Greater China Region, CSA GCR)作为CSA全球四大区之一,2016年在香港独立注册,于2021年在中国登记注册,是网络安全领域首家在中国境内注册备案的国际NGO,旨在立足中国,连接全球,推动大中华区数字安全技术标准与产业的发展及国际合作。

我们的工作

联盟会刊下载地址
了解联盟更多信息

加入我们

CSA大中华区官网
(https://c-csa.cn)

点击会员

加入联盟

填写相关申请信息

成为CSA会员

JOIN US

致谢

报告中文版支持单位

北京江南天安科技有限公司专注于商用密码产品研发、创新和技术服务，是国家级高新技术企业、国家级专精特新“小巨人”企业。公司根植于密码技术研究和应用创新的深厚积淀之上，集密码产品和解决方案研发、生产、销售和服务于一体，是一家致力于为用户提供全面、可靠的密码产品和安全服务的“密码体系服务商”。公司在密码产品创新方面取得了显著成就，如参与国内首个商业银行国密改造项目，发布国内首款云服务器密码机、国内首块云服务密码卡、国内首台国密专线密码机、国内首台三级服务器密码机，以及支持国密协议的开源 SSL 开发套件。为国家的数字经济和用户的数据安全保驾护航。

江南天安是 CSA 大中华区理事单位，支持该报告内容的翻译，但不影响 CSA 研究内容的开发权和编辑权。

英文版编写专家：

主要作者：Christopher Frenz 和 Yuvaraj Madhewaran

作者：Abhishek Julka、Christopher Frenz、Ivan Casacuberta、John Soares、Michael Roza、Parth Jamodkar、Victor Holanda Rusu、Vijay Velusamy、Yuvaraj Madheswaran

审定者：Richard Prescott Stearns Jr.、Meghana Parwate、Kenneth Moras、Vaibhav Malik、Rakesh Datta、Himanshu Sharma、Harie Srinivasa Bangalore Ram Thilak、Akesh Damaraju

CSA 全球人员：Hillary Baron、Claire Lehnert、Stephen Smith

目录

介绍

从定义上说，高性能计算（High-Performance Computing，HPC）系统是指把计算资源聚合在一起，使其性能超过任何单个工作站、服务器或计算机；这种系统如今已成为研究人员不可或缺的工具，涵盖了从科学探索到工程设计创新的广泛领域。这些复杂计算平台提供的计算力量可令传统计算架构解决不了的复杂问题迎刃而解。然而，市场对性能需求的不断增加，给 HPC 系统带来了一大严峻挑战：究竟应该怎样在速度与安全之间权衡取舍，取得适当平衡呢？有关当今十大顶级超级计算机系统的列表，请参见本文附录。

高性能计算（HPC）领域在传统上视安全为次要考虑因素，甚至认为安全是实现峰值性能的障碍。防火墙、入侵检测系统、数据加密等安全措施被执行时，的确有可能造成延迟并降低系统的总体吞吐量。从这个角度考虑的权衡导致许多 HPC 机构在速度和安全之间选择前者优先，从而使这些系统面对网络攻击时表现得十分脆弱。

然而在 2022 年的超级计算大会上，安全终于成为 HPC 专家关注的焦点。对更快速系统的追求造就了一大漏洞，原因就是这些机器上往往保存着可能会被恶意行为者利用的敏感数据。[1]

解决这一权衡问题的关键在于 HPC 供应商、研究人员和安全专家之间的携手合作和共同努力。新的硬件和软件技术在不断涌现，它们可以在不影响性能的情况下增强安全性。例如，基于硬件的安全性能可以把敏感数据与工作负载隔离开来，另外还有专门的软件可以用来为高性能环境优化安全协议。

随着高性能计算（HPC）的持续发展，安全问题已经不容忽视。在速度与安全之间找到平衡，对于保护这些强大机器及其宝贵数据至关重要。把安全问题置于优先地位并投资开发创新性解决方案，使 HPC 机构得以通过这种方式保护这些系统免受网络攻击侵扰，确保它们能够继续发挥推动科学进步和保障国家安全的作用。

复杂的基础设施、远程访问的广泛使用和敏感数据的存储给 HPC 系统带来了

[1] Dark Reading, Security Is a Second-Class Citizen in High-Performance Computing, December 23, 2022, https://www.darkreading.com/dr-tech/security-is-a-second-class-citizen-in-high-performance-computing。

多重安全挑战。正是这些挑战使它们成为网络攻击的主要目标，导致研究成果丧失、数据损毁、研究进程中断以及潜在的法律后果。机构应该以前瞻性安全措施来抑制这些风险，其中包括风险评价、漏洞管理、补丁管理、访问控制、监测和事件响应。机构可以通过采用这样的策略来保护 HPC 系统和确保其研究结果始终完整如一。

目的

本文的目的是帮助参与使用、管理和保护 HPC 系统的各种利益相关人在确保 HPC 系统安全的问题上达成共识并建立共同的目标。本指南旨在证明，HPC 环境的安全性是能够以促进（而非阻碍）HPC 研究人员取得预期科研成果的方式实现的。

受众

本指南适用于参与使用、管理和保护 HPC 系统的任何人员，其中包括但不限于：

- 网络安全专业人员；
- HPC 系统管理员；
- HPC 应用程序的开发人员；
- 使用高性能计算系统的研究人员和数据科学家；
- 管理高性能计算资源的云管理员；
- 网络管理员；
- 存储管理员。

概述

在科学计算领域，开发者需要为自己的应用程序把信息和网络安全方面的问题考虑周全，开发代码时若能时刻绷紧应用安全这根弦，将不仅可以产生更安全的应用程序，还能促进科研工作取得更佳成果。

所谓应用安全，是指为防范会被未经授权访问、篡改等威胁恶意利用的安全漏洞而给应用程序开发、添加和测试安全性能的过程。2更广义地说，应用安全需要贯穿软件开发生命周期（SDLC）的所有方面，从提出要求和设计阶段直到产品发布后阶段，概莫能外，唯有如此，才能最大限度减少安全漏洞。这不绝不只是必须得到安全专业人员关注的问题，研究人员和科学软件开发者也应该高度重视如何消除这些漏洞。根据国家标准和技术研究所（NIST）的定义，安全漏洞是指在软件代码中存在可能会被攻击者恶意利用的缺陷、毛病或弱点。3因此可以说，我们提高软件的安全性，实际上就是在努力消除代码基底中的那些不仅会为恶意利用行为创造条件，而且还会影响应用程序产生的结果之质量的缺陷和毛病。据估计，每千行代码存在 15-50 个这样的缺陷，4这使通过强化 HPC 安全措施改进代码质量，进而提升 HPC 所支持的科研的准确性具有了巨大潜力。

HPC 面临的安全挑战

- 以下列表虽然并不详尽，但列出了一些常见的 HPC 安全挑战：
- 复杂的基础设施：HPC 系统往往由数千个相互连接的节点组成，从而增加了保护和管理 HPC 系统的难度。这种复杂性给识别和修补漏洞、监测可疑活动以及为整个系统部署安全更新增加了许多困难。
- 独有的软件和信息库：HPC 系统往往使用着一些在 HPC 环境之外不常使用的软件库，例如消息传递接口（MPI）、OpenMP、科学建模软件等。这些软件不仅为 HPC 环境带来独有的安全挑战和受攻击面，而且对于这些挑战，现有的安全工具可能并不具有现成的解决方案。例如，漏洞扫描器可能没有在其数据库中收入相关的软件漏洞，而端点检测与响应（EDR）工具可能没有为这

[2] https://www.vmware.com/topics/glossary/content/application-security.html。
[3] https://csrc.nist.gov/glossary/term/software_vulnerability。
[4] https://www.theregister.com/2018/12/11/software_bugs_that_ate_the_world/。

些软件库建立识别漏洞被人恶意利用的规则。

- 供应链问题：科研社区开发并共享了许多将会被拿到 HPC 系统上运行的应用程序，或者将被会整合进 HPC 应用程序的信息库，从而增加了供应链风险和攻击的可能性。
- 远程访问：研究人员和科学家会经常远程访问 HPC 系统，从而可能引入安全风险。未经授权者可能会通过入侵远程访问渠道接触敏感数据、安装恶意软件或干扰研究活动。
- 敏感数据：HPC 系统往往存储着敏感数据，例如科研成果和知识产权。这些敏感数据容易招致网络攻击，以达到攻击者窃取、破坏或删除数据的目的。这些数据如果丢失或被人篡改，可能会给科研社区及数据拥有机构带来严重后果。
- 高级威胁：HPC 系统正逐渐成为吸引可能涉及拒绝服务（DoS）等复杂技术的高级网络攻击的目标。运行 HPC 系统的机构必须对这些新兴威胁有充分有认识并主动采取防御措施。
- 上述 HPC 安全挑战会造成以下后果：
- 研究结果失窃：安全措施不力可能会遭致未经授权者访问并窃取敏感的研究数据，例如科学发现、算法和模拟结果。这种情况有可能给科研社区造成毁灭性影响，因为它有可能导致知识产权丧失、研究工作重复进行以及研究成果无法发布或分享。
- 研究结果遭破坏：恶意软件有可能感染高性能计算（HPC）系统，从而破坏或删除有价值的研究数据。这有可能造成花费数月乃至数年心血的研究成果损失殆尽，研究人员不得不重复实验或模拟。
- 研究工作流程被干扰：拒绝服务（DoS）攻击可能会使 HPC 系统陷于瘫痪，阻止研究人员访问开展工作所需要的资源。这将延误研究项目的持续进行，给研究机构造成经济损失并损害机构的声誉。
- 研究完整性：相关法规要求研究机构保持研究数据的完整性。如果违反这些法规，可能会让人怀疑研究结果的有效性并损害机构的声誉。
- 数据泄露：HPC 系统可能存储着受《通用数据保护条例》（GDPR）、《加州消费者隐私法案》（CCPA）等数据隐私法规保护的敏感数据。违反这些法规可能

