从 Next.js 服务器被黑谈起：PCI DSS 视角下的漏洞管理实战指南

atsec 唐冬生 2026 年 3 月

关键词：PCI DSS、漏洞管理、风险排序、补丁管理、应急响应、安全运维、渗透测试、CVSS、弱点扫描、ASV

摘要 2025 年末，针对 Next.js 框架的 RCE 漏洞（CVE-2025-55182）被大规模利用，受害服务器沦为挖矿节点并被植入 DDoS 木马。事件本身并不复杂，但它暴露的问题具有相当的普遍性：情报获取滞后、资产盘点缺位、修复优先级混乱。本文以此事件为切入点，从 PCI DSS v4.0.1 漏洞管理要求（Req 6.3.1、11.3.1、6.3.3）出发，结合渗透测试与风险管理框架，拆解漏洞“识别与风险排序”和“修复或处置”两个阶段的具体要求，并通过案例说明如何把合规条款转化为实际可执行的防御动作。

目录

1. 引言：从一起挖矿入侵事件说起

2025 年末，一则"**紧急！Next.js 高危漏洞致服务器被黑，我已中招！**"的告警在安全社区流传。从攻击日志还原的攻击链来看：攻击者利用 Next.js 框架的远程代码执行漏洞（CVE-2025-55182 / CVE-2025-66478，CVSS 9.1），通过构造特制 HTTP 请求在服务器上执行任意命令，随后从外部 IP 拉取载荷脚本，赋予 777 权限执行，完成挖矿程序和 DDoS 木马的部署。整个过程利用的是已公开披露的已知漏洞，攻击路径清晰，技术门槛并不高。

一位受害企业的运维负责人事后说：“漏洞公开的时候我们知道，但觉得自己被打的概率不大。”这句话几乎是所有类似事件的注脚。这类安全事件每天都在发生，受害者不同，暴露的问题却高度相似：

- **情报盲区：**对第三方框架的漏洞披露缺乏有效的跟踪机制，没有订阅来源可靠的威胁情报。
- **资产盲区：**不清楚自己的环境里跑着哪个版本的哪些组件，受影响资产的排查靠人工回忆。
- **决策盲区：**漏洞披露之后没有明确的优先级判断流程，修复时限模糊，责任归属不清，最终修复行动一拖再拖。

支付卡行业数据安全标准（PCI DSS）v4.0.1 及其配套文件《Vulnerability Management Infographic》对上述问题均有明确要求。好消息是，这套框架逻辑清晰、具备可操作性；坏消息是，多数企业实际上仅完成了“扫描”环节，后续的风险定级、时限跟踪与修复验证，基本依赖人工和运气。本文将结合具体案例，阐释该框架在操作层面的实际要求。

2. 漏洞从哪里来

漏洞管理的前提是理解漏洞从哪里来。无论表现形式多复杂，安全漏洞本质上都是系统在设计、实现、配置或运维过程中引入的缺陷，给了攻击者可以利用的切入点。按照 NIST NVD 和 CWE（通用弱点枚举）分类信息的分类方式，常见漏洞类型包括以下三类：

- **内存安全漏洞**

内核崩溃、远程执行任意代码、系统被接管，这些后果大多源自同一类漏洞：内存安全问题。这类漏洞集中在用 C/C++编写的底层组件中，包括操作系统内核、设备驱动和网络协议栈，因为这些语言本身没有内存安全保证：

  - **缓冲区溢出（Buffer Overflow）：**向预分配内存空间写入超量数据，覆盖相邻内存区域，可触发远程代码执行（RCE）或导致系统崩溃，是历史上出现频率最高的高危漏洞类型之一。
  - **释放后重用（Use-After-Free）/ 空指针解引用：**访问已释放或未初始化的内存区域，可能导致程序行为异常、数据泄露乃至权限提升。
  - **整数溢出/下溢（Integer Overflow/Underflow）：**运算结果超出变量类型的表示范围，引发逻辑错误，常见于网络数据包解析场景，攻击者可借此绕过边界检查。

