软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006847]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于指标依赖模型构建与监控的攻击检测方法 *

王立敏[1,2], 卜磊[1,2,3], 马乐之[1,2], 于笑丰[4], 沈宁国[5]

[1](计算机软件新技术国家重点实验室(南京大学), 江苏 南京 210023)

[2](南京大学 计算机科学与技术系, 江苏 南京 210023)

[3](南京大学 软件学院, 江苏 南京 210093)

[4](南京大学 商学院, 江苏 南京 210093)

[5](华为技术有限公司, 广东 深圳 518129)

通讯作者: 卜磊, E-mail: bulei@nju.edu.cn

摘 要: 随着攻击技术的不断演进, 防御的难度也与日俱增. 为了及时有效地识别和阻断攻击的实施, 学术界与工业界已提出众多基于攻击检测的防御技术. 现有的攻击检测方法主要着眼于攻击事件, 通过识别攻击特征或者定位异常活动来发现攻击, 分别具有泛化性和攻击导向性不足的局限性, 容易被攻击者精心构造的攻击变种绕过, 造成漏报和误报. 然而本文根据观察发现,尽管攻击及其变种可能采用众多不同的攻击机制来绕过一些防御措施, 以实现同一攻击目的, 但由于攻击目的不变, 这些攻击对系统的影响依然具有相似性, 因此所造成的系统影响并不会随攻击手段的大量增多而随之产生对应的增长. 针对该特点, 本文提出基于攻击指标依赖模型的攻击检测方法以更有效地应对攻击变种. 本文所提出的指标依赖模型着眼于漏洞利用后对系统的影响而非变化多样的攻击行为, 因此具有更强的泛化能力. 基于模型指导, 我们进一步采用多层次监控技术, 以迅速捕获定位攻击迹, 最终实现对目标攻击与变种的精确检测, 有效降低攻击检测的误报率. 本文在 DARPA 透明计算项目以及典型 APT 攻击组成的测试集上与现有的基于攻击事件分析的检测方法进行实验对比, 实验表明在预设场景下本文所提出的方法可以以可接受的性能损耗实现 99.30%的检出率.

关键词: 指标依赖模型; 攻击检测; 漏洞利用; 系统状态

中图法分类号: TP311

中文引用格式: 王立敏,卜磊, 马乐之, 于笑丰, 沈宁国. 基于指标依赖模型构建与监控的攻击检测方法. 软件学报. http://www.jos.org.cn/1000-9825/6847.htm

英文引用格式: Wang LM, Bu L, Ma LZ, Yu XF, Shen NG. An attack detection method based on indicator-dependent model construction and monitoring . Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/6847.htm

An attack detection method based on indicator-dependent model construction and monitoring

WANG Li-Min[1,2], BU Lei[1,2,3], MA Le-Zhi[1,2], YU Xiao-Feng[4], SHEN Ning-Guo[5]

[1](State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210023, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

[3](Software Institute, Nanjing University, Nanjing 210093, China)

[4](Bussiness School, Nanjing University, Nanjing 210093, China)

[5](Huawei Technologies Co., Ltd., Shenzheng 518129, China)

Abstract: With the continuous evolution of attack techniques, the difficulty of defense is increasing rapidly. In order to identify and block the attacks in a timely and effective manner, numerous detection-based defenses have been proposed in academia and industry. The current attack detection methods mainly focus on attack behaviors, and find attacks by identifying attack signals or locating abnormal activities.

* 基金项目: 国家自然科学基金（62232008, 62172200）; 江苏省前沿引领技术基础研究专项（BK20202001）
收稿时间: 2022-09-05; 修改时间: 2022-10-10, 2022-12-14; 采用时间: 2022-12-28; jos 在线出版时间: 2023-01-13

These solutions have the limitation of insufficient generalization and attack-orientation respectively and are easily bypassed by attackers' well-crafted behaviors, resulting in false positives and false negatives. However, we observe that the attacks and their variants usually leverage different attack mechanisms to bypass some defenses and achieve the same attack purpose. Since the attack purpose remains the same, the impact of these attacks on the system is still similar, so the caused system impact will not increase correspondingly with the large increase in attack methods. Based on the observasion, we propose an indicator-dependent model based attack detection method, to detect the attack variants more effectively. The proposed model focuses on the impact of the exploits on the system rather than the various attack behaviors, which is more generalizable. Based on the model, we further adopt the multi-level monitoring technology to quickly capture and locate attack traces, and finally achieve the accurate detection of target attacks and variants, which effectively reduces the false alarm rate. The effectiveness of the proposed method is verified by the experiment, compared with existing attack behavior-based detection methods on the attack set composed of the DARPA transparent computing project and typical APT attacks. The experiment shows our solution is able to achieve 99.30% detection accuracy with an acceptable performance cost.

Key words: indicator-dependent model; attack detection; exploition; system status

APT(高级持续性威胁, Advanced Persistent Threat)攻击通过对目标计算机系统进行持续的攻击, 逐步达到攻击者的目的, 常被用于窃取商业机密或者破坏目标设备. 随着互联网的普及与发展, APT 攻击日渐活跃. 据全球知名网络安全公司 FIREEYE 在 2021 年发布的 APT 攻击报告显示, 该公司仅在 2020 年就追踪到 514 种新的攻击族[1]. 攻击者的定向 APT 攻击通常为企业带来较大的经济损失, 为了及时阻断网络攻击, 最大限度降低损失, 需要对网络攻击进行实时的检测.

当前的 APT 攻击检测技术分别通过日志审计[2–5], 二进制插装检查[6–10]或者两者混合检测技术[11]来识别收集到的程序运行时信息中是否存在攻击信号或者异常行为, 以检测攻击. 基于日志审计的攻击检测技术通过收集系统级事件以分析是否存在攻击, 这类方法对系统的性能损耗较小, 但是粒度较粗. 基于二进制插装的攻击检测技术虽然可以获取程序内部的更细粒度的信息, 然而会大大降低程序本身的运行速度. 而上述两者的混合检测技术则试图在性能损耗与检测粒度上做到较好平衡. 现有的基于日志审计与二进制插装检查的攻击检测方法着重关注攻击事件, 通过分析日志或插装记录中的攻击信号或异常行为进行攻击检测. 然而, 随着攻击变种数量与种类的快速上升, 上述基于攻击事件分析的攻击检测方法将因识别攻击信号时所使用的攻击模式库泛化性不足或者定位异常行为时缺乏攻击导向性, 分别导致漏报和误报的问题:

1. 通过识别攻击信号进行攻击检测的方法主要根据已有的攻击模式库识别攻击相关的行为和事件, 然而攻击者通常组合利用多种攻击策略来实现其攻击目标, 尤其是近年来 APT 攻击大量利用零日漏洞, 其中一些未在预期中的攻击行为可以较为轻易地绕过检测机制. 以业内知名的攻击检测工具 Elastic Security[12]为例, 该工具同时内置了基于规则以及基于机器学习的攻击检测引擎. 其预置的检测规则中, 例如 Log4J 漏洞(CVE-2021-44228)利用的检测规则, 限定了只有同时检测到 java 发起网络请求访问特定端口, 且新建了进程, 才认为发现了攻击. 然而当出现相似漏洞利用时, 即使只有端口与 Log4J 漏洞利用不同, 也需要重新构建检测规则, 泛化性不足, 容易出现漏报的问题.
2. 通过定位异常行为进行攻击检测的方法则通过识别与良性行为偏移较大的程序异常行为来识别攻击, 该方法不依赖攻击模式库, 可以缓解上述泛化性不足的问题, 然而也由于该方法缺乏攻击模式导向, 容易对正常的程序行为产生误报. 例如 Webmin 的更新模块本就可以执行系统指令以进行更新, 虽然 Webmin 对其输入进行了一定的限制, 但是攻击者依然可以利用远程代码执行漏洞(CVE-2020-35606)进行绕过以执行其精心构造的恶意指令. 这种混在程序正常执行操作中的恶意行为, 很难在收集的运行时信息中对其进行识别和区分, 容易引发误报.

本文通过观察发现, 虽然为了适应目标设备平台或者绕过防御措施, 攻击变种往往会通过不同的攻击机制, 或者利用不同的漏洞, 来实现相同的攻击目的. 但是由于攻击目的不变, 即便攻击及其变种可能层出不穷, 它们对系统的影响却依然具有相似性. 例如, 某攻击者原本希望利用 Log4j 漏洞(CVE-2021-44228)实现远程代码执行, 但发现受害者已经升级了该软件或者部署了相应的防御, 使得漏洞难以利用. 因此攻击者只能尝试利用邮件服务器软件 OpenSMTPD 漏洞(CVE-2020-7247)来实现远程代码执行, 从而顺利实施 APT 攻击. 该案例中攻击者虽然使用不同的机制实现了远程代码执行, 并使用反向 Shell 获得目标设备的控制权的目的.

但是由于攻击者在实现远程代码执行的过程中, 难免会劫持受害者控制流以执行自己的恶意指令, 尤其是构建反向 Shell, 这会导致原程序中的相关函数无法在规定时间内执行完毕. 因此我们可以使用“典型功能是否合理时间内响应”这一通用监控项, 来同时监控上述两个手段截然不同的攻击.

针对该特点, 本文提出基于攻击指标依赖模型的实时攻击检测方法以应对上述挑战. 与基于事件分析的攻击检测方法通过检测攻击信号或者异常行为来识别攻击不同, 本文提出的基于攻击指标依赖模型的实时攻击检测方法将通过关注攻击事件造成的影响而非攻击事件本身来实现攻击检测, 以提升攻击检测的泛化性, 精准识别攻击及其变种, 同时基于模型指导, 以降低攻击检测的误报率. 本文将以漏洞利用为攻击最小单位, 分析提取漏洞利用所需要满足的条件以及成功利用后对系统产生的影响, 最终构建漏洞利用的指标依赖模型. 此外, 攻击者的一次攻击行动往往需要对多个漏洞进行组合利用, 因而更近一步地, 本文可以通过合并目标漏洞利用的模型为目标攻击构建模型. 基于上述模型, 最终可以提取预警指标以及关键指标, 并用于指导多层次监控技术的部署, 以对系统和相关软件关键部位进行监控, 最终实现对攻击及其变种的及时响应. 实验表明本文方法对 DARPA 透明计算项目与典型 APT 攻击组成的攻击集可达到 99.30%的检出率, 高于目前广泛使用的基于攻击事件进行攻击检测的商用检测工具 Elastic Security.