会导致机构受罚和承担法律责任。

HPC 的架构

为了使本指南有尽可能广泛的适用范围，我们将围绕着 NIST 的高性能计算（HPC）参考架构描述文中讨论的控制和其他建议。以下是对 NIST 参考架构中各个区域的简要描述：

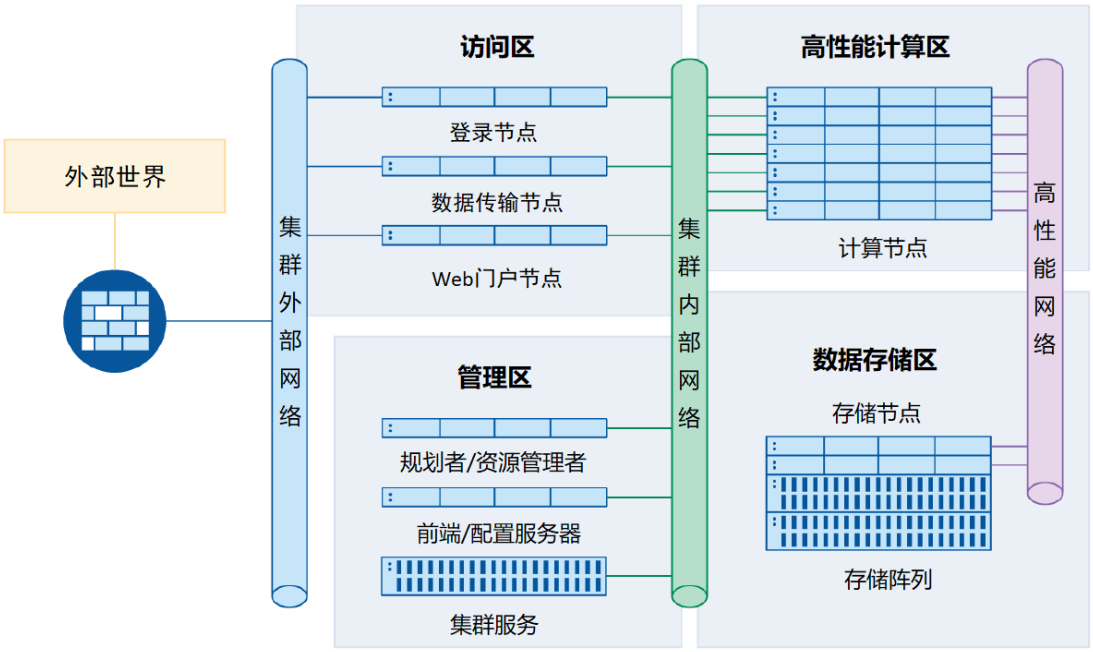

图 1 HPC 参考架构

高性能计算区：这个区域包含通过高速网络互连的计算节点，通常使用 GPU 等硬件加速器。这个区域的软件堆栈的安装和配置是被集中管理的。

数据存储区：这个区域安装着存储系统，其中包括并行文件系统（PFS）、节点本地存储器和归档文件系统。它存储着 HPC 应用程序的数据。

访问区：用户和管理员通过登录节点、数据传输节点和 Web 门户访问 HPC 系统。这些节点提供数据传输和作业提交等各种服务。

管理区：这个区域负责管理和维护 HPC 基础设施。这里包含用于配置管理、网络管理和服务管理等各种功能的服务器和交换机。

了解 HPC 架构有助于为执行以下章节列举的诸多安全控制打下基础，而合理布局的架构是实现 HPC 安全的关键。例如，许多 HPC 系统就是因为在安全外壳（SSH）

密钥管理和外部访问 HPC 系统的方式上存在架构性缺陷而被加密币挖矿组织攻陷的。5

更强的安全保护带来更优质的科研成果

我们将用以下小节来探讨一些应用安全控制推动科研进步的例子。

输入验证

输入验证是一种编程技术，可确保只能给应用程序输入正确类型和格式的数据。输入验证是一项关键安全控制，可用于防止恶意输入进入应用程序。6举例来说，输入验证是增强抵御某些类型攻击——例如跨站脚本（XSS）或 SQL 注入（SQLi）——的保护措施的一种潜在方法。输入验证例程可用来限制应用程序只接受哪些类型数据，从而使攻击者更难提供内含可执行内容的输入，比如在 XSS 攻击中输入“<script>alert('XSS')</script>”，或者在 SQLi 攻击中输入“'1' = '1'”。在典型的商业化应用程序中，输入验证码被用来确保用户只给电话号码字段输入电话号码，或者只给电子邮件字段输入电子邮件地址，而在科研环境中，输入验证同样具有重要价值。

下面，让我们以几个用于检查输入是否为有效 DNA 序列的输入验证伪代码作为例子。

伪代码举例：

```
#read in DNA Sequence to $Seq as input
#Check to see if the input is actually just DNA sequence characters before
processing it
if($Seq in ['A', 'T', 'C',  'G']) {
  #run some operation
}
else{
  #output error message
}
```

这个伪例程的设计是为了读取 DNA 序列并检查该序列是否只由与 4 个典型 DNA 碱基对应的字母 A、T、C、G 组成。该伪例程可以防止 RNA 或蛋白质序列被错误输入，从而帮助验证，应用程序只使用适当数据。

[5] https://www.securityweek.com/crypto-mining-campaign-hits-european-supercomputers/。

[6] https://owasp.org/www-project-top-ten/。

同样，对数字数据也可以进行类似的验证。例如，pH 值被要求作为输入项，那么最好确保不要把 25 用作可接受值，因为 pH 值的范围仅为 0 到 14。在某些情况下，这种控制可能会显得更为重要，例如当软件定义的模型只对某些输入范围有效时。5000 kg 可能是一个完全有效的质量，但是如果软件模型在设计上不能处理超过 100 g 的质量，则 5000 kg 就不会是有效输入。重要的是我们必须认识到，输入验证控制不仅应该用于手动输入的字段，还应该用于被作为输入读取的文件或被从应用程序编程接口（API）或其他来源获取的数据。

“输入的是垃圾，输出的必然也是垃圾”一直是计算机科学的一个信条，而输入验证是确保减少错误输出的主要手段。输入验证通过消除输入错误可能导致的错误或无效结果，提高了科学应用程序生成结果的质量。应用程序的科学完整性和抵御攻击的能力都会因为输入验证而得到改善。

错误处理

错误处理是高性能云计算领域的一个关键组成部分，会对系统的可用性、性能和数据完整性产生直接影响。让我们一种经典错误处理情况为例——除以零。无论是出于缺少输入验证还是其他什么不可预见情况的原因，我们都很难想象在一系列冗长计算中会出现分母为零的情况。任何数除以零都会导致一个未定义的值，这可能会在应用程序中引发问题，例如应用程序崩溃，或者更糟糕的是，计算进程继续使用这个未定义的值，导致后续计算产生错误的输出。

伪代码举例：

```
i=5
j=0
try {
   K=i/j
}
catch {
   Output divide by zero error and safely exit application
}
```

错误处理为应对这种情况提供了一种更好的手段——通过错误处理，应用程序可以适当退出执行并提醒用户进行纠正，而不是让用户在可能不知情的情况下接受基于错误计算的结果。因此，错误处理是确保 HPC 应用程序内进行的计算的完整性，以及这些计算支持的研究的完整性的关键。

高性能云应用程序在错误处理方面面临着独特的挑战，这主要是由云环境的

分布式特性、不断增加的复杂性以及对实时响应的需要造成的。机构可以通过为云原生 HPC 应用程序执行强大的定制化错误处理，最大限度实现系统的可用性、保持数据的完整性，以及保证任务关键的计算和研究工作的可靠性。传统的错误处理方法可能无法有效应对这些挑战，因此需要为云原生应用程序专门量身定制策略。

用于高性能云计算的错误处理技术

有效的错误处理对于保持高性能云计算环境的可靠性和性能至关重要。执行强大的错误处理技术不仅可以保证系统弹性，还能增强安全性、确保合规和优化用户体验。以下是将先进的错误处理技术集成到高性能云计算系统中的几点关键策略：