- **逻辑与配置漏洞**

即使服务器不存在缓冲区溢出等内存安全问题，在业务逻辑上，例如支付接口的商品金额

校验逻辑仍可能存在漏洞。这类问题的根源与内存无关，而是源于程序设计或配置层面的失误：

- **业务逻辑缺陷（Business Logic Error）**：例如支付流程中的商品价格篡改、权限校验绕过、状态跳转异常等。PCI DSS Req. 6.2.4 要求在安全软件开发生命周期（SDLC）中系统防范此类问题，代码审查时需重点关注。
- **不安全的默认配置（Security Misconfiguration）**：OWASP Top 10 长期将配置类问题列为高频项。默认密码未修改、调试接口暴露、不必要的服务对外开放，都属于这一类。PCI DSS Req. 2 对此有专门的配置基线要求，包括变更所有默认密码、关闭非必要功能。
- **注入类漏洞（Injection Flaws）**：SQL 注入、OS 命令注入、跨站脚本（XSS）等均属此类，是 Req. 6.2.4 和 Req. 6.4（面向公众的 Web 应用保护）的重点关注范围。

- **供应链与第三方组件风险漏洞**

现代应用高度依赖开源库与第三方组件。以 Node.js（一种广泛应用于网站后端开发的 JavaScript 运行环境，Next.js 则是基于 Node.js 构建的流行开发框架）生态为例，一个典型项目往往包含数百甚至数千个间接依赖，其中任一组件的漏洞都可能波及所有上层应用。Next.js 此次漏洞正是此类风险的典型体现，因此框架层面的安全问题，会影响所有部署该框架的应用。

PCI DSS 要求 6.3.2 规定，组织须维护完整的**软件物料清单（SBOM：Software Bill of Materials）**，范围不仅限于自研代码，还须覆盖所有第三方及开源组件，并持续跟踪其安全状态。缺乏这份清单，即使漏洞情报到手，但仍然无法定位受影响资产。

3. PCI DSS v4.0.1 漏洞管理标准框架详解

基于对漏洞形成机理的深入理解——从代码缺陷的产生、第三方组件的引入，到资产清单的缺失——PCI DSS 的漏洞管理要求须同时覆盖开发、运营、情报三个维度。遵循 PCI SSC 官方指引，可将 Req. 6 与 Req. 11 的核心要求整合为漏洞管理的两大核心阶段：一，漏洞识别与风险分级；二，漏洞的修复与处置。两阶段首尾相接、循环往复，构成持续运转的完整的漏洞治理体系。