本文的主要贡献在于:

1. 提出一种新的攻击建模方法. 针对攻击及其变种层出不穷难以建模的问题, 本文利用漏洞的重复性和相似性, 着眼于漏洞的根因以及利用后对系统状态的影响而非变化多样的攻击行为, 提取指标依赖模型, 作为该漏洞利用的模型, 以降低建模的难度, 提高模型的泛化性.
2. 提出基于指标依赖模型的攻击检测方法. 针对攻击变种难以检测, 异常行为定位误报率高的问题, 本文通过分析典型攻击及其变种, 观察到变种经常为了适应目标平台或者绕过防御措施, 通常会利用不同的攻击手段实现相似的攻击目的. 针对该特点, 本文以指标依赖模型为基础, 以提高攻击检测的泛化性. 同时, 通过提取指标依赖模型的关键指标以指导多层次监控器的部署与监控, 进一步降低误报率.
3. 利用广泛使用的 DARPA 透明计算项目以及典型的 APT 攻击组成攻击测试集, 基于该攻击测试集, 与现有的基于事件分析的攻击检测方法进行充分的实验对比, 以体现本文方法的有效性. 实验表明本方法可以以可接受的性能损耗实现较好的检测精度.

本文结构如下, 第 1 章主要对本文背景进行阐述, 通过分析现有的 APT 攻击以及漏洞利用相关情况进行分析总结, 并引出本文的研究动机. 第 2 章则对本文提出的基于指标依赖模型的攻击检测方法的总体框架进行介绍. 第 3 章与第 4 章分别详细阐述本文方法中的指标依赖模型的构建方法以及基于该指标依赖模型的攻击检测技术. 第 5 章则主要对本文提出的基于指标依赖模型的攻击检测方法进行方法实现, 并且针对该方法进行有效性评估. 第 6 章主要对相关工作进行介绍, 并进行相应的总结与分析, 第 7 章则进一步将本文方法与相关工作进行对比与分析, 指出本文方法的优势与局限性, 并引出后续工作.

1 背景与动机

1.1 APT攻击

APT 攻击会在长时间内发起多次攻击, 以逐渐深入目标计算机系统, 直至达到攻击目的. 每一次 APT 攻击活动, 可以根据网络杀伤链[13]划分为 7 个攻击步骤, 2 个阶段, 如图 1 所示.

- **攻击准备阶段:**在该阶段, 当攻击者明确攻击目标以后, 需要收集目标设备信息, 以进行“设备跟踪侦察”. 例如通过扫描目标机器发现其开放的端口或者发现其使用的软件及版本, 以进一步制定攻击计划. 随后, 攻击者需要根据观察到的目标信息, 进行相应的“武器构建”, 针对目标系统存在的弱点构建攻击程序, 供后续使用. 最终, 攻击者需要将构建好的攻击程序送入目标计算机, 实现“载荷投递”.送入计算机的途径通常为构建钓鱼网站, 邮件等, 以欺骗用户自行下载运行. 此外, 攻击者也可以尝试利用目标机器上的远程执行漏洞, 直接侵入目标机器的电脑, 并送入攻击程序.
- **攻击实施阶段:**攻击者需要针对目标系统中存在的漏洞, 进行“漏洞利用”, 以帮助攻击者执行非法

操作, 达成自己的攻击目的, 例如控制目标计算机, 提升自己的权限等. 通过漏洞利用, 可以使攻击者达到一定的攻击目的, 例如通过“安装植入”实现渗透驻点, 利用“命令与控制”实现与受害者的通讯与指令下发. 攻击者通过不断地利用目标设备的漏洞, 逐步实现自身的攻击目的, 直到攻击者可以较为完整地控制目标系统, 实现诸如窃取重要机密等目标. 此时, 一次完整的 APT 攻击结束.

在每一次攻击, 攻击者可以根据目标平台采用不同的攻击手段, 从而组合实现攻击目的. 以 DARPA 透明计算项目[14]中针对 TRACE 的一次 APT 攻击为例, 如图 1 所示. 攻击者首先进行“初步攻击”, 在“跟踪侦查”中, 攻击者通过观察目标设备得知目标设备的系统为 Ubuntu14.04, 使用的浏览器为 Firefox 54.0.1. 随即, 攻击者进行“武器构建”, 根据 Firefox 版本, 可以从 CVE(Common Vulnerabilities & Exposures)漏洞列表中得知该软件版本中的漏洞, 并针对漏洞构建攻击程序. 此外, 攻击者同时还构建了钓鱼网站. 随后, 攻击者尝试“载荷投递”, 引诱受害者来访问该钓鱼网站, 以便利用受害者浏览器中的漏洞. 当受害者访问到钓鱼网站, 攻击者正式实施攻击. 利用浏览器的漏洞, 攻击者在浏览器内存空间中注入恶意程序. 该恶意程序将与攻击者建立临时的联系通道, 实现攻击者对目标设备的“命令与控制”. 此时“初步攻击”已经完成, 但由于临时连接很容易断开, 且其不具有 ROOT 权限, 因此攻击者还将进一步发起更加深入的攻击, 以进行“渗透驻点”. 由于之前初步的攻击, 当前攻击者可以连上目标设备, 因此在更深入的攻击阶段, 将先植入攻击程序, 以完成“安装植入”阶段. 随后攻击者再次进入漏洞利用阶段, 利用提权漏洞, 将攻击者权限提升至 ROOT 权限. 攻击者更进一步地以 ROOT 权限来构建反向 Shell, 以实现“命令与控制”阶段的目的. 自此, 攻击者可以以 ROOT 权限对目标设备进行控制. 最终攻击者可以在目标设备中实现自己的攻击目的. 由上述案例可知, APT 攻击过程中, 不可避免地需要利用多个漏洞来实现自己的攻击目的, 因而本文攻击检测方法后续将着重监控攻击实施阶段以及时识别攻击.

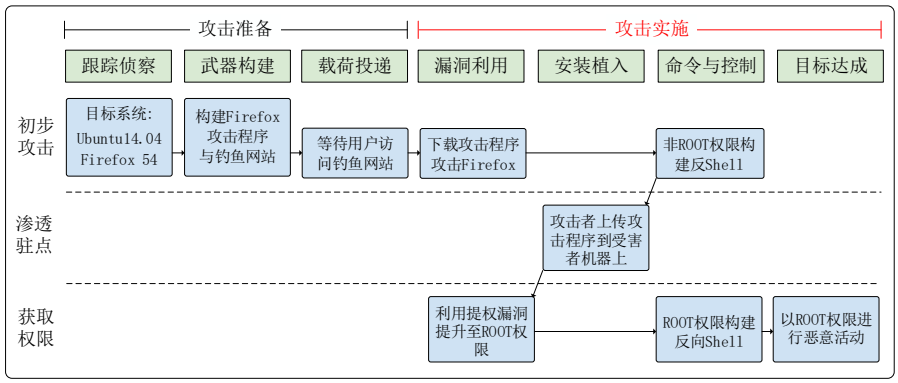

图 1 DARPA 透明计算项目中针对 TRACE 发起的攻击示意图

1.2 漏洞利用

软件漏洞通常是在软件开发过程中未考虑完善所导致. 随着软件供应链的发展和使用, 软件漏洞也有可能通过开源软件库等第三方途径被刻意或者无意地引入, 这为软件带来了无法预期的安全风险. 为了应对日益增长的漏洞, 本研究通过大量统计分析 CVE 中的漏洞信息, 观察到漏洞及其漏洞利用的两个特性.

(1) 特性 1: 为适应目标设备, 或绕过防御方案, 攻击变种往往使用不同的攻击策略来实现同样的目的.

同一攻击族的攻击及变种, 往往使用不同的攻击策略来利用相同的漏洞. 例如近些年十分瞩目的幽灵(Spectre)攻击, 其利用 CVE-2017-5753(bounds check bypass, 边界检查绕过)和 CVE-2017-5715(branch target injection, 分支目标注入)漏洞突破进程间的隔离, 实现非法访问未授权的地址空间. 据已有研究统计[15], 当前的 Spectre 攻击变种根据触发可恶意利用的分支预测策略的不同, 可分为 Spectre-PHT, Spectre-BTB, Spectre-RSB, 以及 Spectre-STL 等. 这些攻击变种均利用了上述分支预测相关漏洞, 但是利用的方式有所不同. 并且, 利用成功后, 攻击者都能达到突破进程间隔离, 窃取目标数据的目的.

（2） 特性 2：虽然攻击与其变种日益增多，但它们所利用的系统缺陷以及利用后对系统的影响并没有随之大量增长.

我们对近 5 年的漏洞类别进行了统计和分析，表 1 表示其中的漏洞占比. 随后我们进一步统计了近五年缓冲区溢出漏洞在溢出类漏洞中的数量及其占比，可以发现，在 7182 个溢出类漏洞中，缓冲区溢出漏洞有 3673 个，占比 51.14%. 这些缓冲区溢出漏洞利用后将产生相似的系统影响，即被利用后目标程序所申请的缓冲区附近的数据，如返回地址，被非法替换. 类似的，我们也统计了远程代码执行漏洞中，发现由“绕过预置的指令筛选规则，非法传入攻击者构造的指令”引起的占 15.48%. 攻击者往往利用这类漏洞实现对目标机器的控制，然而劫持受害者的控制流程序，往往会导致其某些功能无法在预定时间内响应. 从上述案例可以看出同一类漏洞在不同的软件中不断重复地出现，这些漏洞的根因类似，且利用后对系统会造成相似的影响. 因此本文将尝试利用这些相似漏洞的共同根因以及利用后造成的系统影响，而非漏洞利用的行为本身，来进行攻击检测，从而提高攻击检测的泛化性.