- 执行把多项服务相互隔离的微服务架构，使每个组件都能独立处理错误和容错。
- 用先进的监测和日志记录工具主动识别错误、追踪其来源和促进快速调试。为错误日志和监测工具执行强访问控制，确保只有得到授权的人员能够访问与错误相关的信息，以防止未经授权的访问或操纵。
- 用电路断路器检测和处理故障，防止级联故障，实现系统在高负载或有故障发生的情况下的平稳降级。
- 引入智能重试机制，自动从瞬态错误中恢复，增强系统弹性。
- 进行受控实验，模拟系统故障并评估系统做出的响应，识别错误处理中的薄弱环节，增强系统的稳健性。
- 在测试过程中有意给系统注入故障，以评估系统在不同场景下的错误处理能力。
- 为应用程序设计遇到错误时平稳降级的能力，确保即便在降级状态下，关键服务仍然可用。
- 确保错误日志不包含个人可识别信息（PII），通过数据匿名化或伪匿名化落实《通用数据保护条例》（GDPR）的规定。
- 制定并执行强有力的数据泄露响应计划，其中包括处理涉及个人数据的错误的具体措施，确保达到《通用数据保护条例》（GDPR）的报告要求。

- 参考 NIST SP 800-64，让安全编码实践规范贯穿应用程序的整个开发生命周期。对代码进行审查，着重关注错误处理机制，强调输入验证和安全错误消息，以防漏洞被人恶意利用。
- 为错误处理活动全面保持审计踪迹，以保证透明度和落实《通用数据保护条例》（GDPR）的可问责性规定。
- 制定相关计划，定期对参与错误处理的人员进行安全培训和意识培养，确保员工能够熟练识别和响应安全事件。

严格依照《通用数据保护条例》（GDPR）和 NIST 指南的要求开展错误处理工作，可令机构从一种全面和系统化的安全方法中获益：既能保证高性能云计算的安全，同时还能给用户带来更好的体验并提高研究完整性。

编码和转义

注入攻击始终是软件系统面临的一种持续威胁。这些攻击往往利用用户输入中存在的漏洞，操纵经过攻击者解释的代码，从而达到破坏系统完整性的目的。本节将深入探讨 SQL 注入的具体情况，强调由其带来的风险，介绍通过转义和编码实现的防御措施。所谓编码，是指把特殊字符转换成某种不同但等效的形式，而这种形式在目标解释器中将不再危险，例如，在以 HTML 格式显示的数据中用“&gt”取代“>”。转义则是指在值之前添加一个特殊字符，以避免产生误解，例如，在引证字符之前添加一个反斜杠“\”，以便将其解释为文本，而非一个字符串值的结束。为了让读者更好地了解这些控制的工作原理，我们列举了以下场景，其中一个 Web 应用程序通过嵌入在应用程序中的简单 SQL 查询收集用户凭证以进行身份验证[7]：

伪代码举例：

```
string query = "SELECT * FROM users WHERE username = '" +
request.getParameter("userName") +
"';AND password= '" + request.getParameter("password") + "'";
```

对于像“some_user@email.com”和“R@ndomPwd”这样的合法用户输入，SQL 查询直截了当：

[7] https://owasp.org/www-project-top-ten/2017/A1_2017-Injection。

SQL：

```
SELECT * FROM users WHERE username = 'some_user@email.com' AND password =
'R@ndomPwd';
```

然而，攻击者可以通过注入诸如“admin'--”之类的恶意输入来利用这一点，绕过出具口令的要求。这时，SQL 查询可能会变成：

SQL：

```
SELECT * FROM users WHERE username = 'admin'--' AND password = 'password';
```

SQL 中的“--”表示一条注释，使查询的其余部分变得无效。

SQL：

```
SELECT * FROM users WHERE username = 'admin';
```

转义原本可以防止这种攻击，因为在 SQL 语句执行之前，如果将单引号转义，使其只被视为普通文本而非特殊字符，就可以防止它们被解释为一条注释的开始。

下面是另外一个复杂一些的攻击例子：

伪代码：

```
string query = "SELECT * FROM users WHERE username = '" +
request.getParameter("userName") +
"'AND password= '" + request.getParameter("password") +
"' AND state = 'ACTIVE' AND ip =" + request.getParameter("ip");
```

攻击者可以通过将口令输入为“' OR 1=1/*”来操纵这一点：

SQL：

```
SELECT * FROM users WHERE username = 'admin@admin'
AND password = '' OR 1=1 /*
AND state = 'ACTIVE'
AND ip = x.x.x.x;
```

这个巧妙的注入成功了，因为口令条件始终为真，而查询的其余部分都被注释掉了。通过转义引号，使其在执行 SQL 语句之前不再作为特殊字符，将再次防止攻击的成功执行。

防御策略：输入转义和编码

为了帮助抑制此类攻击，人们通常会按下文所述方式使用转义和编码。[89]

输入转义：

- 准备语句：
  - 使用准备语句或参数化查询，把用户输入用作参数，可防止恶意代码注入。
- 存储规程：
  - 执行存储规程，在数据库内封装和验证输入，可以降低未经授权操纵的风险。

建议读者通过确保适当的类型转换、长度限制以及对特殊字符的检查，把上述策略与前文所述严格的输入验证结合使用。

输出编码：

- HTML 实体编码：
  - 将像“<”、“>”、“″”、“’”和“&”这样的字符转换成其 HTML 实体，可防止它们在 HTML 中被解释成代码。
- JavaScript 编码：
  - 在动态生成 JavaScript 时对用户输入进行编码，可防止在脚本元素内发生注入攻击。
- URL 编码：
  - 对用于 URL 的用户输入实施 URL 编码，可确保它们被正确解释而不产生歧义。

对于 HPC 环境来说，执行强健的输入转义和编码正变得越来越重要。这些技

[8] https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data。
[9] https://www.nature.com/articles/d41586-021-02211-4。

术增强了系统对抗注入攻击的能力，为保持关键数据的完整性和安全性提供了一个关键保护层。尽管网络威胁环境在不断演变，但主动处理输入的方式依然是构建富有弹性的安全软件系统的基石。在科学研究方面，这些控制也有助于促进产生更高质量的科研成果，因为它们可以帮助消除数据完整性问题。转义和编码可以确保文本字符串被正确解释，不会赋予它们以超出预期的特殊含义，而这有助于减少发生问题的潜在可能性，例如基因数据有时会由于字符解释错误而被误认为是日期。

更新机制

作为庞大科学设备的 HPC 系统与射电望远镜和粒子加速器类似，需要投入巨额资金来构建和维护才能保持它们的运行和有效性。对这些系统的维护与维护其他大型 IT 基础设施一样，需要定期更新，以纳入错误修复、执行安全补丁和集成新功能，从而确保实现最佳性能和安全性。在 HPC 环境中，故障停机可能会造成巨大财务损失，金额往往高达数十万美元乃至更多。即便是短暂的系统中断也会干扰正在进行的研究、计算进程或操作工作流程，导致显著的生产力损失和错失机会。因此，更新旨在尽量减少故障停机时间，确保关键计算资源可被最大限度访问。

HPC 系统更新的主要目的是不断增强系统能力，特别是在加快和实现大规模数值模拟（即我们常说的“数值运算”）上。这些更新涵盖了系统的各个方面，其中包括优化计算算法、增强并行处理技术以及集成新硬件技术（例如加速器或协处理器）。HPC 更新的另外一个关键目的是保持系统的完整性、可用性和可靠性，使其成为生成可重复科研结果的一致工具。可重复性是科学方法的基础，确保研究结果可被独立检验和验证。

HPC 系统更新的频率和策略所基于的是与其他 IT 系统类似的原则。更新往往与企业的具体需要密切相关，由获得对于保持竞争力或应对新挑战至为关键的新性能的要求驱动。无论是提升计算能力还是启用更先进的算法，是否进行更新的决定都应该建立在对这些改进究竟会在多大程度上符合企业战略目标和运行需求进行全面评估的基础上。因此，安全方面的因素也不容忽视。有效的 HPC 系统更新管理离不开健全的威胁建模和风险管理实践。

这其中包括了解企业的风险偏好以及高层领导对待风险的态度。识别潜在威胁和漏洞可令 HPC 站点得以根据更新的潜在影响以及它们被恶意利用的可能性排列各项更新的先后顺序，确保最关键的安全隐患优先得到有效解决。当因安全威胁的出现而需要进行更新时，企业对 HPC 系统内执行的安全控制的信心会对企业将以什么方式进行更新产生影响。HPC 站点在确定更新的紧迫性和范围之前，必须对现行安全措施抑制潜在风险的有效性做出评价。对安全控制的高度信任可能会允许企业采取比较平稳的更新策略，而对漏洞的担忧则可能促使企业采取更激进的打补丁策略。

HPC 系统对于企业的重要性在确定更新的频率和性质方面起着关键作用。举例来说，如果系统负责处理来自科研项目或运行流程中重要仪器的连续数据流，那么为了保持数据的完整性、可靠性和整体系统性能，可能需要迅速部署更新。而另一方面，非关键的 HPC 系统可能能够容忍更长时间的突发停机。