3.1 阶段一：漏洞识别与风险分级

本阶段的目标是系统性的发现漏洞，并结合组织自身环境评估实际风险。

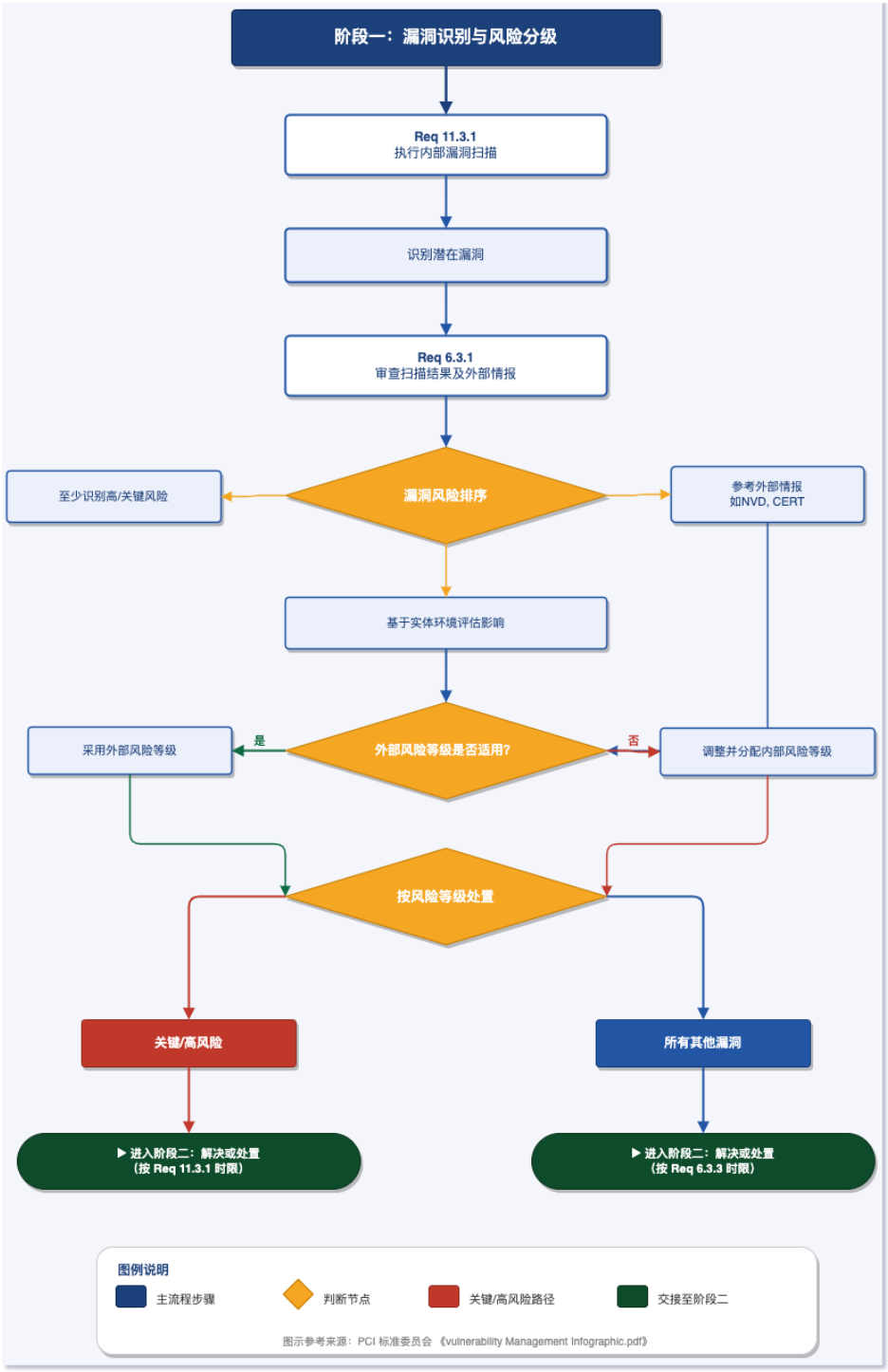

图 1：PCI DSS v4.0.1 漏洞管理核心流程阶段一示意图

- **要求 11.3.1 内部漏洞扫描**

PCI DSS 要求对持卡人数据环境（CDE：Cardholder Data Environment）内的**所有系统组件至少每季度执行一次内部漏洞扫描**。扫描须由具备独立性的合格人员使用持续更新的工具执行，以确保结果可信。

这里有一点需要区分清楚：**漏洞扫描（Vulnerability Scanning）**与**渗透测试（Penetration Testing）**是两类性质不同的活动，不能互相替代。漏洞扫描靠自动化工具匹配已知特征，输出漏洞列表。渗透测试是安全人员模拟攻击者，手动尝试实际利用漏洞，其目

的是验证防御是否真正有效，同时能发现扫描工具找不到的逻辑漏洞。PCI DSS 对两者都有独立要求（参考 PCI DSS 标准 Req. 11.3 和 Req. 11.4）。