表 1 近 5 年（2018-2022 年）CVE 漏洞种类据数量

序号	漏洞种类	平均 CVSS 分数	数量	占比（%）
1	远程代码执行漏洞	7.22	14526	28.02
2	跨站脚本漏洞	3.97	10079	19.44
3	溢出漏洞	6.51	7182	13.85
4	拒绝服务漏洞	5.27	7064	13.63
5	权限绕过漏洞	5.59	3464	6.68
6	SQL 注入漏洞	6.8	2548	4.91
7	目录遍历漏洞	5.53	2055	3.96
8	跨站请求伪造漏洞	5.83	2045	3.94
9	内存损坏漏洞	7.21	1866	3.6
10	信息泄露漏洞	4.83	561	1.08
11	权限提升漏洞	6.83	237	0.46
12	文件包含漏洞	5.89	176	0.34
13	HTTP 响应拆分漏洞	4.79	40	0.07

1.3 研究动机

攻击者发起一次攻击行动通常需要组合利用目标系统中的多个漏洞，软件中潜在的漏洞会为整个系统的安全性带来巨大的安全威胁，因此检测软件中的漏洞是极其重要的工作[16–18]. 然而，由于漏洞具有隐蔽性，此外，随着软件分工越来越明确，漏洞可能会在软件供应链上恶意植入，常规的静态漏洞扫描越来越难以应对此类安全威胁[19,20]，亟需高效的实时攻击检测方法.

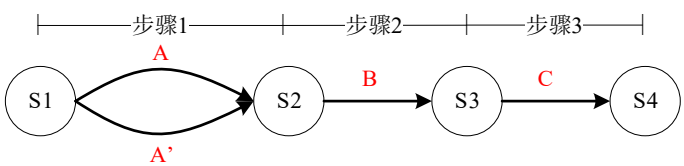

图 2 攻击过程示意图

然而现有的实时检测方法, 基本着眼于攻击行为. 以如图 2 所示的攻击过程为例, 其中每一个节点表示漏洞利用过程中的系统状态(依赖的系统/软件条件或者造成的系统/软件影响). 而有向边则表示漏洞利用过程的各个步骤, 每个步骤实施成功, 将使得系统状态发生改变. 针对攻击 A->B->C 与其变种 A'->B->C, 基于攻击信号识别的攻击检测方法容易被未预期行为 A'绕过, 导致漏报. 而基于异常行为定位的攻击检测方法在检测过程中可能存在大量误报. 为了应对上述挑战, 本文尝试落脚于攻击行为对系统的影响, 而非攻击行为本身, 来构建攻击的指标依赖模型, 使攻击检测更具有泛化性, 例如可通过 S1->S2->S3->S4 这一状态变化序列构建的模型, 来同时识别攻击 A->B->C 以及变种 A'->B->C. 此外, 基于该模型, 可进一步指导监控器的安装部署以降低攻击检测的误报率.

2 基于指标依赖模型的实时攻击检测框架

本文提出的基于指标依赖模型的实时攻击检测框架（如图 3 所示）主要可分为三个阶段, 分别为漏洞信息采集, 指标依赖模型构建, 模型驱动的攻击检测.

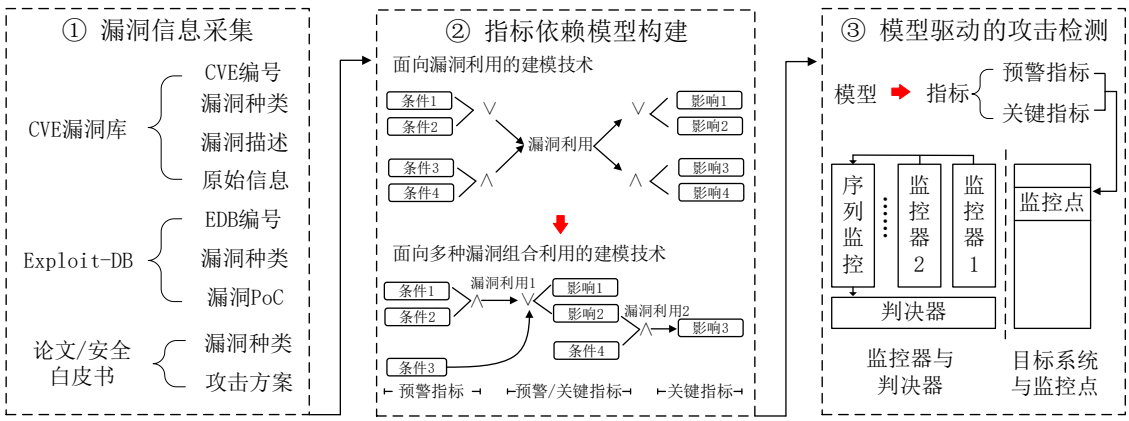

图 3 基于攻击指标依赖模型的实时攻击检测框架

1. 漏洞信息采集: 由于本文所提出的攻击检测机制需要从漏洞以及其相关描述中提取关键的指标, 因此需要首先对不同来源的漏洞信息进行采集与精化. 当前较为主流和权威的漏洞库主要为 CVE, 但是由于 CVE 漏洞库需要审核, 因此也有很多漏洞并不会第一时间公布在 CVE 漏洞库. 此外, CVE 漏洞库中的描述通常仅包含漏洞介绍, 缺乏漏洞利用方式等更为详细的信息. 基于上述原因, 本文将 Exploit-DB, 各种安全论文和安全白皮书也纳入信息收集的渠道. 本文方法将首先从 CVE 漏洞库以及 Exploit-DB 中根据目标平台, 目标厂商等信息筛选得到目标漏洞及其相关信息. 同时, 本文也将从网络与信息安全领域的权威学术会议以及期刊 [1]上搜索漏洞挖掘, 攻击防御相关的论文并收集新挖掘出的漏洞以及新的攻击手段. 此外, 本文也将从知名安全厂商的实验室官网 [2]收集漏洞分析报告, APT 攻击分析报告等与漏洞相关的详细信息.
2. 指标依赖模型的构建: 根据收集到的漏洞描述以及漏洞描述等关键信息, 我们可以进一步对每一个

1 本文收集漏洞信息以及漏洞利用相关资料过程中所参考的权威会议与期刊主要包括: CCS (ACM Conferenceon Computer and Communications Security)，S&P (IEEE Symposium on Security and Privacy), USENIX Security (Usenix Security Symposium), NDSS (Network and Distributed System Security Symposium), TDSC (IEEE Transactions on Dependable and Secure Computing), TIFS (IEEE Transactions on Information Forensics and Security).

2 本文中我们收集漏洞分析报告与 APT 攻击分析报告所参考的知名安全厂商实验室官网主要包括: 腾讯(科恩、玄武、湛泸、云鼎等实验室官网), 启明星辰(ADLab 实验室官网), 绿盟科技(星云、格物、伏影、天机等实验室官网)

漏洞利用的依赖条件以及利用后会对系统产生的影响进行人工分析, 形成预警指标和关键指标. 针对每一次攻击都需要组合利用多种漏洞这一特性, 可进一步对漏洞利用的指标进行合并, 以构建攻击指标依赖模型.

3. 模型驱动的攻击检测: 从模型中提取到必要的预警指标以及关键指标后, 本文将针对每一类指标构建监控器, 并在必要的监控点安置监控器, 以实现对攻击及其变种的实时检测, 在 APT 攻击的早期阶段实现对其的精准识别和及时阻断.

3 指标依赖模型

本章节将阐述如何构建指标依赖模型. 一次完整的攻击具有两点重要特征, 一方面攻击往往可分为准备阶段以及实施阶段, 具有明显的阶段性. 另一方面, 为了适应不同的系统或者防御措施, 攻击者往往需要组合利用多个漏洞以实现攻击目的. 因此本章将首先介绍漏洞利用的建模方法, 随后可再根据需求将漏洞利用模型组合成目标攻击模型. 本章节首先定义相关的概念, 基于这些概念, 我们进一步阐述攻击模型的构建方法.

3.1 定义

定义 1(漏洞(Vulnerability)). 漏洞通常为软件编写过程中有意或无意留下的缺陷. 本文使用$\mathrm{Vul}\epsilon\{\mathrm{DV},\mathrm{UV}\}$表示漏洞, 其中DV表示在档漏洞(Documented Vulnerabilities), 通常被 MITRE[1]等安全组织的漏洞库收录, 拥有一个编号, 而UV则表示非在档漏洞(Undocumented Vulnerbilities).

定义 2(漏洞利用(Exploits)的指标依赖模型). 漏洞利用是黑客利用漏洞进行恶意活动的攻击手段. 本文使用元组$\mathrm{E}=<\mathrm{Name},\ \mathrm{Vul},\ \mathrm{Conditions},\ \mathrm{Effects}>$表示该漏洞利用的指标依赖模型. 其中Name表示漏洞利用名称, Vul表示所利用的漏洞信息. $\mathrm{Conditions}=<\mathrm{Conjunctive},\ \mathrm{Disjunctive}>$为攻击前提, 表示其利用所需要满足的条件. 其中$\mathrm{Conditions.Conjunctive}$表示利用该漏洞时需要同时满足的条件, $\mathrm{Conditions.Disjunctive}$则表示利用该漏洞时只需满足其一的条件. $\mathrm{Effects}=<\mathrm{Conjunctive},\ \mathrm{Disjunctive}>$为攻击影响, 表示该步骤成功后将对系统状态产生的影响. 类似的, $\mathrm{Effects.Conjunctive}$表示利用该漏洞后会同时产生的攻击影响, $\mathrm{Effects.Disjunctive}$则表示利用该漏洞后可能只会发生其一的攻击影响.

定义 3(攻击(Attacks)的指标依赖模型). 攻击是黑客通过利用系统漏洞, 逐步实现攻击目的的方式. 一次攻击通常是对多种漏洞组合利用的结果. 因此, 本文用$\mathrm{A}=(E_1,E_2,\ E_3\ldots E_n)$表示某一次攻击行动, 其中每一个元素$E_i$表示漏洞利用的指标依赖模型.