为了避免停机并减轻因更新带来的不利影响，HPC 系统的更新应该采取兼顾多方面需要的策略。首先，企业在执行更新之前应该对更新作全面升级测试，通常应该分阶段或在测试环境中进行。在这些受控环境中，应该用严格的回归测试工具（例如 ReFrame）来审查系统变更的兼容性和稳定性，以及这些变更对受支持科研工作流程的影响。其次，HPC 系统可以借助诸如 Spack 和 EasyBuild 之类的软件包管理工具来确保更新可在用户环境中重复部署。这些工具可帮助系统化安装和管理软件包，保证不同计算节点和用户会话之间的一致性。此外，HPC 设施还要依靠自己的工程团队和与供应商的合作来迅速解决测试阶段遇到的任何问题。

用户在保持通过 HPC 资源取得的科研成果的可重复性和完整性方面发挥着关键作用。因此，管理员与用户之间的合作对于预防问题发生和保持科研成果的完整性至关重要。用户应该遵循强有力的数据和软件管理实践规范，其中包括版本控制、来源跟踪、数据验证、软件物料清单（SBOM）生成，以及利用不可变的软件安装方式，例如容器和 SquashFS 镜像。此外，机构还应该鼓励用户采用与 HPC 中心相同的工具来测试和部署自己的软件。这些实践规范可以确保计算工作流程的可追溯性、可靠性和透明性，把更新过程引入错误或差异的风险降至最低，同时还可以巩固用户与 HPC 中心之间的关系。

为了降低更新过程中的网络传输成本，特别是在涉及数千节点的云环境中，一种常用的方法是向节点静态提供引导映像。这些引导映像通常经过预配置，只需要在启动时进行最低程度的节点自定义设置，例如设置主机名等。这种方法在映像创建过程中就开始执行安全控制，可为在部署更新之前识别系统存在的潜在漏洞和配置错误带来很大方便。另外，这种方法还能生成将会成为识别和处理未来漏洞的宝贵资源的工件，例如软件物料清单（SBOM）。

除此之外，把映像以只读方式挂载还可以抑制与系统篡改相关的潜在问题，从而额外提供了一个安全层。限制对映像的写访问可以大幅度降低未经授权修改或篡改的风险，有助于保持已部署系统的完整性和安全性。这种只读配置可以增强基础设施的整体弹性，保护其免受潜在安全威胁侵扰，同时还能确保更新过程的平稳可靠。

虽然分阶段和在测试环境中测试更新可以最大限度降低 HPC 环境中存在的与更新相关的风险，但是认识到它们的局限性也很重要。这些测试环境通常由几百个节点组成，与动辄由数千节点构成的生产性 HPC 系统相比，规模明显要小得多。因此，尽管更新在这些受控环境中接受了严格的测试，但是在把它们转移到更大规模的生产环境中时，仍然存在与生俱来的风险。这里的主要挑战之一是，我们无法在测试系统中完全复制生产环境的复杂性和细微差别。现实世界使用场景中具有代表性的大规模工作流程可能会表现出难以被复制到测试环境中的行为和依赖关系。因此，尽管更新分阶段接受了全面测试，但是它们在生产环境中的表现，尤其是在大规模工作流程中的表现，始终存在一定程度的不确定性。此外，测试的有效性取决于测试集覆盖范围的全面性。尽管我们会在测试过程中尽可能广泛地覆盖用例和场景，但实现完全覆盖实际上是不可能的。因此，存在与未经测试的边缘案例或系统组件之间未被预见到的交互相关的剩余风险在所难免，而这些风险可能只会在生产环境中表现出来。另外，由于资源和时间有限，测试不可能无限期进行。即便拥有尖端测试框架和自动化工具，模拟和验证系统变更的能力也是有限的。因此，在测试的深度和持续时间与及时部署更新以满足运行要求之间，总是需要做出权衡。

按计划停机维护影响生产的严重程度由系统是进行更新还是升级，以及接受维护的具体组件决定。关键基础设施组件，例如网卡（NIC）和电缆，可能会对

连接形成干扰，而对工作负载管理器等中心服务的更新可能会影响作业调度和资源分配。操作系统的更新从打小补丁到重大升级不等，每种情况对系统功能和与用户应用程序的兼容性的影响程度各不相同。而要求重新编译用户应用程序的更新则又增加了另外一层复杂性，极可能延长停机时间。因此，应该采用什么更新策略，主要取决于所涉资产是否参与数值模拟的提交或执行。在集群的外部网络边界上组部署虚拟化网关节点或堡垒（通常由 3 至 7 个节点组成）是一种常见的做法。这些节点是抵御蛮力攻击的重要保护措施，并可作为跳转主机进入 HPC 登录节点。为这些堡垒选择操作系统时，通常要针对其所提供的具体服务而量身定制——无论它们只是充当跳转主机，还是提供诸如用户主文件夹之类的最小服务，都是如此。由于它们发挥着 HPC 网络接口的关键作用，并且有着与 HPC 系统的不同的用途——它们不参与科学模拟或数值预测——因此相较于基础设施中的其他资产，它们更新的频率会更加频繁。一般来说，对它们应该采取金丝雀更新策略，最初只更新一个节点，如果所有测试全部通过，才会更新其余节点。更新的触发规程会因 HPC 站点所用设备的不同而各异，但也可以简单得像下面列举的这个 cron 作业一样。

```
0 2 * * * root /path/to/is_the_lead_node_sane && /usr/bin/dnf update -y
--security && /usr/bin/dnf needs-restarting -r || /usr/sbin/reboot
```

在这个特定场景中，更新规程计划在凌晨 2:00 启动，只有当包管理器（本例中为 DNF）确定节点处于可重启状态时才会激活节点重启。脚本“is_the_lead_node_sane”扮演着关键角色，被用来评价主节点是否处于可以开始更新进程的稳定状态。任何偏离正常状态的情况都表明更新失败，工程团队会立即收到通知并着手干预。包管理器的选项，例如“--security”，由更新的频率决定。一些设施会选择每日进行安全更新，同时每隔 X 天进行一次全面系统更新。这种方法允许在固定且错开的时间间隔内系统化更新剩余节点——同时、分批或单个进行均可，条件是“is_the_lead_node_sane”脚本在有效运行。

自动化测试机制可以通过外部触发器激活，也可以在系统启动后由“systemd”模块执行。后者提供了对外部服务的自主性和独立性，可以通过具有适当权限的“systemd”执行命令，可能还会用 SELinux 来增强安全性。然而，如果

“is_the_lead_node_sane”脚本出现故障，则需要系统工程师手动介入，通过终端命令启动更新，或者等待下一个更新周期。另外，外部触发方法有其自身的优势，可根据所采用的外部触发解决方案，只需简单点击鼠标即可手动进行更新。

数据传输节点采用的更新机制与堡垒节点的更新机制基本相同，但有一个关键区别：需要检查节点是否介入了任何正在进行的传输作业。因此，传输节点作业在后记中额外集成了一层验证，用于防止关键数据传输操作受到干扰，同时确保安全更新能够顺利进行。决定进行更新的条件是节点的正常运行时间，必须保证只有当节点处于闲置状态时才启动更新。在这种框架下，可以安排每天在没有传输作业运行的时候更新传输节点，也可以按传输作业策略规定的时段进行更新。因此，制定有限制的传输作业策略至关重要，而不可选择无持续时间限制的作业执行方式，因为持续不断地执行作业可能会妨碍系统及时更新。

另外一种方法是时刻检查更新，并且无论当前是否有传输作业在进行，都执行安全更新。在这种情况下，需要由用户负责在更新完成后重新启动传输作业。这一策略优先考虑了系统安全，同时又承认把传输操作的连续性交给用户管理的必要性。系统完整性与运行连续性之间取得平衡后，HPC 环境下的数据传输节点即可达到安全标准，也可保证运行效率。

Web 门户通常要借助云编排技术来进行部署，由这些技术为管理和更新基于 Web 的应用程序提供灵活和可扩展的解决方案。门户采用的具体更新机制会因负责其运行的工程团队偏好的策略而各异。一种更新 Web 门户的方法是利用持续集成/持续部署（CI/CD）管道。在这种模型中，更新通过一系列自动化步骤自动集成、测试并部署到生产环境中。一些工程团队可能会选择采用手动更新规程，特别是对具有复杂架构或敏感数据要求的 Web 门户。手动更新往往需要更为谨慎，即应该先在分阶段环境中进行全面测试之后，再把更新应用到生产环境中。

此外，云编排技术通常还会提供内置的滚动更新功能，使更新得以逐步施用到 Web 门户，只对用户有极小影响。滚动更新即为一次更新应用实例的一部分，确保门户在整个更新过程中始终保持可访问和功能正常状态。这种方法有助于把停机和服务中断时间缩减至最短，使更新过程无缝进行，同时保持良好的用户体验。

从我们在架构图中列出的资产列表可以看出，HPC 系统中的其余资产主要用

于数值模拟的提交和/或执行。所以，更新过程必须以一种能够让用户适应其工作流程有可能受干扰的方式进行。有鉴于此，更新一般每年进行几次，寻求在系统维护与计算能力不被中断之间达到某种平衡。一些 HPC 中心每年进行两次更新，在这个过程中需要全面停机，以确保所有组件的更新都得到全面执行和测试。