- **要求 6.3.1 安全漏洞的识别与风险分级**

根据标准的要求，可以将漏洞进行识别与分级：

1. **漏洞识别——多源情报融合：**组织须建立持续的信息获取机制，从行业认可的来源识别新的安全漏洞，来源包括但不限于国际及国家计算机应急响应团队（CERT）发布的安全警报、行业认可的漏洞数据库（如美国国家漏洞数据库 NVD）、软件厂商安全通告（如 Next.js 官方安全公告）以及内部扫描结果。各渠道信息孤立是响应滞后的主要原因之一，需要统一汇聚。
2. **风险分级——基于影响的评估：**
   - 依据行业最佳实践，结合潜在影响为漏洞分配风险等级。
   - 至少须识别所有被认定为对本组织环境影响为“High”或“Critical”的漏洞。
   - 可参考 CVSS 评分，但须结合组织实际情况进行调整。
3. **自主风险评估（关键差异化要求）：**PCI DSS 明确要求，组织须采用正式、客观、可论证的方法，独立确定漏洞风险等级，而非直接沿用扫描工具或 NVD 的评级。评估时须考虑以下因素：
   - **资产位置：**该组件是否位于持卡人数据环境（CDE）范围内，是否直接处理、存储或传输持卡人数据（CHD）。
   - **暴露面：**是否直接面向互联网，现有防火墙或 WAF 等控制措施是否有缓解效果。
   - 可利用性：是否已有公开的 PoC 利用代码，是否已经出现在野利用（In-the-Wild Exploitation）。
   - **数据敏感性：**受影响系统所处理业务数据的机密性级别和合规性要求。
   - **系统关键性**：是否为安全系统、面向公众的设备和系统、数据库等关键系统。
4. **风险等级的动态调整**：组织应该认识到，风险等级并非一成不变的，单个低风险或中风险漏洞，若存在于同一系统或可被利用访问 CDE，可能构成高风险或严重风险，例如一个 NVD 中评分为 CVSS 5.5（中级别）的漏洞，如果存在于直接处理支付交易的服务器上，且已有在野利用记录，组织完全可以将其内部定级为“Critical”并优先处理。