3.2 攻击模型的构建

3.2.1 面向漏洞利用的建模技术

在攻击过程中往往会涉及到较多的系统缺陷, 包括较为核心的目标系统漏洞, 以及其它并非十分严重的缺陷或者设计并不十分完善的地方. 本文所提出的攻击建模技术需要提取漏洞利用所依赖条件的逻辑关系, 以及利用后对给定平台的影响的逻辑关系. 例如针对同一目标漏洞, 存在原始攻击和攻击变种两种利用方式, 它们分别依赖于两个不同的系统条件, 则该两个系统条件存在逻辑或的关系, 当满足其一时即可实现目标漏洞的利用. 需要注意的是, 最终需要将这些逻辑关系结构化地存储, 以供后续进行运行时监控.

如图 4 所示, 本文所提出的攻击建模技术首先需要考虑攻击过程中所需满足的条件, 例如系统环境, 软件缺陷等, 本文将其作为预警指标. 其次, 本文也需要明确攻击者在攻击过程中改变的可区别于良性程序的系统属性, 文中称其为关键指标. 前者可为系统是否可被攻击提供预警, 而后者可用于判断系统是否正在遭受到攻击. 这些属性需要被提取出来, 分析它们必须同时满足还是满足其一即可, 以作为后续攻击检测的监

[1] MITRE 是全球知名的网络安全组织, 该组织长期维护 CVE (Common Vulnerabilities & Exposures) 漏洞库, CWE (CommonWeakness Enumeration) 常见缺陷列表, ATT&CK 攻击技术框架等广泛使用的安全数据库.

控候选项, 用于驱动监控器的构建与部署.

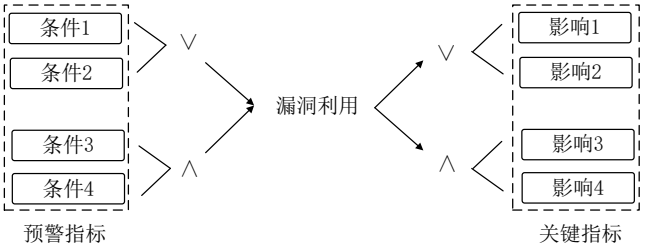

图 4 面向漏洞利用的指标依赖模型

图 5 为面向漏洞利用的建模流程. 当给定目标漏洞列表, 我们需要逐个对其中的相关漏洞进行人工分析, 直到待分析漏洞列表为空. 针对每一个待分析漏洞 Vul, 我们将通过分析 CVE 漏洞描述, 漏洞代码, 漏洞利用程序（PoC）等, 来提取其所依赖的系统条件 Conditions, 以及利用成功后将对系统或者软件造成影响 Effects. 若所提取的系统条件以及造成的影响为首次出现, 则需要将它们分别加入漏洞利用条件集 (Set of Conditions, SC) 以及利用后系统影响集 (Set of Effects, SE).

以漏洞 CVE-2022-0778 OpenSSL 拒绝服务漏洞为例, 该漏洞由 OpenSSL 的相关版本在证书解析时使用的 BN_mod_sqrt() 函数存在的一个错误引起. 该错误会导致 OpenSSL 在非质数的情况下永远循环. 假设该漏洞利用过程的指标依赖模型为E =< Name = “拒绝服务漏洞利用”, Vul = “CVE-2022-0778”, Conditions = {}, Effects = {} >, 其中 Conditions 以及 Effects 在模型初始时为空. 通过分析, 可知该漏洞的根因为 BN_mod_sqrt() 函数中存在特殊的循环, 该循环中具有不可达的退出条件, 因而 Conditions = {“Conjunctive”:{1:“存在具有不可达退出条件的循环”}}. 随后我们进一步分析利用成功后的系统影响 Effects, 通过分析该漏洞的描述可知, 当该漏洞被触发时, BN_mod_sqrt()函数会因为死循环而无法及时退出. 因此分析后, Effects = {“Conjunctive”:{1:“目标函数未在规定时间内响应”}}. 最终上述的分析结果也将被合并到全局共用的漏洞利用条件集 SC 以及利用后系统影响集 SE, 以供后续漏洞的分析.

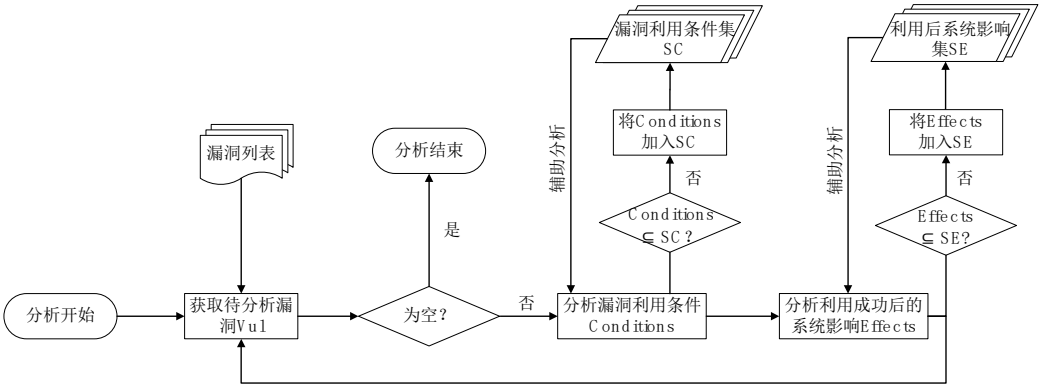

图 5 面向漏洞利用的建模技术

3.2.2 面向多种漏洞组合利用的攻击建模技术

由于攻击行为往往由若干漏洞利用组成, 因此我们为漏洞构建指标依赖模型后, 可进一步通过引入专家知识, 人工合并涉及到的漏洞利用模型来构建目标攻击的模型. 如图 6 所示, 漏洞利用 1 的成功实施需要前提条件 1 或者条件 2 满足其一, 同时条件 3 也满足. 漏洞利用 1 成功后将同时产生影响 1 和影响 2. 此时影响 2 以及条件 4 将成为漏洞利用 2 的前提条件, 从而形成漏洞利用间的指标依赖. 当漏洞利用 2 实施成功后, 会导

致影响 3.

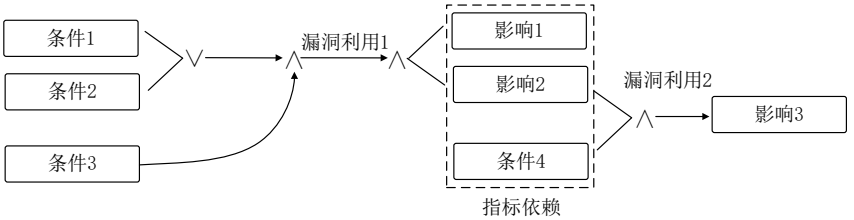

图 6 指标依赖模型示例

当给定目标 APT 攻击时, 本文首先分析并划分其攻击阶段，随后确定其每个攻击阶段所利用的漏洞, 最终根据其每个漏洞利用的指标依赖模型的依赖条件以及利用后的影响, 分析攻击阶段前后的指标依赖关系, 即前一攻击步骤对系统造成的影响, 是否为后一攻击步骤利用成功所需要的条件.

以一次获取目标设备 ROOT 权限控制权的攻击为例, 攻击者分别利用了 CVE-2021-44228 Log4J 远程代码执行漏洞以及 CVE-2020-7247 OpenSMTP 提权漏洞, 前者实现非 ROOT 权限的命令与控制（Command and Control, C&C）连接, 后者进一步进行了权限提升. 令上述攻击案例$A=\{E_1,E_2\}$, 其中E_1表示 CVE-2021-44228 Log4J 远程代码执行漏洞的利用, E_2表示 CVE-2020-7247 OpenSMTP 提权漏洞的利用. 如图 7 所示为攻击 A 的模型, 当攻击者实施E_1时, 一方面需要从外部网络访问目标网页, 多次尝试输入恶意指令字符串. 另一方面需要输入的字符串正好能被 Web 服务器使用 Log4j 组件写入日志. 此外最为关键的是, 输入的字符串正好可以绕过 Web 服务器的字符串筛选机制. 因此漏洞利用E_1的条件 Conditions = {“Conjunctive”:{1:“来自单一来源的网络[写]请求”, 2:“目标软件使用 Log4j 写日志”, 3:“输入没有经过严格的筛查”}}. 当E_1实施成功后, 会构建反向 Shell 与攻击者进行通信. 此时目标程序的控制流被劫持, 成为了与攻击者交互的跳板, 可能会导致原先用于记录日志的函数迟迟没有响应. 因此E_1利用成功后, 其对系统或者目标软件的影响 Effects = {"Conjunctive":{1:"目标函数未在规定事件内响应", 2:"命令行执行/新建进程"}}. 紧接着攻击者继续实施E_2, 此时E_1对系统造成的影响之一, 即“命令行执行/新建进程”, 将成为E_2的前置条件. 当攻击者利用E_1实现与目标设备的交互后, 可以进一步利用目标设备 OpenSMTP 未对输入信息的特殊字符进行编码的漏洞, 构建邮件并在本地进行发送以实现本地提权, 此时获取到的命令行具有 ROOT 权限. 因此E_2的Conditions = {“Conjunctive”:{1:“命令行执行/新建进程”, 2:“特殊符号转义处理不当”}}. E_2 实施成功后, 其 Effects = {“Conjunctive”:{1:“目标函数未在规定时间内响应”, 2:“命令行执行/新建进程(ROOT)”}}.

通过上述方法即可为多漏洞组合利用的目标攻击构建指标依赖模型, 当得到漏洞利用以及目标攻击的指标依赖模型时, 便可从中提取指标体系以用于攻击检测.

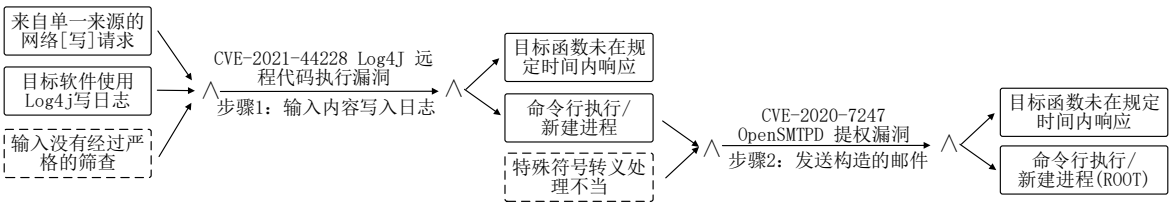

图 7 获取目标设备 ROOT 权限的攻击案例

4 基于指标依赖模型的攻击检测方法

4.1 指标体系

当模型构建完成, 即可根据预警指标以及关键指标进行监控器的构建和部署以进行攻击检测, 这些模型中的指标共同组成了指标体系.