管理 HPC 服务的另外一种策略是利用云编排器，这种做法可以为传统的系统部署和更新方法带来范式转变。云编排器的采用可以大幅度减少对通过专用分阶段系统进行操作系统或服务更新的依赖。这种变革性方法使多项服务升级并行接受测试成为可能，缩短了关键组件的测试周期。在这个场景下，云编排器能够用一个专被分配来用于测试目的的节点子集部署一个小型集群。这种创新性测试环境为在受控环境中评估更新，进而在把更新部署到生产环境之前对更新进行充分验证提供了方便。当然，这种测试方法并非不存在缺点——认识到这一点非常重要。部署小型集群进行测试会不可避免地带来权衡取舍问题，因为这样做需要与测试或生产系统争夺资源。资源的这种重新分配意味着要从这些系统中抽取一部分容量来部署每个单独的测试，从而会影响整个系统的性能和容量利用率。尽管存在这些缺点，但是云编排器带来的并行测试和更新周期缩短好处往往会超过相关的资源分配挑战。

不过，通过云编排器为计算节点和集群服务提供 HPC 服务的做法会引入额外的复杂性和安全考虑因素。这种架构扩大了保护和更新 HPC 系统和服务的传统任务，提供集群的云编排器的整个基础设施都被涵盖其中。范围被这样扩大后，HPC 中心不仅必须满足其核心 HPC 资源的安全保护和维护需要，还必须满足云编排器环境的需要。云编排器的集成要求全面掌握传统 HPC 系统和云技术，这对系统管理员提出了独有的挑战。他们必须在两个各自有一套工具、协议和最佳安全实践的不同技术栈之间穿行并实施管理。这种双重性给 HPC 中心实施的安全教育培训和意识培养计划施加了额外的压力——它们必须培养出能够有效管理和保护这两种环境的管理员。

信息库验证

科学计算领域的开发人员常常会开发使用多个信息库的应用程序。标准化信息库让科学界的开发者享受了许多好处，因为使用验证过准确性的信息库可以帮

助确保科研结果的有效性和可重复性——与白手起家建库相比，使用现成的信息库通常会大大降低引入错误的可能性。

信息库中的内容是可供重复使用的代码片段，可以使应用程序的开发变得更快、更容易。这些代码可由构建应用程序的同一团队开发（第一方库），可由与应用程序开发团队有合作关系的另一团队开发（第二方库），也可由提供信息库或资源的任何其他方开发（第三方库）。无论是哪种情况，我们都必须清楚，一个库可能会同时使用其他库，而这些库可以属于前面提到的任何类型。由此不难看出，即便研究项目的复杂性极低，都可能存在一条由多个库组成的长链，而这些库在许多情况下并不是主要开发者自己构建的。

信息库具有通过以下两种主要方式在已完成开发的应用程序中引发安全问题的潜在可能性。首先，信息库可能会给应用程序的代码库引入漏洞，在存在依赖树的情况下，开发者以及托管和支持应用程序的团队可能对这些漏洞并不知情，甚至根本就没有想到，这也是软件物料清单（SBOM）日益变得关键的原因之一。其次，针对软件库的供应链攻击越来越猖獗，威胁者会给库插入恶意软件或其他恶意内容，以达到破坏目标环境的目的。针对 Python 包仓库 PyPI 的攻击便是一个例子10。虽然把恶意软件挡在机构大门之外至为关键，但是保持结果的完整性，确保所使用的任何信息库均不曾被人篡改也同样重要。信息库被人篡改后，可能无法再达到预期的准确性或产生预期的结果。

以下几种最佳安全实践可以帮助减少脆弱的信息库面临的风险：

开发人员在构建和测试应用程序时，应该只使用来自已知可信来源的信息库。开发人员应该避免使用过时或文档不完善的信息库。此外，对信息库的依赖项应该逐一进行测试。每个信息库都会有多个依赖项，例如其他第三方库、外部 API、用户输入等。当前有多种工具可以帮助对信息库进行此类测试。开发人员应该通过集成测试来保证应用程序以符合预期的方式运行。最后，开发人员还应进行静态应用程序安全测试（SAST）和动态应用程序安全测试（DAST），以找出信息库中可能存在的漏洞。如果所用信息库得自可信来源，而且是最新版本，这些安全测试取得理想结果应该不成问题。

---

10 https://news.sophos.com/en-us/2021/03/07/poison-packages-supply-chain-risks-user-hits-python-community-with-4000-fakemodules/。

应用程序投入运行后，开发人员应该定期检查所用信息库是否有了新的漏洞。检查可以使用上面提到的技术，也可以通过执行版本管理系统来进行。如果正在使用的信息库变得越来越脆弱，则应该有一个流程来确保尽快进行更新，以防攻击者利用新的漏洞篡改科研结果。尽管上述技术可以帮助机构检查他们考虑使用的信息库的安全问题，但是机构可能还需要考虑他们预计要分发的信息库面临的安全挑战，因此，机构需要考虑采取一种方法，使信息库的用户可以验证库的完整性。校验和或安全代码签名等技术可用来帮助验证代码的完整性和真实性。

内存安全控制和 OpenMP

在 HPC 中，内存对于执行复杂科学和工程应用至关重要。内存是计算机处理器处理数据的临时存储空间，直接影响着 HPC 处理大型数据集和执行计算密集型任务的能力。

HPC 内存安全是指通过技术手段来防止 HPC 应用程序出现内存访问错误。内存访问错误是一种常见问题，有可能导致程序崩溃、数据损坏和安全漏洞。以下是 HPC 应用程序中常见的一些内存访问错误。

- 数据竞争：当有多个线程尝试同时访问和修改同一内存位置时，就是发生了数据竞争。这有可能导致数据损坏，因为不同的线程可能会彼此覆盖更改。
- 内存泄漏：内存被分配后不再释放出来，就是发生了内存泄漏。这最终有可能导致程序耗尽内存并崩溃。
- 堆栈溢出：当线程尝试访问超出其堆栈帧的内存时，就是发生了堆栈溢出。这有可能导致程序崩溃或产生安全漏洞。
- 双重释放内存：当内存被分配、释放，然后又再次释放时，就是发生了双重释放。这有可能导致内存损坏和安全漏洞。
- 释放后使用：当先前已释放的内存被直接或间接使用时，就是发生了释放后使用。这有可能导致数据损坏和安全漏洞。

内存安全控制技术可通过执行访问规则、检测违规行为和提供恢复机制来帮助防止出现这些错误。常用的 HPC 应用程序内存安全控制技术包括：

- 内存屏障：这些指令确保所有线程在完成一个内存操作之后才开始下一

个内存操作。

- 原子操作：这些操作保证能够以原子方式执行，即便操作被其他线程中断也是如此。
- 内存映射文件：这些文件被映射到进程的虚拟内存中，可帮助改善内存访问性能和安全性。
- 内存调试器：这些工具可以识别和调试内存访问错误。
- 指针标记:这项技术涉及为指针分配标签,以标明指针的类型和拥有权。根据标签执行访问规则可以帮助防止内存错误。
- 保护页：这项技术涉及给被分配的内存区域周围添加额外的内存页。这些保护页可在被访问时触发异常，以此来检测内存访问错误。
- 内存回滚：这项技术可将内存状态恢复到发生错误之前的某个时间。

HPC 开发人员和用户还可以采取其他措施来解决内存安全问题：

- 使用具有内存安全性能的语言和信息库：一些编程语言，例如 C++17、Rust 等，内置有内存安全性能，可帮助预防常见内存访问错误。
- 采用静态分析工具和模糊测试：静态分析工具可以在开发过程中识别代码中的潜在内存安全漏洞，而模糊测试可以生成随机测试用例来触发与内存相关的漏洞。
- 遵循安全编码实践规范：避免缓冲区溢出、恰当释放内存和正确初始化所有变量。
- 执行严格的测试和验证规程：这其中包括在各种情况下用自动化测试工具测试应用程序，以验证内存安全性。
- 使用内存管理库：这些库可以提供比传统编程语言构造更强和更高效的内存管理方式。像 libunwind 和 Valgrind 这样的管理库可帮助跟踪内存分配和内存释放、检测内存泄漏和识别与内存相关的其他问题。
- 对开发人员和用户开展有关内存安全的教育：开发人员和用户需要了解内存安全问题的潜在风险以及遵循安全编程实践规范的重要性。这一点可以通过培训、发放相关文件和开展意识培养活动来实现。
- 持续监测和改进内存使用：持续监测 HPC 应用程序的内存使用情况，以识别和解决潜在问题非常重要。这一点可以通过使用性能计数器和监测

面板等工具实现。

HPC 开发人员和用户可以通过采取这些措施并把其他因素（例如内存膨胀和地址空间布局随机化[ASLR]）考虑周全来创建更可靠、更安全、更高效和更有效的应用程序。

内存膨胀是指内存被分配后不再释放出来，导致未被使用的内存始终占用资源。这可能会影响性能并增加发生内存相关错误的可能性。HPC 开发人员应该严格执行内存管理策略，确保内存既被适当分配也被适当释放，从而解决这个问题。

