Separation of Duties

职责分离

Separation of duties—especially separating work between developers

and operations engineers—is spelled out as a fundamental control

in security and governance frameworks like ISO 27001, NIST

800-53, COBIT and ITIL, SSAE 16 exams, and regulations such as

SOX, GLBA, MiFID II, and PCI DSS.

职责分离，特别是开发和运营工程师之间的分离，在安全和治理框架中被称为基本控制，如ISO 27001、NIST 800-53，COBIT和ITIL，SSAE 16，以及诸如SOX、GLBA、Mifid II和PCI DSS等监管制度。

Auditors look closely at separation of duties, to ensure that requirements

for data confidentiality and integrity are satisfied: that data

and configuration cannot be altered by unauthorized individuals,

and that sensitive or private data cannot be viewed by unauthorized

individuals. They review change control procedures and approval

gates to ensure that no single person has end-to-end control over

changes to the system. They want to see detailed audit trails to prove

all of this.

审核员密切关注职责分离，以确保满足对于数据的保密性和完整性要求：该数据和配置不能让未经授权的个人更改，敏感或私人数据不能被未经授权的个人查看。他们审查变更控制程序和批准门，确保没有一个人对系统的更改拥有端到端的控制权。他们希望看到详细的审计跟踪来证明所有上述这些。

Even in compliance environments that do not specifically call for

separation of duties, strict separation of duties is often enforced to

avoid the possibility or the appearance of a conflict of interest or a

failure of controls.

即使在没有特别要求职责分离的合规环境中，也经常采用严格的职责分离来避免可能出现的利益冲突或控制失效。

DevOps, by breaking down silos and sharing responsibilities

between developers and operators, seems to be in direct conflict

with separation of duties. Allowing developers to push code and

configuration changes out to production in Continuous Deployment

raises red flags for auditors. However, as we’ll see in “Compliance

as Code” on page 51, it’s possible to make the case that this can

be done, as long as strict automated and manual controls and auditing

are in place.

DevOps所提倡的打破筒仓和在开发和运营人员之间共担责任似乎与职责分离之间存在直接冲突。在持续部署中，允许开发人员向生产环境推送代码和配置，对审计人员来说是亮了红旗。但是，正如我们将在第51页“合规即代码”看到的，只要严格的自动和手动控制，以及审计，就可以实现上述目标。

Another controversial issue is granting developers access to production

systems in order to help support (and sometimes even help

operate) the code that they write, following Amazon’s “You build it,

you run it” model. At the Velocity Conference in 2009, John Allspaw

and Paul Hammond made strong arguments for giving developers

access—at least limited access—to production:

Allspaw: “I believe that ops people should make sure that developers can see what’s happening on the systems without going through operations… There’s nothing worse than playing phone tag with shell commands. It’s just dumb.

“Giving someone [i.e., a developer] a read-only shell account on

production hardware is really low risk. Solving problems without it

is too difficult.”

Hammond: “We’re not saying that every developer should have root

access on every production box.”

At Etsy, for example, even in PCI-regulated parts of the system developers get read access to production system metrics dashboards (“data porn”) and exception logs so that they can help find problems in the code that they wrote. But any fixes to code or configuration are done through Etsy’s audited and automated Continuous Deployment pipeline.

另一个有争议的问题是允许开发人员接触生产系统，开发人员帮助支持（有时甚至帮助运营）他们写的代码，遵循亚马逊 “你构建它，你运行它“的模式。在2009年的Velocity大会上，John Allspaw和Paul Hammond为给开发者提供了有力的理由至少可以有限制地访问生产系统：

Allspaw：“我相信运营人员应该确保开发人员可以在不经过运营人员的情况下，查看系统上发生的情况……没有比通过shell命令玩电话追逐游戏更坏地事情了。真是太蠢了。

“为一些人（即开发人员）提供生产系统地只读shell帐户的风险很低。没有它，解决问题太难了。”

Hammond：“我们不是说每个开发者都应该有所有生产系统地管理员访问权限。”

例如，在Etsy，甚至在符合PCI监管的系统部件中，开发人员也可以读取生产系统的度量仪表盘（数据色情）和异常日志，以便帮助他们发现自身写的代码中的问题。但是对代码或配置的任何修复需要通过Etsy经过审计和自动化的持续部署管道来完成。

Any developer access to a financial system, even read-only access,

raises questions and problems for regulators, compliance, InfoSec,

and customers. To address these concerns, you need to put strong

compensating controls in place. Limit access to non-public data and

configuration to a minimum. Review logging code carefully to

ensure that logs do not contain confidential data. Audit and review

everything that developers do in production: every command they

execute, every piece of data that they look at. You need detective

change control in place to report any changes to code or configuration.

In financial systems, you also need to worry about data exfiltration:

making sure that developers can’t take data out of the system.

These are all ugly problems to deal with.

任何开发人员访问金融系统，即使是只读访问，向监管机构、合规部、信息安全部和以及客户都提出了[提出了](https://cn.bing.com/dict/search?q=%E6%8F%90%E5%87%BA%E4%BA%86&FORM=BDVSP6&mkt=zh-cn)[疑问](https://cn.bing.com/dict/search?q=%E7%96%91%E9%97%AE&FORM=BDVSP6&mkt=zh-cn)[和](https://cn.bing.com/dict/search?q=%E5%92%8C&FORM=BDVSP6&mkt=zh-cn)[问题](https://cn.bing.com/dict/search?q=%E9%97%AE%E9%A2%98&FORM=BDVSP6&mkt=zh-cn)。为了解决这些问题，你需要把补偿控制措施实施到位。限制对非公共数据和配置的访问到最低水平。仔细查看日志代码确保日志不包含敏感数据。审核和审查开发人员在生产环境中所做的一切：他们的每一个命令执行，他们看到的每一段数据。你需要探测变更控制，以报告任何更改对代码或配置的变更。

在金融系统中，您还需要担心数据泄露：确保开发人员不能从系统中取出数据。

这些都是棘手的问题。

You also need to realize that the closer developers are to operations,

the more directly involved they will get in regulatory compliance.

This could lead to developers needing to be licensed, requiring

examinations and enforcing strict restrictions on personal conduct.

For example, in March 2015 FINRA issued a regulatory notice proposing

that any developer working on the design of algorithmic trading strategies should be registered as a securities trader.

您还需要认识到，开发人员离运营越近，他们越直接参与监管合规。这可能导致开发人员需要获得许可，需要检查和严格限制个人行为。

例如，2015年3月，FINRA发布了一份监管通知，建议任何致力于算法交易策略的开发人员，都应该注册为证券交易员。