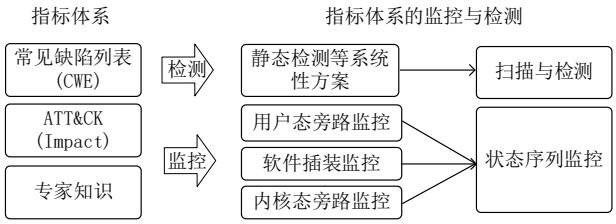

图 8 本文方法所关注的指标体系

如图 8 所示, 本文的指标体系主要由三部分组成, 后续这些指标将由不同的监控机制进行分别监控, 以达到攻击检测的目的:

常见缺陷列表(Common Weakness Enumeration, CWE). CWE 由 Mitre 组织所维护，被广泛采用. CWE 中的项通常是漏洞产生的根因, 在编码的过程中引入, 因此仅会出现在漏洞利用条件集 SC. 例如图 7 中 OpenSMTPD 提权漏洞(CVE-2020-7247)的利用过程, 该漏洞的根因为 CWE-78, 即“OS 中使用的特殊符号转义处理不当”, 该漏洞是由于程序员考虑不完善, 没有对程序的输入信息进行严格筛选导致的. 由于 CWE 是已存在的广泛公认的知识库, 目前已经有很多诸如静态检测等系统化的手段对其进行扫描和检测[21], 因此本文后续将主要关注于 ATT&CK[22]的 Impact 项以及专家知识总结出来的用于动态监控的指标. 此外, CWE 属于已有的指标体系, 可直接加入到漏洞利用条件集 SC 以及漏洞利用后系统影响集 SE 中, 用于指导模型的构建.

ATT&CK 的 Impact 项. 指标体系第二部分是通过对 ATT&CK 框架中的 Impact 项进行二次处理得到. 总结所得的指标同样可以加入到漏洞利用条件集 SC 以及漏洞利用后系统影响集 SE 中用于指导模型的构建. 本文整理了 ATT&CK 框架中的 Impact 项(如表 2 的“ATT&CK 编号”以及“Impact 项”所示), 并总结了监控指标（如表 2 第一列“监控指标”所示）. 以其中最为常见的 T1485 数据破坏项为例, 某些攻击可能会导致系统中重要数据被破坏. 针对该指标, 我们需要监控数据的完整性, 通过对指定的重要数据进行数据完整性的监控, 以实时检测系统中是否正在遭遇此类破坏数据的攻击.

表 2 ATT&CK 中指标项与监控目标

监控指标	ATT&CK 编号	Impact 项
完整性（数据）破坏	T1531	账号访问移除（Account Access Removal）
	T1485	数据破坏（Data Destruction）
	T1486	造成影响的数据加密（Data Encrypted for Impact）
	T1565	数据操作(Data Manipulation)
		存储数据操作（Stored Data Manipulation）
		传输数据操作（Transmitted Data Manipulation）
		运行时数据操纵（Runtime Data Manipulation）
	T1491	篡改（Defacement）
		内部篡改（Internal Defacement）
		外部篡改（External Defacement）
	T1561	磁盘数据永久删除（Disk Wipe）
		磁盘内容擦除（Disk Content Wipe）

		磁盘结构擦除（Disk Structure Wipe）
系统资源异常	T1499	终端拒绝服务（Endpoint Denial of Service）
		操作系统耗尽泛洪 OS Exhaustion Flood
		服务耗尽泛洪 Service Exhaustion Flood
		应用耗尽泛洪 Application Exhaustion Flood
		应用或系统漏洞利用 Application or System Exploitation
完整性（代码/组件）破坏	T1495	固件损坏(Firmware Corruption)
	T1490	阻碍系统恢复（Inhibit System Recovery）
网络流量/频次异常	T1498	网络拒绝服务（Network Denial of Service）
		直接网络泛洪（Direct Network Flood）
		反射放大（Reflection Amplification）
系统状态异常（处理器，内存，服务，异常开关机）	T1496	资源劫持（Resource Hijacking）
	T1489	服务停止（Service Stop）
	T1529	系统关闭/重启（System Shutdown/Reboot）

基于专家知识提取的平台无关指标. 指标体系第三部分必须利用专家知识在建模的过程中分析补充得到. 为了使得该类指标合理可实践, 我们参照现有工作的做法[23], 向企业专家进行咨询, 并一起总结分析了 Linux 上典型的软件漏洞种类(本文实现中, 主要选取表 1 中占比最高的 4 种常见软件漏洞各 50 个, 进行总结分析). 表 3 为汇总后的常见漏洞种类及其利用机制和监控指标. 每一类漏洞根据根因, 可细分为多种利用机制, 我们分析了这些漏洞利用后会对系统造成的影响, 并总结出相应的监控指标.

以拒绝服务漏洞为例, 它主要有 4 种利用方式, 其中之一为攻击者通过大量耗尽系统资源, 导致网络, 处理器, 内存等系统资源被异常消耗, 因此需要对这些系统资源进行必要的监控. 类似的, 攻击者也可以通过恶意触发关闭程序信号, 甚至通过劫持目标程序访问未授权内存空间, 最终引发内存错误导致程序异常关闭. 此外, 攻击者也可以通过精心构造特定的输入信息以触发目标软件进入死循环, 从而实现拒绝服务的目的. 因此我们也需要将“程序异常关闭”以及“典型功能未在合理时间内响应”作为监控指标.

表 3 漏洞的种类及其利用机制和监控指标

序号	漏洞种类	漏洞利用机制	监控指标
1	拒绝服务漏洞	大流量耗尽资源	出现大量异常流量以及系统资源消耗/程序异常关闭/未在规定时间内响应
		触发关闭信号, 停止工作	
		触发内存错误, 停止工作	
		触发死循环, 无法工作	
2	远程代码执行/ 权限提升漏洞	控制流劫持	命令行的执行/进程的创建/未在规定时间内响应/多个状态机的一致性/调用链或消息链的完整性/模块间不符合预期的调用次序
		通过伪装绕过限制条件	
3	溢出类漏洞	缓冲区溢出	返回地址的不一致/不符合预期的参数和返回值/配置信息等输入的核心数据的完整性被破坏/变量阈值异常
		整数溢出	
4	内存损坏漏洞	访问未初始化内存	程序异常崩溃
		访问未授权的内存/越界访问	
		内存泄露/释放非堆区或未分配的内存	

4.2 指标体系的监控

根据监控范围, 监控对象的不同, 我们将 4.1 小节中提及的指标针对性地细化为表 4 中的监控项, 为后续实现多层次部署, 全方位监控奠定技术基础. 需要注意的是, 此处的监控项还可根据实际需求进一步地扩展和新增.

表 4 中的“1. 异常流量以及系统资源消耗监控”项主要用于覆盖表 2 中的“系统资源异常”, “网络流量/频次异常”, “系统状态异常”指标以及表 3 中“拒绝服务漏洞”造成的“出现大量异常流量以及系统资源消耗”影响. 它们都有可能通过泛洪等方式异常占用目标机器的网络、系统资源或者软件资源, 使它们进入拒绝服务的状态, 从而达成攻击目的.

同理, 表 4 中的“2. 数据资源监控”项可以覆盖表 2 中的“完整性（数据）破坏”以及表 3 中的“配置信息等输入的核心数据的完整性”等监控目标. 上述两类监控项, 不需要获取目标程序中的详细状态, 因此通过用户态旁路监控技术即可实现监控任务.

表 4 中的“3. 模块代码监控”可覆盖表 2 中的“完整性（代码/组件）破坏”的监控指标, 同时也可覆盖表 3 中“返回地址不一致”等目标程序内部重要状态相关的监控指标. 因此针对这一大类的监控指标, 我们需要对目标程序的指定监控点进行插装, 以提取必要的程序运行信息.

同时为了防止程序在运行过程中被篡改或者被劫持, 本研究认为同样需要在内核态进行旁路监控, 以对程序的运行时活动进行监控. 例如可利用“4. 内部快照”以及“5. 模式检查”的监控项目来覆盖“模块间不符合预期的调用次序”等监控指标.

此外, 由于现有的攻击活动, 往往需要组合利用多个漏洞实现, 仅依靠上述的独立监控器不足以确定完整的目标攻击. 因此本文进一步设置了监控器序列的监控, 用于对监控器报告的信息进行进一步分析.

表 4 目标监控项

监控项
1. 异常流量以及系统资源消耗监控
- 1.1 心跳
- 1.2 网络流量/频次监控
- 1.3 程序异常关闭/崩溃
- 1.4 系统资源消耗
-> 1.4.1 CPU 资源
-> 1.4.2 内存资源
2. 数据资源监控
- 2.1 数据资源完整性
-> 2.1.1 日志完整性
-> 2.1.2 重要数据完整性（账户, 权限等）
-> 2.1.3 外部动态链接库完整性
-> 2.1.4 配置文件完整性
3. 模块代码监控
- 3.1 调用链或消息链的完整性
- 3.2 典型功能是否合理时间内响应
- 3.3 函数参数和返回值
- 3.4 变量阈值检测
- 3.5 调用返回地址的一致性
- 3.6 状态机的一致性
->3.6.1 变量值变迁
->3.6.2 模块交互状态变迁
（文件系统状态机, 协议状态机等）
4. 内部快照
- 4.1 代码的完整性

	– 4.2 核心数据的一致性
	– 4.3 命令行的执行/进程的创建
5. 模式检查	
	– 5.1 模块间依赖关系和调用次序
6. 监控器序列的监控	

利用上述监控项，不仅可以对漏洞利用的关键指标进行及时报警，同时也可以对组合利用多个漏洞的攻击进行检测. 依然以图 7 的攻击案例为例，根据该攻击模型可知，可通过“3.2 典型功能是否合理时间内响应”，“4.3 命令行的执行/进程的创建”等监控项的信息来判断“目标功能未在规定时间内响应”以及“命令行执行/进程新建”两大指标是否满足，若满足，则说明可能存在攻击者利用了远程代码执行漏洞，并建立新进程与攻击者建立起连接，从而导致该进程无法及时结束运行. 若后续，进一步检测到“命令行执行/进程新建”，且“进程为 ROOT 权限”，同时“目标功能未在规定时间内响应”，则说明可能攻击者利用先前构建起的连接，进一步实现了提权. 为了实现该案例中的攻击检测机制，下文将详细介绍监控器的构建与部署以及攻击模型驱动的攻击识别.