地址空间布局随机化（ASLR）是另外一项可帮助提高内存安全性的技术。ASLR 随机化虚拟内存空间中内存页的位置，增加了攻击者预测和利用内存漏洞的难度。这一技术可以有效抑制基于内存的攻击和增强 HPC 系统的整体安全性。

HPC 开发人员还可以利用加速器；这是一种专为处理特定计算而设计的硬件设备，比 CPU 更高效。把任务卸载到这些专用设备上可以最大限度减少主内存系统的负载。而对主内存使用的减少可以显著降低发生内存相关错误的风险，例如内存泄漏和悬空指针。

值得一提的是，内存管理的改进还可以提高代码的科学质量。例如，竞态条件不仅可能导致安全问题，还可能导致由于线程或进程访问或写入内存中存储的值的顺序不同而产生不一致的输出。如果内存被以错误的顺序访问，这些问题往往会导致不正确的输出，从而造成结果与算法原本应该产生的预期结果不符。

消息传递接口（MPI）

消息传递用于协调构成 HPC 系统的诸多节点，它允许多个进程在 HPC 系统的不同节点上独立（拥有各自的内存和执行环境）但并发地运行。这些进程可以通过使用消息传递接口（MPI）交换数据和相互通信。如果架构设计不当，通过消息传递进行通信的节点会极易受竞态条件影响，还可能会通过传递中的消息给进程注入错误或恶意内容。

通过 MPI 确保 HPC 通信的安全是保证数据的完整性和保密性的关键。以下是一些常用于这一目的技术手段和实践规范：

- 加密：在进程之间传输的数据应该加密，以防数据被未经授权访问。这在处理敏感数据时尤为重要。

- 身份验证：在允许进程加入 MPI 通信组之前，首先要对进程进行身份验证。这样做可以阻止未经授权的进程参与计算并访问数据。
- 完整性检查：可以用校验和或其他完整性检查来确保数据在传输过程中不曾被人篡改。
- 安全的 MPI 执行方案：一些 MPI 执行方案内置有安全性能。例如，MPICH2 的 MPI 执行方案支持安全套接层（SSL）和传输层安全（TLS）协议，由这些协议提供安全的加密通信。
- 网络安全：从更广泛的层面上说，用于 MPI 通信的网络基础设施也应该得到安全保护。这其中包括通过防火墙控制流量、把 MPI 通信网络与其他网络隔离，以及监测网络活动以发现入侵迹象。

我们以这样一个场景为例，其中有一个 HPC 应用程序正在处理敏感数据。该应用程序通过 MPI 在多个节点上进行并行计算。为了确保 MPI 通信的安全，该应用程序使用了支持 SSL/TLS 的 MPI 执行方案。在数据被从一个进程发送到另一个进程之前，应用程序用 SSL/TLS 给数据加密。接收进程在接收数据时进行解密。这确保了即便数据在传输过程中被截获，也无法被未经授权的第三方读取。

此外，提高消息传递的安全性还可以让研究受益。例如，消除竞态条件的潜在可能性有助于确保进程按正确顺序执行，从而保证了研究结果的准确性。

请注意，这些技术手段尽管可以大幅度增强 MPI 通信的安全性，但是它们同时还会带来额外的开销并影响 HPC 应用程序的性能。因此，在设计和执行 HPC 应用程序时，必须在安全性和性能之间找到平衡。

零信任

零信任的原则是“永不信任，总要验证”。在零信任架构中，任何实体，无论来自内部还是外部，都不被默认可信。每个用户、设备和应用程序都被视为不可信，不论它们处于什么位置，在访问资源之前都必须接受身份验证和认证。在 HPC 环境中采用零信任安全模型，可为解决与研究完整性相关的安全问题提供一种主动和全面的方法。这种主动和持续的验证可以最大限度缩小受攻击面、降低网络内部的横向移动风险和增强整体安全态势。

零信任高性能云计算的关键策略：

- 遵循 NIST 零信任架构（ZTA）框架，强调持续验证、严格的访问控制和最低权限原则的重要性。
- 实施微分段，把网络划分成相互隔离的小段，以限制横向移动并控制潜在威胁。
- 为用户身份验证执行多因素认证（MFA），以在口令之外增加一层安全保护，确保只有得到授权的人员可以访问。
- 利用强大的身份和访问管理（IAM）解决方案来管理和控制用户的访问、权限和角色，确保落实最低权限原则。
- 执行持续监测和实时威胁检测，以快速识别和响应可疑活动或异常情况。
- 给传输中的和静止状态的数据加密，以保护敏感信息不被未经授权访问。
- 通过相关机制征得用户对数据处理活动的明确同意，为个人数据提供透明性和控制。
- 促进数据主体权利的行使，例如访问、更正和删除个人数据的权利，同时确保这些过程的安全性。
- 通过适用的身份验证和授权机制来保护 API，确保只有得到授权的应用程序能够访问 API 和与 API 交互。
- 在可行的情况下通过先进的端点保护机制来加强端点安全（例如通过专门用于访问 HPC 系统的工作站或跳转箱），因为一些端点安全工具可能会影响性能。
- 加固系统，定期更新和打补丁，以抑制漏洞风险。
- 采用网络安全控制，例如防火墙、入侵检测/预防系统和安全网页网关，以过滤和监测流量。
- 整合 NIST SP 800-53 第 5 修订版规定的安全和隐私控制[11]，确保以一种基于风险的全面方法管理云安全。
- 采用 NIST SP 800-53 规定的安全 DevOps 控制，让安全保护贯穿系统的整个开发生命周期，确保从一开始就把安全问题考虑周全。
- 遵循 NIST SP 800-37 第 2 修订版阐明的 NIST 风险管理框架（RMF）[12]，在高性能云应用程序的部署和运行工作中落实风险管理原则。

[11] https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final。
[12] https://csrc.nist.gov/pubs/sp/800/37/r2/final。

机构可以通过遵循这些《通用数据保护条例》（GDPR）和 NIST 指南，为高性能云计算构建一个强大的零信任（ZT）安全基础，把身份管理、持续监测、安全配置、数据保护和事件响应等关键方面全部涵盖其中。

零信任还可以帮助提高 HPC 系统和应用程序的科学完整性，因为它确保只有得到授权的操作被允许执行。零信任通过这种做法限制了错误操作对输出结果的质量产生负面影响的潜在可能性。机构可以借助零信任建立一个富有弹性的安全基础，不仅保护敏感数据，还保持科研社区的可信性和声誉。

HPC 的网络安全

在 HPC 中，网络分段是一项关键的网络安全策略，涉及把企业网络划分成离散的子网或网段。这种方法可以把关键组件单独隔离，强化了安全性和对敏感数据的保护。

网络分段在 HPC 中的一个重要应用是把 HPC 系统的各个组件隔离到不同的安全区域内，例如把访问区与其他区域（例如数据存储区和计算区）隔离。这种分段可以阻止未经授权者访问关键组件，从而降低潜在安全漏洞的影响。网络分段还可以充当一种安全控制，帮助提升 HPC 系统的性能，因为适当的网络分段可以减少网络噪声，进而可以改善网络吞吐量和延迟。

把一个网络划分成 NIST 建议的高性能计算区、数据存储区、访问区和管理区，并视每个区域为一个独立的安全区域，可以带来许多安全优势，具体如下所述。

安全区域是网络和信息安全的一个基本概念。它们被用来根据安全要求、可信级别和数据敏感性对网络或计算环境的不同区域进行分类和隔离。安全区域是网络分段和访问控制的关键组成部分。

安全区域在网络内定义了不同的网段，每个网段都设置有特定的安全控制、访问策略和信任边界。这些区域的建立旨在实现这样几个目标：

- 隔离：安全区域把特定区域内的资源和数据与其他区域隔离开来，限制潜在威胁横向移动。
- 访问控制：安全区域执行访问控制，规定了哪些人员或哪些设备被允许与特定区域内的资源通信。

- 数据保护：通过把敏感或关键数据隔离在指定区域内来帮助保护它们。
- 风险降低：安全区域可最大限度缩小受攻击面和限制安全漏洞，从而降低整体风险。
- 区域之间的通信应该仅限于在得到批准的通信信道内进行，而这些信道具有增加控制的潜力。

通信信道即为数据和信息在各安全区域之间流动的路径或管道。这些信道包括物理网络、虚拟连接或机构架构内的逻辑路径。设立通信信道的目的是在保持安全和信任边界的同时，促进数据和服务在安全区域之间的受控交换。

通信信道受基于其所连接的区域的特定安全要求和可信级别的安全控制和策略辖制。通信信道与安全区域之间的交互涉及：

- 访问控制策略：每条通信信道都受其所连接的安全区域规定的访问控制策略辖制。这些策略决定了哪些人员或哪些设备被允许通过该信道访问两端的资源。
- 网络分段：通信信道用于执行网络分段，确保把不同的安全区域隔离开来。这一点对于控制数据的流动和减少安全漏洞的影响至关重要。
- 安全措施：通信信道必须严格执行其所连接的安全区域定义的安全措施和控制。例如，通过通信信道传输的敏感数据可能被要求必须实施强加密。
- 监测和审计：通信信道内的活动接受监测和审计，以查出任何未经授权访问或数据泄露。安全信息和事件管理（SIEM）系统常被用来达到这一目的。