3.2 阶段二：漏洞修复或处置

仅识别漏洞并无实质意义，关键在于落实修复。本阶段的核心目标是：依据前期风险定级结果，在规定时限内采取有效行动，进而消除或降低安全风险。

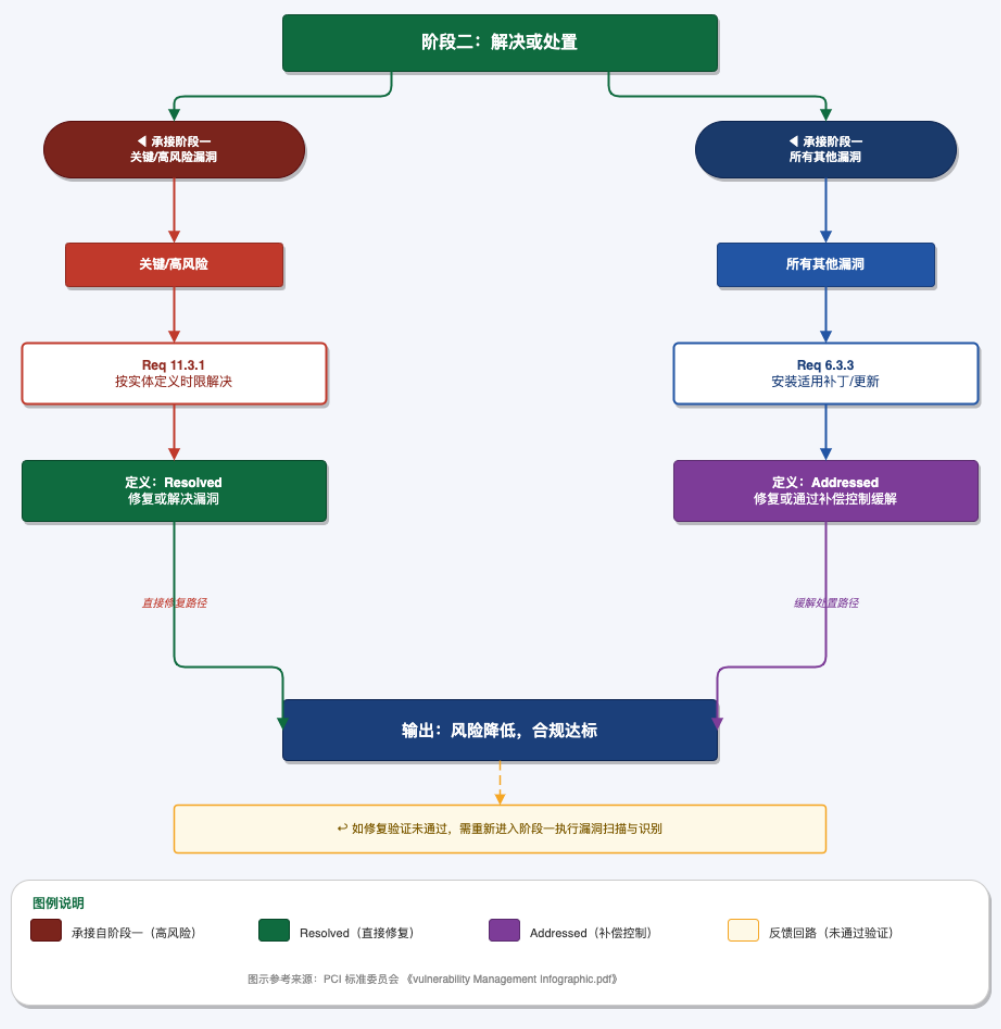

图 2：PCI DSS v4.0.1 漏洞管理核心流程阶段二示意图

概念辨析：“修复”（Resolve）与“处置”（Address）

PCI DSS 在这里区分了两类针对漏洞的处理方式：

术语	含义与适用场景
修复（Resolve）	通过打补丁、版本升级或代码修复彻底消除漏洞本身。有官方补丁且变更风险可控时，这是首选路径。例如将 Next.js 升级至已修复版本（14.2.30+ 或 15.3.5+）。
处置（Address）	无法立即修复时，通过其他手段临时降低风险，作为过渡措施。常见方式包括：在 WAF 上配置虚拟补丁规则、禁用或隔离受影响的服务、网络隔离等。过渡措施需明确记录在案，持续跟踪，直到漏洞被彻底修复。

- **要求 6.3.3 安全补丁/更新的安装时限（关键红线）**

这一要求为漏洞修复或处置划定了明确的时间边界，是 PCI DSS 漏洞管理中不可妥协的量化指标：

- **严重（Critical）漏洞：**由组织按 Req. 6.3.1 流程自主评定为“Critical”级别的漏洞，安全补丁/更新必须在官方补丁发布后一个月内（30 天）内完成安装。
- **其他非严重（Critical）漏洞：**其它非严重的漏洞更新须纳入组织通过目标风险分析（TRA）确立的分级处置框架。时间窗口须与风险评级精准匹配，并以正式文档固化、经管理层审批后执行。

行业针对漏洞修复和处置最佳实践为：高风险 60 天内，中风险 90 天，低风险 180 天。

- **要求 11.3.1 重新验证达到修复闭环**

Req. 11.3.1 标准原文明确要求：“Rescans are performed that confirm all high-risk and all critical vulnerabilities (as noted above) have been resolved.”扫描发现的高风险和严重漏洞，修复完成后必须重新扫描以验证。内部漏洞扫描不仅是发现漏洞，更要确认修复动作是否真正落地。重新扫描须验证漏洞已被修复或降级，这是 Req. 11.3.1 的硬性要求。修复完成不复扫，等于未修复，此环节断不可省。