4.3 监控器的构建与部署

监控项可用于指导监控器的构建与部署，但是一方面若不加分辨地将所有监控点均以插装的方式实现，将大大降低系统与目标程序的运行效率. 另一方面，仅使用一种监控技术很难覆盖所有监控项. 因此，为了降低性能损耗，且尽可能全面地覆盖监控项，本文将针对监控项的特点进行多层次部署，以实现全方位监控.

表 5 目标监控项与相应的监控技术

监控技术	监控项
用户态旁路监控	1. 异常流量以及系统资源消耗监控 2. 数据资源监控
插桩监控	3. 模块代码监控
内核态旁路监控	4. 内部快照 5. 模式检查
状态序列监控	6. 监控器序列的监控

根据表 4 中各监控项的特点，本文根据监控器特点，将其归入对应的监控技术，分别为用户态旁路监控，插桩监控，内核态旁路监控以及状态序列监控，结果如表 5 所示. 图 9 则为不同层次监控技术的部署示意图.

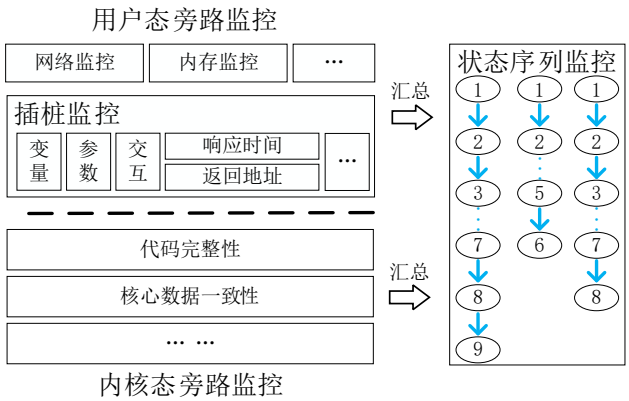

图 9 不同层次监控技术的部署

用户态旁路监控:用户态旁路监控主要用于监控一些无需高权限即可访问到的资源, 例如日志、配置文件等容易被篡改的重要外部数据. 因此我们也需要进一步在用户态设立统一的监控器, 来监控系统资源以及重要的外部文件. 例如本文通过 Python 的 psutil 库实现对 CPU 以及内存占有率的监控, 一旦指标出现异常, 将及时进行通知. 此外, 本文也根据给定的配置选项, 得知需要进行监控的文件地址, 使用 MD5 编码对其完整性情况进行监控, 对未授权的篡改进行及时的报警. 这类监控器均部署在用户态, 可根据需求对目标进行监控, 无需 ROOT 权限.

插装监控:对于目标软件, 也需要在运行时进行必要的监控, 以防止其中存在潜在的漏洞或者供应链中植入的后门被攻击者所利用. 为了能深入获取程序中的一些重要状态, 以供本文的攻击检测方法进行分析, 需要对目标软件进行必要的插装. 例如可以根据监控器配置信息, 对目标代码行中的指定变量的值域进行监控. 类似的, 也可以根据给定的监控器配置信息, 在一些重要函数的起始与返回位置记录时间戳, 监控该函数是否可在合理时间内结束执行. 需要注意的是, 某些函数会在执行过程中新建子进程, 此时函数执行时间, 需要配合内核态旁路监控技术中的“进程新增/减少监控”才可实现子进程执行时间的记录.

内核态旁路监控:一些重要的信息, 需要在内核态才可见, 例如运行时载入内存的程序代码, 要防止代码被恶意篡改等问题, 有必要在内核态对这类重要资源进行监控. 此外, 针对进程等内核态才可获取完整信息的资源, 我们也基于内核中的 Netlink Connector 技术（用于内核态与用户态之间, 或者用户态进程之间的 IPC 通信）实现了进程监控器, 从而获得新增的或者减少的进程信息.

需要注意的是, 在进程监控方面, 传统的方法通常是利用 Linux 预加载技术, 构建 execv 等 linux 进程创建相关的系统调用的钩子, 以在这些函数被调用的时候, 通过钩子监控到进程创建行为[24]. 也有一些内核态的监控器, 通过监控重要系统调用的执行来监控进程创建行为[25]. 然而, 事实上, 一些特殊的攻击, 例如 ROP 攻击, 通常会通过“INT 80H”来避免对系统调用的使用, 因而可能会绕过上述的这些监控技术. 但是本文仅关注于“进程新增”这一状态, 而对进程如何创建的（进程创建行为）并不关注. 因而我们采用 Netlink Connector 技术, 当进程创建时, 会通知 Netlink Connector, 从而被我们捕捉到. 通过该方法, 无论什么方法创建的进程, 都能被我们捕获, 从而避免了传统进程监控技术可能出现的漏报问题.

状态序列监控:利用上述 3 类不同层次的监控技术可对目标系统进行较为完善且高效的监控, 通过汇集这些监控器的信息, 可以进一步在状态序列监控器处进一步对其进行汇总和分析. 以识别完整的攻击路径并及时警报.

4.4 攻击模型驱动的攻击识别

当得到攻击模型并且部署响应的监控器后, 我们将接受各监控器监控结果以检测正在发生的漏洞利用. 如算法 1 所示, 第 2-6 行开始不断地接受来自监控器的结果. 这里假设监控器结果$e =< Indicator, Es >$, 其中 Indicator 表示当前上报指标, Es 则表示与当前指标关联的漏洞利用. 当确实收到监控器上报的指标异常, 第 7 行开始遍历与当前指标异常相关联的漏洞利用. 首先第 8 行设定标记 SEFlag, 其中一个标记位 SEFlag [0]表示当前攻击步骤的依赖条件 Conditions 是否满足事先构建的逻辑条件, 而另一个标记位 SEFlag [1]表示当前步骤利用成功后对系统的影响是否满足事先构建的逻辑条件. 第 9 行将当前攻击步骤的依赖条件以及影响条件合并为一个指标集, 方便后续在第 10-15 行遍历匹配. 第 17 行到 27 行, 算法对当前漏洞利用的依赖条件的匹配情况进行分析和处理. 算法分别遍历当前漏洞利用的依赖条件中需要同时满足的条件集合(Conditions.Conjunctive)以及满足其一即可的条件集合(Conditions.Disjunctive), 以判断它们的逻辑关系是否满足, 若满足则在 26 行将标记 SEFlag [0]设为 1. 类似的, 后续也将在第 29-39 行遍历利用后的系统影响中会同时出现的状态变化（Effects.Conjunctive）以及出现一次即可的状态变化（Effects.Disjunctive）, 以判断当前漏洞利用的关键指标是否满足相应的逻辑关系. 第 41-43 行表示, 若当前漏洞利用的预警指标已全部满足, 则将预警信息写入日志. 第 44-47 行则表示, 若当前漏洞利用的关键指标已全部满足, 则将发出警报, 并将当前漏洞利用信息写入漏洞利用警报序列以供后续的基于指标依赖模型的攻击检测技术进一步处理.

算法 1. 基于指标依赖模型的漏洞利用检测(ExploitDET).

输入: 漏洞利用指标依赖模型集合SE = $\{E_1, E_2, \ldots E_n\}$, 漏洞利用模型数量 n.

```
1 ExploitDET(SE, n):
2    WHILE True:
3         e=RecvMonitors() //从监控器发送的事件序列中取下一个指标异常事件
4         IF !e:   // e =< Indicator, Es >, Indicator 为当前上报指标, Es 则为与当前指标关联的漏洞利用
5           continue
6         ENDIF
7         FOR Cur_E in e.Es: //Cur_E 表示遍历到的漏洞利用
8              SEFlag[2]={0}
9              CEs= Cur_E. Conditions.Conjunctive ∪ Cur_E. Conditions.Disjunctive
                   ∪Cur_E.Effects.Conjunctive ∪ Cur_E. Effects.Disjunctive
10             FOR item in CEs: //将监控到的异常指标与当前漏洞利用的依赖条件和系统影响进行匹
配
11                  IF item. Indicator==e.Indicator:
12                       item.flag=True
13                            break
14                  ENDIF
15             ENDFOR
16             /* 判断当前遍历到的漏洞利用中，其前置条件 Conditions 是否满足预置的逻辑关系 */
17             flag_conjunctive=True
18             flag_ disjunctive=False
19             FOR item in Cur_E. Conditions.Conjunctive:
20                       flag_conjunctive= flag_conjunctive ∧ item.flag
21             ENDFOR
22             FOR item in Cur_E. Conditions.Disjunctive:
23                  flag_ disjunctive = flag_disjunctive ∨ item.flag
24             ENDFOR
25             IF flag_conjunctive==True ∧ flag_disjunctive==True:
26                  SEFlag [0]=1
27             ENDIF
28           /* 判断当前遍历到的漏洞利用中，其系统影响 Effects 是否满足预置的逻辑关系 */
29             flag_conjunctive=True
30             flag_ disjunctive=False
31             FOR item in Cur_E. Effects.Conjunctive:
32                  flag_conjunctive= flag_conjunctive ∧ item.flag
33             ENDFOR
34             FOR item in Cur_E. Effects.Disjunctive:
35                       flag_ disjunctive = flag_disjunctive ∨ item.flag
36             ENDFOR
37             IF flag_conjunctive ∧ flag_disjunctive==True:
38                   SEFlag [1]=1
```

```
39              ENDIF
40          /* 进行监控结果进行判决，并进行上报 */
41          IF SEFlag [0]:
42              Log(Cur_E)  //预警指标满足, 记录以提示
43          ENDIF
44          IF SEFlag [1]:
45              Alert(Cur_E)      //关键指标满足, 发出警报
46              SendToQueue(Cur_E) //加入到漏洞利用警报序列
47          ENDIF
48      END FOR
49  END WHILE
```

算法 1 已经对监控器发送的异常信息进行初步处理, 并对检测到的漏洞利用发出警报. 当预警指标或者关键指标满足时, 算法 1 将分别记录(算法 1 第 42 行)或者发出警报(算法 1 第 45 行), 提醒用户可被利用或者正在被利用的漏洞(Cur_E). 用户可根据所提示的漏洞信息, 利用该漏洞的指标依赖模型, 即漏洞利用所依赖的条件(Cur_E. Conditions)以及所造成的影响(Cur_E. Effects), 定位高风险软件模块和系统组件, 以确认问题所在, 及时阻断攻击进程. 为了进一步对完整 APT 攻击进行检测，需要在算法 2 中采用状态序列监控分析这些监控器警报组成的漏洞利用状态序列是否与目标攻击的指标依赖模型相匹配. 若匹配, 状态序列监控则认为系统正在遭受目标攻击, 将及时发出危险警报以供安全人员分析并加固系统.