对不同的安全区域还可以按不同的可信级别进行划分，以帮助确定安全需要。例如，访问区的可信级别应该有别于管理区。

每个安全区域都会被分配可信级别，以反映要求该区域必须达到的可信度和安全水平。可信级别有助于确定可在一个区域内托管的数据和服务类型，以及所需要的访问控制和安全措施级别。

与安全区域的交互涉及：

- 数据敏感性：可信级别影响安全区域内的数据分类。更高的可信级别往往与更敏感的数据对应，因此要求采取更严格的安全措施。

- 访问控制：可信级别决定了一个区域必须执行的访问控制策略。可信级别更高的区域可以对受权用户执行比较宽松的访问策略。
- 安全措施：可信级别决定了一个区域需要采取的安全措施的级别。可信级别更高的区域通常必须执行更强的加密、身份验证和入侵检测系统。
- 通信边界：可信级别有助于建立通信边界，决定了哪些安全区域之间可以相互通信。可信级别更高的区域与其他高可信级别区域之间的通信边界可以比较宽松。

安全飞地

前面讲的许多策略还常被用来通过创建安全飞地来提高安全性和改善研究的完整性。13这些安全飞地是指配备了基于硬件的加密和隔离技术的私密内存区域，被突出为一种解决方案。安全飞地可以保护各种人工智能（AI）和机器学习（ML）资产，其中包括：

- 原始数据：用于机器学习（ML）算法的敏感数据可以在使用、传输和存储过程中得到保护，从而降低暴露风险和确保数据隐私。
- 专有训练引擎：安全飞地保护用于训练机器学习模型的算法和技术，即便这些算法在不可信硬件上运行。
- 推理/专家引擎：基于实时数据的决策引擎受到保护，确保企业的专业知识和核心价值安全无恙。
- 数据结论：在安全飞地内生成的数据被默认是安全的，对暴露的风险可以通过执行策略来控制。

使用安全飞地不仅可以降低数据和知识产权风险，还可以为从更广泛的数据集构建强大能力提供机会。安全飞地通常与安全区域和数据分类理念结合使用。

日志记录

日志记录在保护 HPC 应用程序的许多方面发挥着关键作用。

- 故障排除：日志记录可帮助识别和诊断系统、应用程序或网络中存在的问题。系统管理员通过查看日志，可以发现表明存在潜在问题的模式或

[13] https://info.ornl.gov/sites/publications/files/Pub71949.pdf。

错误。这使及时排除故障和解决问题成为可能。

- 性能监测：日志记录允许通过跟踪响应时间、资源使用情况和数据吞吐量等指标来监测系统性能。分析这些日志有助于优化系统和提高整体性能。
- 安全增强：日志记录可帮助检测和响应安全威胁。日志可以通过跟踪失败的登录尝试、未经授权访问和其他可疑行为提供有价值的见解。这些信息有助于识别和抑制安全漏洞，保护敏感数据和保持系统完整性。
- 合规要求：《医疗保险流通与责任法案》（HIPAA）、《支付卡行业数据安全标准》（PCI DSS）、《网络安全成熟度模型认证》（CMMC）等监管标准往往要求实施强有力的日志记录和审计实践。对日志记录数据的访问、修改和其他系统变更可为司法调查提供审计踪迹。合规是保持可信度和遵守行业法规的关键。
- 历史记录：日志创建了系统活动和变更的历史记录。这些记录为未来参考、趋势分析和决策提供了宝贵的资源。了解过去的事件可帮助就改进和更新系统做出决策。
- 根原因分析：当有事件发生时，详细的日志使人得以对根原因进行深入分析。日志可以通过追溯导致问题发生的事件，帮助确定漏洞、错误配置或恶意活动。
- 取证与调查：安全事件发生后，日志将充当关键证据。它们可以协助开展事后调查，帮助安全团队掌握攻击向量、受影响的系统以及事件的影响。

尽管日志记录是安全生态系统的一个重要组成部分，但它同时也为这个环境引入了某些风险。

- 性能影响：日志记录可能会显著影响系统性能。写入日志涉及输入/输出操作（例如硬盘写入），这会消耗 CPU 周期、内存、存储带宽、硬盘空间和缓冲内存等资源。过度的日志记录可能会降低应用程序的运行速度，对应用程序的响应能力和吞吐量造成影响。
- 缓冲区溢出：当程序写入超出被分配的缓冲区的边界时，就是发生了缓冲区溢出。如果日志库没有进行适当的边界检查，有可能导致内存损坏

和安全漏洞。

- 悬挂指针：悬挂指针是指指向已被释放的内存或无效内存的指针。如果日志库处理指针不当，可能会导致未被定义的行为和安全问题。
- 格式字符串漏洞：如果日志库允许使用不受控制的格式字符串（例如用 printf 风格格式化），可能会导致任意代码执行或信息泄露。
- 注入攻击：日志库如果未适当清理输入，面对注入攻击（例如 SQL 注入或命令注入）时会变得十分脆弱。
- 竞态条件：在多线程环境中，日志库如果未能正确处理并发访问，可能会出现竞态条件。这可能会导致非预期行为或安全漏洞。
- 内存泄漏：日志库如果未能适当释放内存，可能会导致资源耗尽和潜在的拒绝服务（DoS）攻击。
- 敏感数据泄露：日志中记录的敏感信息（例如口令、令牌）可能会暴露给未经授权人员。
- 身份验证/授权缺失：日志库应该执行适当的访问控制，以防未经授权用户篡改日志。
- 不安全的文件权限：如果日志文件允许任何人访问，敏感数据可能会暴露。
- 不充分的错误处理：日志库的错误处理不当有可能导致非预期行为或系统崩溃。
- 依赖漏洞：日志库往往需要依赖其他组件（例如第三方库）。由这些依赖性产生的漏洞可能会影响整体安全性。例如，Boost C++日志库14曾被发现存在一些安全风险。

漏洞管理

HPC 系统的漏洞管理对于提升科学成果至关重要。

我们以一个用 HPC 系统进行气候建模的研究机构为例。这些模型需要使用大量数据和计算资源，得出的结果对于了解和应对气候变化具有重大影响。

- 数据完整性：有效的漏洞管理可以确保这些模型所用数据的完整性。如

[14] https://www.cvedetails.com/vulnerability-list/vendor_id-7685/Boost.html。

果有漏洞被人利用，数据可能会被篡改，导致生成不准确的模型结果。机构可以通过主动管理漏洞，获得对其数据完整性和模型准确性的信心。

- 系统可用性：漏洞还可以被利用来破坏系统可用性。例如，DoS 攻击可以使 HPC 系统陷于瘫痪。而这将延迟模型的处理，减缓研究的步伐。漏洞管理可帮助防止出现这种系统中断，确保研究人员能够在需要的时候访问他们需要的资源。
- 保密性：一些研究可能涉及必须保密的敏感数据。漏洞管理可以帮助保护这些数据免遭未经授权访问。
- 声誉：有效的漏洞管理可以提升机构的声誉。机构可以通过表明自己对网络安全的承诺来赢得合作伙伴、资助者和公众的信任。

HPC 系统漏洞管理直接支持机构的科研使命。它确保研究人员能够有效、安全、自信地开展工作，从而产生更可靠和具有影响力的科研成果。

以下是可协助做到这一点的几个流程。

- 资产发现与清单管理：IT 专业人员可以用资产清单管理系统来跟踪和维护公司数字环境内所有设备、软件、服务器等的记录。
- 漏洞扫描：漏洞扫描器可以对系统和网络进行一系列测试，以查找常见弱点或缺陷。例如，像 npm audit for node 和 maven dependencies-check 这样的包管理工具可用来检测库的依赖关系中存在的漏洞。
- 补丁管理：补丁管理软件可帮助确保计算机系统打上最新安全补丁。大多数补丁管理解决方案会自动检查更新，并在有新补丁发布时提示用户。
- 配置管理：安全配置管理（SCM）软件可帮助确保以安全的方式配置系统，它们能够跟踪和批准对设备安全设置的更改，同时保证系统安全策略合规。
- 指标测量：漏洞管理程序会对某些指标进行测量，以评价它们的有效性。这些指标可能包括扫描覆盖率、扫描频率、关键漏洞数量、已关闭漏洞数量以及排除项。

例如，美国陆军作战能力发展指挥部分析中心借助 HPC 对国防部的可存活性、脆弱性和致命性建模进行更快速和更复杂的分析。15

15 https://www.army.mil/article/252336/army_vulnerability_lethality_analysis_bolstered_through_high_performance_computi

需要注意的是，漏洞管理必须不间断持续进行，才能始终适应新出现的威胁和不断变化的环境。

机构在为 HPC 系统制定漏洞管理计划时要充分认识到，市场上有售的许多漏洞扫描器可能无法检测出运行在 HPC 系统上的许多专业软件应用程序和信息库的过时版本，因此，使用商业化漏洞扫描器可能只能揭示系统中实际存在的部分漏洞。机构可能还需要考虑采取诸如资产清单和版本跟踪之类的策略，以此来补充商业化漏洞扫描器的功效。