4. 案例推演：Next.js 漏洞应对全流程——基于 PCI DSS 最佳实践

这一章节，让我们以开头的 Next.js 漏洞事件来完整演示 PCI DSS 要求的漏洞管理框架的实践。

4.1 阶段一：识别与风险分级

步骤 1：情报获取（Req. 6.3.1）

CVE-2025-55182/CVE-2025-66478 公开后，建立了情报订阅的安全团队应在数小时内通过以下渠道收到告警：

- NVD/CISA KEV（已知被利用漏洞目录）自动推送；
- Next.js 官方安全公告（GitHub Security Advisories）；
- 订阅的行业威胁情报平台（如 VirusTotal Intelligence、OpenCTI 等）；
- 内部安全团队的 SIEM/SOAR 系统告警（如果已预先配置检测规则）。

步骤 2：资产排查（Req. 6.3.2 和 Req. 11.3.1）

收到告警后，立即基于 SBOM（软件物料清单）和资产清单排查所有运行受此漏洞影响的 Next.js 实例，标注至资产地图。对于尚未建立 SBOM 的组织，这一步往往是拖慢整个响应进程的主要瓶颈。

步骤 3：内部风险定级（Req. 6.3.1）

安全团队对受影响资产逐一评估业务上下游，示例如下：

资产场景	内部风险定级	定级依据
CDE 内支付接口服务器（面向互联网）	严重	CDE 核心资产，直接暴露于互联网，已

	（Critical）	有公开 PoC，攻击成功可直接导致 CHD 泄露
CDE 内部管理系统（无直接互联网暴露）	高（High）	CDE 范围内，无直接互联网访问，攻击者需通过内网横向移动才可到达
非 CDE 的内部测试服务器	中（Medium）	CDE 范围外，无敏感数据，网络访问受限，仍需修复以防内网横向移动

4.2 阶段二：修复与处置

步骤 4：时限启动（Req. 6.3.3）

CDE 内支付服务器被定为“Critical”级别，修复计时器即刻启动（30 天倒计时自官方补丁发布日起算），紧急变更流程须同步发起，不可等待常规发布窗口。

步骤 5：并行行动（Resolve + Address）

- **首选路径——Resolve：**对 CDE 内的关键服务器，立即启动紧急变更，升级 Next.js 至安全版本（14.2.30+ 或 15.3.5+），经完整测试后在 30 天内完成生产部署。
- **临时缓解——Address：**在升级测试和部署窗口期间，在 WAF 上针对该 CVE 的利用特征配置防护规则，拦截利用该中间件绕过漏洞的特制 POST 请求，以此作为过渡缓解措施，并记录风险。
- **分级处理：**测试环境的“中”风险服务器，通过 TRA 文档确认修复时间框架（如 90 天内），纳入常规变更排期。

步骤 6：验证闭环（Req. 11.3.1）

修复完成后，对所有已修复资产重新扫描，扫描报告作为修复完成的验证证据，确认漏洞状态更新为“已修复”。这份记录同时也是 PCI DSS QSA 审计的关键文档要求。

经过这套流程，从漏洞公开到完成修复的时间窗口被明显压缩，实际暴露时间变得可量化、可管控，也许此次入侵的安全事件就不会发生。

5. 落地实践：将框架转化为防御能力

理解框架仅是起点，要将 PCI DSS 漏洞管理框架从书面落到实处，可从以下四个维度着手：

5.1 先把责任说清楚

首先将流程书面化，清晰界定各方职责：

- 漏洞情报订阅与分发：通常由安全运营团队负责；
- 内部风险定级：由安全架构师或评估团队承担，须独立于运营团队；
- 修复进度跟踪与逾期风险上报：由安全合规/GRC 团队负责；
- 修复与验证责任人：IT 运维/DevOps 负责修复，安全团队负责验证。