算法 2 第 2 行 SA_INDEX 为每一个攻击模型设立索引, 用于标记攻击模型中的最新攻击步骤. 第 3-7 行, 该检测算法便开始不断地接受漏洞利用警报. 算法 2 的第 4 行, 监控异常信息将按时间顺序, 以序列的形式不断地送入该算法. 当接收到非空警报, 第 8-17 行开始遍历攻击模型并尝试匹配刚接收到的漏洞利用警报. 第 9 行获取当前遍历到的攻击. 第 10 行用于获取当前遍历到的攻击的最新攻击步骤, 也即漏洞利用. 第 11-13 行表示, 若当前攻击模型中的最新漏洞利用正好与发出漏洞利用警报相匹配, 则认为该攻击步骤已经实现, 将进入对下一个攻击步骤的匹配, 索引向后移动. 第 14-16 行进行判断, 如果已经是该攻击的最后一步, 则发出攻击警报. 根据攻击警报所对应的攻击, 用户可及时进行紧急响应, 事后也可根据攻击警报中所记录的攻击的每一阶段的漏洞利用信息, 进行进一步复盘.

算法 2. 基于指标依赖模型的攻击检测算法(APTDET).

输入: 目标攻击模型集合SA = $\{A_1, A_2, \ldots A_n\}$, 攻击模型数量 n.

```
1 APTDET(SA, n):
2     SA_INDEX[n]={0}//为每一个攻击模型设立一个索引
3     WHILE True:
4         E=RecvFromQueue()//从漏洞利用警报序列中取下一个报告事件
5         IF !E:
6             continue
7         ENDIF
8         FOR i from 0 to n:
9             A=SA[i]
10            Cur_E= A[SA_INDEX[i]]
11            IF Cur_E. Effects ==E. Effects:
12                SA_INDEX[i]++
13            ENDIF
14            IF SA_INDEX[i]==len(A):
```

```
15                Alert(A)
16            ENDIF
17        END FOR
18    END WHILE
```

通过算法 1 与算法 2, 可实现基于指标依赖模型的攻击检测方法. 下文将对本文所提出的攻击检测方法进行评估, 以表明其有效性.

5 基于指标依赖模型的攻击检测方法实现与实验分析

本章首先介绍本文提出方法的具体实现, 并介绍用于对比的商用 APT 攻击检测软件 Elastic Security. 后续将设计实验, 分别从 4 个研究问题对本文方法进行进一步的评估.

5.1 方法实现

本小节主要阐述本文所提出的攻击检测方法的具体实现, 如图 10 所示, 为本文所提出的基于指标依赖模型的攻击检测方法的整体配置框架. 用户首先通过 WEB 用户界面进行监控器的定制, 随后相关的监控器配置信息将通过通信接口传送给后端模块, 经过配置解析后, 分发给驱动器以进行监控器的部署. 后续监控器监控到相应的信息后, 也会将处理后的结果通过通信接口传送到 WEB 用户界面供用户分析使用.

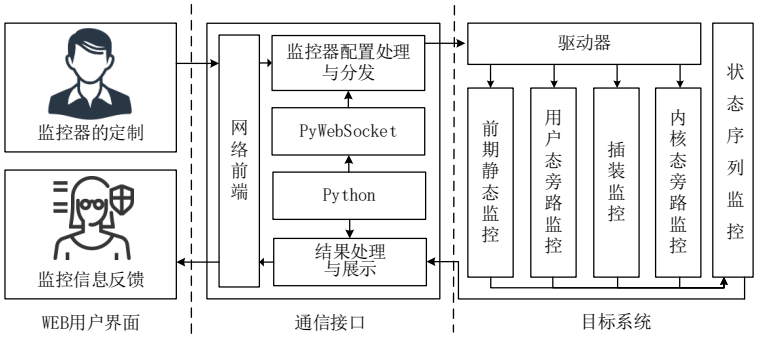

图 10 基于指标依赖模型的攻击检测方法实现框架

Web 用户界面: 本文针对监控项构建了用户界面以供用户按需选择并配置, 如图 11 所示. 其中自定义表单所预先设定的监控项(图 11 右侧)表示所提供的监控能力, 随后用户可以从所提供的监控中, 根据自己的系统平台以及目标软件监控需求, 自行选择所需要的监控项到图 11 左侧白色区域. 用户选择完毕并填写相应配置信息后, 将通过通信接口, 传给后端进行解析并部署监控器.

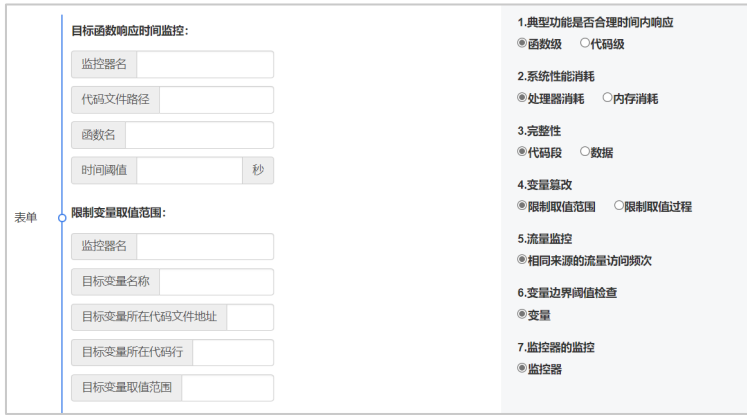

图 11 监控器定制界面

类似的, 我们也为监控信息结构构建了用户界面, 如图 12 所示. 当监控有结果时, 将通过通信接口从后端汇总于该页面, 以便用户进一步进行风险的确认并且及时采取措施.

图 12 监控信息反馈界面

通信接口: 用于前端页面与后端监控平台进行交互, 使用 Python 进行实现. 其中“监控器配置处理与分发”利用 PyWebSocket 接受用户配置完毕的信息, 启动“驱动器”并将相关的监控配置转发给驱动器. 后续将由“驱动器”解析并启动监控体系. 当后续监控有结果, 将回传结果给通信接口, 并最终在用户界面处(图 12)中显示以提醒用户.

目标系统: 驱动器开始工作后, 将对用户提供的监控器配置文件进行解析, 并将不同的监控器分发给目标系统上相应的监控机制以开始监控工作. 在运行时监控过程中, 每个监控器根据其实现方式获取目标程序或者系统环境的运行数据, 通过确认数据是否满足漏洞利用的监控指标来判断目标程序的运行是否出现异常. 异常信息将进一步上报状态序列监控, 以对所有监控器结果进行统筹分析与处理.

5.2 实验配置

实验对比的攻击检测方法: 尽管当前学术界与工业界均提出较多的攻击检测机制[2–11], 然而, 一方面部分方法并未开源, 另一方面大部分攻击检测软件仅采用单项检测技术, 即采用基于攻击信号识别的攻击检测或者基于异常行为定位的攻击检测技术. 由于 Elastic Security[12]同时集成了上述两类典型技术, 并被业界广泛采用, 因此本文选取商用 APT 攻击检测软件 Elastic Security 作为对比对象.

Elastic Security 将 SIEM 威胁检测能力与端点检测和响应能力结合, 通过 Agent 客户端在网络中各节点收集系统事件并汇总到中心节点. 中心节点同时采用基于规则的攻击检测引擎以及基于机器学习的异常检测机制对收集的网络事件进行审计, 全方位地保护系统.

实验环境: 本文实验环境为三台 Linux 设备组成的小型网络, 分别如下:

- 设备 1: 部署 Debian10.8 系统, 处理器为 Intel i7 6700, 内存大小 16G.
- 设备 2: 部署 Ubuntu 20.04 系统, 处理器为 Intel i7 6700, 内存大小 16G.

- 设备 3: 部署 Debian10.8 系统, 处理器为 AMD 4800U, 内存大小 16G.

其中, 本文将设备 1 作为中心节点, 本文方法提出的攻击检测方法, 以及 Elastic Security 将部署于设备 1,用于统筹分析各设备上监控到的数据. 此外, 所有设备上均部署本文的监控器以及 Elastic Agent 监控器, 用于收集网络中各设备的数据.

研究问题: 利用上述实验环境, 本文将通过以下研究问题, 来评估本文所提出的攻击检测方法的有效性.

- RQ1: 本文提出的基于指标依赖模型的攻击检测方法的可行性?
- RQ2: 针对攻击及其攻击变种的检测, 本文所提出的方法是否可以达到较好的效果?
- RQ3: 插装软件的运行效率下降如何?
- RQ4: 本文提出的方法对系统性能的损耗如何?