尽管打补丁修补漏洞确实有助于提高研究结果的完整性，因为经过更多次修补的软件版本往往意味着其代码库错误更少，但是我们必须清楚，并非所有旧库和旧版软件都是可以修补或更换的。科学研究的可重复性需要往往决定了旧版软件必须保留，以防将来出现重复一组关键计算的需要。虽然打补丁是一项关键的安全控制，但是 HPC 系统的漏洞管理还必须考虑采用抑制漏洞的补偿性控制，以应对没有补丁可用的漏洞或需要为科研的可重复性而维持的遗留软件。

结论

利用高性能计算（HPC）取得更好研究成果之旅需要精心的规划、强大的基础设施和对安全的高度关注这三点的紧密结合。我们的这次探索表明，提高 HPC 系统和应用程序的安全性，具有提升各领域（包括金融、医疗和科学研究）研究质量和研究完整性的巨大潜力。

首先，采用 HPC 架构和部署的最佳实践规范至关重要。这涉及精心设计可扩展和高效的系统，以满足现代研究工作流程的计算需求。无论是基于云的解决方案还是本地集群，架构都必须经过优化，以提高性能、可靠性、灵活性和安全性。

数据管理已成为 HPC 方案取得成功的一个关键方面。执行有效的数据存储、访问和共享机制，可确保研究人员能够在严格遵守安全和合规要求的同时，充分发挥数据资产的潜力。此外，先进数据分析技术的采用可使研究人员得以从庞大的数据集中提取有意义的见解，推动创新和发现。

在 HPC 领域，考虑安全因素的重要性不容低估。随着数据泄露和网络威胁持续带来重大风险，采取强有力的安全措施以保护敏感研究数据和知识产权正变得

---

ng。

越来越重要。当提高安全性与改进研究质量和研究完整性之间的联系日渐彰显时，这一点的价值不言自明。通过提高安全性，可以取得更好的科研成果。

总之，拥抱 HPC 以提高研究水平需要有一种全面的方法把技术、数据管理和安全性全部涵盖其中。机构通过这样做，并把持续改进置于优先地位，可以有效抑制安全风险、提高研究完整性以及保持科研社区的可信性和声誉。通过合作、教育和创新，HPC 社区将能在不断变化的威胁形势下自由徜徉，在日益互连和动态的计算环境中保证研究数据的完整性和保密性。

参考文献

NIST Interagency Report (IR) 8476, 3rd High-Performance Computing Security Workshop: Joint NIST-NSF Workshop Report, September 2023, https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8476.pdf

Cloud Security Alliance (CSA) - High-Performance Computing (HPC) Tabletop Exercise Guide, May 2023, https://cloudsecurityalliance.org/artifacts/high-performance-computing-tabletop-guide/

NIST SP 800-223 (Initial Public Draft) - High-Performance Computing (HPC) Security: Architecture, Threat Analysis, and Security Posture, February 2023, https://csrc.nist.gov/pubs/sp/800/223/ipd Dark Reading, Security Is a Second-Class Citizen in High-Performance Computing, December 23, 2022, https://www.darkreading.com/dr-tech/security-is-a-second-class-citizen-in-high-performance-computing

How Zero Trust Privilege Addresses Five High-performance Computing Security Risks, 2023 https://www.somerfordassociates.com/wp-content/uploads/2020/01/how-zero-trust-privilege-addresses-five-high-performance-computing-security-risks.pdf

Security in High-Performance Computing Environments, September 2017 https://cacm.acm.org/magazines/2017/9/220422-security-in-high-performance-computing-environments/fulltext?mobile=false2-security-in-high-performance-computing-environmentsfull-text?mobile=false

OPENMP API Specification: Version 5.0, November 2018, https://www.openmp.org/spec-html/5.0/openmp.html

The Case for Memory Safe Roadmaps, December 2023, https://media.defense.gov/2023/Dec/06/2003352724/-1/-1/0/THE-CASE-FOR-MEMORY-SAFE-ROADMAPS-TLP-CLEAR.PDF

Introduction to Memory Unsafety for VPs of Engineering, August 2019, https://alexgaynor.net/2019/aug/12/introduction-to-memory-unsafety-for-vps-of-engineering/

Prossimo, What is memory safety and why does it matter? As of January 2024, https://www.memorysafety.org/docs/memory-safety/

ZERO TRUST: THE CYBERSECURITY MINDSET ALL ORGANIZATIONS NEED TO ADOPT, MARCH 8, 2022 https://cyber-center.org/zero-trust-the-cybersecurity-mindset-all-organizations-need-to-adopt/

Karakasis, V. et al. (2020). Enabling Continuous Testing of HPC Systems Using ReFrame. In: Juckeland, G., Chandrasekaran, S. (eds) Tools and Techniques for High Performance Computing. HUST SE-HER WIHPC 2019 2019 2019. Communications in Computer and Information Science, vol 1190. Springer, Cham. https://doi.org/10.1007/978-3-030-44728-1_3

Todd Gamblin, Matthew P. LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, & W. Scott Futral (2015). The Spack Package Manager: Bringing order to HPC software chaos. In Supercomputing 2015 (SC'15).

Geimer, M., Hoste, K., & McLay, R. (2014). Modern scientific software management using EasyBuild and Lmod. In *Proceedings of the First International Workshop on HPC User Support Tools* (pp. 41–51). IEEE Press.

OWASP Top Ten Proactive Controls 2018 | C4: Encode and Escape Data | OWASP Foundation. https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data.html

附录 1 十大顶级超级计算机

本列表更新至 2023 年 11 月。

Rmax 和 Rpeak 值以 PFlop/s（每秒千万亿次浮点运算）为单位。有关其他字段的详细信息，请查看 TOP500 的描述。

Rpeak 值基于 CPU 的标称时钟频率计算得出。请考虑 Turbo CPU 时钟频率对系统效率的影响。

排名	系统	处理核心	Rmax (PFlop)	Rpeak (PFlop)	功率 (kW)
1	Frontier-Cray EX235a，惠普制造，配置包括AMD优化的第三代EPYC 64核 2GHz 处理器、AMD Instinct MI250X加速器、Slingshot-11互连架构，部署在美国能源部/橡树岭国家实验室。	8,699,904	1,194.00	1,679.82	22,703
2	Aurora-Cray EX，惠普制造，采用 Intel Exascale Compute Blade，配备 Xeon CPU Max 9470，拥有 52 核 2.4GHz 处理器、Intel Data Center GPU Max 加速器和 Slingshot-11 互连架构，部署在美国能源部/阿贡国家实验室。	4,742,808	585.34	1,059.33	24,687
3	Eagle，微软构建，采用 Xeon Platinum 8480C 48核2GHz处理器，配备 NVIDIA H100 GPU 和 NVIDIA Infiniband NDR互连架构，部署在美国的微软Azure云平台。	1,123,200	561.20	846.84	
4	Supercomputer Fugaku，富士通理化学研究所研发，采用A64FX 48核 2.2GHz处理器和Tofu互连架构，部署在日本的富士通理化学研究所计算科学中心。	7,630,848	442.01	537.21	29,899
5	LUMI-Cray EX235a，惠普构建，采用AMD优化的第三代EPYC 64核2GHz处理器，配备AMD Instinct MI250X加速器和Slingshot-11互连架构，部署在芬兰的HPEEuroHPC/CSC中心。	2,220,288	309.1	428.7	6,016

6	Leonardo-BullSequana XH2000，Atos公司构建，配置包括Xeon Platinum 8358 32核2.6GHz处理器、NVIDIA A100 SXM4 64GB GPU和四路NVIDIA HDR100 Infiniband互连架构，部署在意大利的EuroHPC/CINECA中心。	1,824,768	238.7	304.47	7,404
7	Summit IBM Power System AC922，IBM 构建，配备 IBM POWER9 22 核 3.07GHz 处理器，搭载 NVIDIA Volta GV100 GPU，使用双路 Mellanox EDR Infiniband 互连架构，部署在美国能源部橡树岭国家实验室。	2,414,592	148.6	200.79	10,096
8	MareNostrum 5-ACC-BullSequana XH3000，Alto公司研发，配备Xeon Platinum 8460Y+ 40核2.3GHz处理器，搭载NVIDIA H100 64GB GPU，使用EVIDEN Infiniband NDR200互连架构，部署在西班牙的EuroHPC/BSC中心。	680,960	138.20	265.57	2,560
9	Eos NVIDIA DGX SuperPOD-NVIDIA DGX H100，NVIDIA公司构建，采用Xeon Platinum 8480C 56核3.8GHz处理器，搭载NVIDIA H100 GPU，使用Infiniband NDR400互连架构，由NVIDIA公司部署在美国。	485,888	121.40	188.65	
10	Sierra IBM Power System AC922，IBM、NVIDIA和Mellanox联合开发，配备IBM POWER9 22核3.1GHz处理器，搭载NVIDIA Volta GV100 GPU，使用双路Mellanox EDR Infiniband互连架构，部署在美国能源部/国家核安全局/劳伦斯利弗莫尔国家实验室。	1,572,480	94.64	125.71	7,438

资料来源：https://www.top500.org/lists/top500/2023/11/。