5.2 利用工具自动流转，不靠人工传话

避免孤立工具间的人工传递，降低信息丢失与延误风险。整合目标在于实现数据自动流动：

- 漏洞扫描工具（例如：Nessus、Qualys 等）与 ITSM/工单系统集成，扫描结果自动转单、自动分配负责人、自动设置 SLA 时限。
- SBOM 管理工具（例如：OWASP Dependency-Track）接入 CI/CD 流水线，在代码构建阶段即识别已知脆弱性依赖。
- SIEM/SOAR 与威胁情报平台对接，实现漏洞情报自动关联、告警自动聚合。

5.3 CVSS 为基础，但须叠加使用场景

一个 CVSS 9.1 的漏洞，若存在于隔离的测试环境，实际影响有限；若存在于承载真实资金流转的支付接口，则可能导致严重损失。CVSS 作为通用评分工具，无法区分这种使用场景的差异。

评估维度	权重说明
资产权重	CDE 核心资产（处理/存储/传输 CHD）权重最高，CDE 相邻资产次之，非 CDE 资产权重最低
威胁活跃度	是否列入 CISA KEV 目录、是否存在在野利用证据、是否有公开 PoC
暴露程度	互联网直接暴露权重最高，内网可达次之，物理隔离最低
补偿控制	WAF、IPS 等现有控制措施对该漏洞的实际缓解效果

5.4 把修复时限写进政策

对于非“Critical”漏洞，PCI DSS 允许组织通过目标风险分析（TRA）自行确定合理的修复时间框架。建议将时间框架落成书面 SLA 政策：

- **严重（Critical）**：补丁发布后 30 天内（PCI DSS 硬性要求）
- **高（High）**：60 天内（建议值，基于 TRA）
- **中（Medium）**：90 天内（建议值，基于 TRA）
- **低（Low）**：180 天内（建议值，基于 TRA）

关于 TAR 的详细方法论，PCI DSS 标准中已有明确要求，本文不再赘述，具体可参考 PCI DSS《Targeted Risk Analysis Guidance》。需要强调的是，TRA 中的所有时间框架均须以书面形式记录，经管理层审批后作为内部政策执行。

6. 结语

回到开篇的 Next.js 案例，若当时运行了 PCI DSS 要求的漏洞管理流程，此次入侵或许可避免：漏洞公开当日，情报订阅即触发告警、同日，SBOM 快速定位受影响资产、业务影响评估完成后，CDE 内服务器被标记为“Critical”、紧急变更流程随即启动，确保在一个月内（甚至更短）完成所有受影响系统的修复，从而极大可能避免此次入侵。

PCI SSC 漏洞管理框架并不追求“漏洞”发现，其思路更为务实。先发现漏洞，再按风险排序，继而依时间表逐一处置，最终验证修复成效。它为我们提供了一套标准化、可审计的“作战地图”，使漏洞管理从盲目的“救火”转变为有章可循的“防空演习”。

将这套框架真正落地，收获的远不止审计合格。扫描机制、情报来源、风险定级、修复时限、验证环节，一应俱全，攻击者利用漏洞的窗口便从“不确定”变为“可管控”。这才是漏洞管理的根本意义所在。

参考资料

[1]PCI Security Standards Council. *Payment Card Industry Data Security Standard, v4.0.1*. 2024.

[2]PCI Security Standards Council. *Vulnerability Management Infographic*. 2025. https://blog.pcisecuritystandards.org/new-infographic-pci-dss-vulnerability-mangement-processes

[3]PCI Security Standards Council. *Frequently Asked Question: Vulnerability Management*. 2025.

[4]National Vulnerability Database (NVD). https://nvd.nist.gov/

[5]MITRE Corporation. *Common Weakness Enumeration (CWE)*. https://cwe.mitre.org/

[6]CISA. *Known Exploited Vulnerabilities Catalog (KEV)*. https://www.cisa.gov/known-exploited-vulnerabilities-catalog

[7]OWASP Dependency-Track https://owasp.org/www-project-dependency-track

[8]《Next.js 高危漏洞致服务器被黑》案例来源 https://mp.weixin.qq.com/s/Zdw9-cOEj_rlbs96H1YOwA