5.3 攻击测试集

为了使得我们组建的攻击测试集更为客观, 本文使用广泛采用的 Darpa 透明计算项目[14]所提供的 APT 攻击案例以及其它典型 APT 攻击作为本实验的攻击测试集. 测试集中的攻击可覆盖远程代码执行, 权限提升等多种典型的攻击类型, 详细说明如下:

- **DARPA 透明计算项目:** 如表 6 所示, Darpa 透明计算项目主要包含 3 组攻击, 分别为 TRACE, CADETS, THEIA. 这 3 组攻击分别采用浏览器钓鱼, Web 服务器(Nginx), 以及浏览器漏洞对系统进行侵入. 需要注意的是, 本研究限定系统环境为 Linux, 因此我们仅考虑该项目中 Linux 相关的攻击, 对于仅能在 Windows 上进行的攻击(例如涉及修改 Windows 注册表的攻击)不在本实验范围内.
- **其它典型的 APT 攻击:** 此外, 为了使得漏洞种类可以覆盖表 1 中前 5 项中与 Linux 软件相关的安全漏洞(CSRF 跨站脚本攻击属于 Web 安全问题, 故暂时不在本研究范围内). 本实验还选用了一些具有代表性的攻击作为实验样本. 例如近期著名的 Log4J 漏洞[26]利用, 网络安全公司 DARKTRACE 跟踪到的一例与之相关的真实 APT 攻击[27], 该攻击利用知名的 Log4J 远程执行漏洞(CVE-2021-44228) 对目标服务器进行攻击. 当攻击者获得控制权后, 在目标服务器内下载挖矿病毒, 利用服务器算力进行持续隐匿的挖矿活动. 此外, 2014 年十分重要的心脏滴血漏洞(Heartbleed)相关的 APT 攻击[28], 以及拒绝服务(Denial of Service, DoS)攻击也将作为攻击样本用以对本研究提出的方法进行评估.

表 6 本实验攻击测试集所包含的攻击样本

编号	攻击名称	攻击模式	涉及的攻击类型				
			远程代码执行	权限提升	攻击程序植入	钓鱼	信息收集
1	TC-TRACES-1	1. 浏览器钓鱼, 实现 C&C. 2. 上传攻击程序 1, 提权打开并进行 C&C, 实现 ROOT 访问. 3. 关闭非 ROOT 的 Shell, 转向 ROOT Shell. 4. 下载攻击程序 2.	√	√	√	√	
2	TC-TRACES-2	1. 利用浏览器钓鱼实现 C&C, 失败	√			√	
3	TC-TRACES-3	1. 利用浏览器钓鱼进行 C&C 2. 上传程序 3 3. 打开程序进行 C&C 4. 扫描目标网络	√		√	√	√
4	TC-CADETS-1	1. 利用 Nginx 实现 C&C. 2. 下载攻击程序 1, 提权打开并进行 C&C, 实现 ROOT 访问. 3. 下载攻击程序 2 尝试注入 SSH	√	√	√		
5	TC-CADETS-2	1. 利用 Nginx 实现 C&C 2. 下载攻击程序 2 尝试直接注入 SSH	√		√		
6	TC-CADETS-3	1. 利用 Nginx 实现 C&C	√	√	√		√

		2. 下载程序 3 提权打开程序 3, 失败 3. 下载程序 4 提权打开程序 4 进行 C&C 4. 不提权, 打开程序 3 进行 C&C, 扫描目标网络				
7	TC-CADETS-4	1. 利用先前的 C&C, 尝试直接注入 SSH, 失败, 关闭该 C&C. 2. 利用 Nginx 进行 C&C 3. 下载攻击程序 5 提权进行 C&C 4. 注入 SSH	√	√	√	
8	TC-THEIA-1	1. 利用浏览器漏洞 C&C 2. 上传攻击程序 1 3. 提权打开程序进行 C&C 4. 攻击者关闭非 ROOT 的 Shell, 并切换到 ROOT 的 Shell	√	√	√	
9	TC-THEIA-2	1. 利用 Nginx 进行 C&C, 下载攻击程序 2	√		√	√
10	TC-THEIA-3	1. 利用 Nginx 进行 C&C, 扫描目标网络	√			
11	LOG4J	1. 利用 LOG4J 进行 C&C 2. 植入挖矿程序	√		√	
12	Heartbleed	1. 利用 Openssl 泄露服务器资源 2. 窃取 SSH 连接证书	√			√
13	DoS	1. 进行 C&C 2. 尝试提权失败, 不小心触发拒绝服务	√	√		

5.4 实验与分析

5.4.1 针对 RQ1: 原理性实验与案例分析

本小节将采用 Log4J 远程代码执行漏洞（CVE-2021-44228）以及邮件服务器软件 OpenSMTPD 远程代码执行漏洞（CVE-2020-7247）作为案例进行实验, 以分析本文所提出的方法在 APT 攻击检测场景下的可行性. 由于上述两个相似漏洞的利用均可实现对目标设备的远程控制, 因此攻击者可以选择其一来实现 APT 攻击. 本小节假设 Log4J 漏洞为已知的漏洞并做好了相应的攻击检测准备. 同时, 我们假定 OpenSMTPD 漏洞为未知漏洞. 当攻击者尝试使用 Log4J 进行攻击未果时, 将利用 OpenSMTPD 漏洞来构建攻击变种继续进行攻击. 我们分别测试 Elastic Security 以及本文提出的方法在已知 Log4J 漏洞利用以及未知 OpenSMTPD 漏洞利用过程中的攻击检测情况, 以说明本文所提出的基于指标依赖模型的攻击检测技术在 APT 攻击检测场景下的可行性.

首先针对 Elastic Security 进行攻击检测实验, 我们将该实验分为两个阶段. 第一阶段为仅使用基于规则的攻击检测引擎, 第二阶段, 我们加入 Elastic Security 提供的基于机器学习的异常检测机制, 与基于规则的检测机制相结合, 共同进行攻击检测. 首先在第一阶段, 根据 Elastic Security 官方提供的 Log4J 漏洞利用的监控规则[29], 如图 13 所示, 可以发现 Elastic Security 对于 Log4J 的检测, 主要关注于 java 的网络访问以及其子进程的情况. 检测规则的第 3-6 行表示, 检测到事件的种类为网络, 并且发起事件的进程为 java, 目标端口为 1389, 389, 1099 等敏感端口. 检测规则第 8-20 行则表示, 当上述事件被检测到以后, 监控器又检测到 java 启动了子进程并且子进程为列举的 12 种之一. 当监控到的事件序列符合上述监控规则时, Elastic Security 将发出 Log4J 漏洞利用的警报. 我们在两天时间内, 随机发起 5 次 Log4J 远程代码执行攻击以及 5 次 OpenSMTPD 远程代码执行攻击, Elastic Security 对 5 次 Log4J 攻击均正确检出. 由于 OpenSMTPD 是 C 语言编写的程序, 并不依赖 Java, 因而仅使用 Log4J 的检测规则, 无法对其进行有效检测. 实验结果也表明 Elastic Security 对 5 次 OpenSMTPD 远程代码执行攻击的检出率为 0.

第二阶段, 基于规则的攻击检测方法与基于异常的攻击检测方法协同工作以进行攻击检测. 我们同样在两天时间内, 随机发起 5 次 Log4J 远程代码执行攻击以及 5 次 OpenSMTPD 远程代码执行攻击. 由于 Log4J 攻击检测规则的存在, Elastic Security 对 5 次 Log4J 攻击均正确检出. 而对于对 5 次 OpenSMTPD 远程代码执行攻击, 基于机器学习的异常检测机制发挥作用, 总共产生 48 个“Unusual processes exexution”的临时警报. 经过进一步人工分析, 其中 4 个进程执行警报为 OpenSMTPD 远程代码执行攻击直接产生的, 用于构建反向 Shell 与攻击者进行连接. 20 个警报为攻击者为了顺利启用反向 Shell 程序而提前发起的“apt-get”进程, 用于

安装相关依赖包. 剩余 24 个进程执行警报为“appstreamcli”, “echo”, “ Nautilus（Debian10.8 的文件管理器）”等正常进程. 因而针对 5 次 OpenSMTPD 漏洞利用, Elastic Security 成功检出 4 次, 有效检出率为 80%, 且在发出的 48 个警报中, 良性程序活动占 24 个, 误报率为 50%.

```
sequence by agent.id with maxspan=1m
 /* java 进程企图访问 LDAP,  RMI 或者 DNS 的标准端口  */
[network where event.category == "network" and
  process.name : "java" and
  destination.port in (1389,  389,  1099,  53,  5353)] by process.pid
 [process where event.type == "start" and
  /* 可疑的 java 子进程 */
  process.parent.name : "java" and
   process.name : ("sh",
                   "bash",
                   "dash",
                   "ksh",
                   "tcsh",
                   "zsh",
                   "curl",
                   "perl*",
                   "python*",
                   "ruby*",
                   "php*",
                   "wget")] by process.parent.pid
```

图 13 Elastic Security 的 Log4J 监控规则

与 Elastic Security 采用的方法不同的是, 本文所提出的方法利用漏洞利用造成的影响, 而非它们的事件来进行监控与攻击检测. 由于远程代码执行类漏洞在成功利用后, 原程序正常的控制流会被劫持, 并被攻击者利用而转去执行攻击者构造的恶意指令, 因而通常会导致一些非预期影响. 例如对于 Log4J 与 OpenSMTPD 而言, 它们在记录日志与收发邮件时, 均需要在较短时间内得到最终运行结果, 并进行响应. 因而当上述两个软件的控制流被劫持, 去执行恶意指令时, 尤其是构造反向 Shell, 长时间与攻击者通讯时, 通常其运行时间会超过预期时间. 根据图 14 中两个漏洞利用的指标依赖模型可知, 同一类型的相似漏洞中, 由于根因与利用后对系统的影响相似, 因此其相关指标具有较高的泛化性. 根据 Log4J 漏洞利用的指标依赖模型, 我们分别针对表 4 中“3.2 典型功能是否合理时间内响应”, “4.3 命令行的执行/进程的创建”两个监控项构建相关的监控器, 并进行攻击检测. 与 Elastic Security 实验一样, 我们在两天时间内, 随机发起 5 次 Log4J 远程代码执行攻击以及 5 次 OpenSMTPD 远程代码执行攻击, 本文方法可对上述两种漏洞利用均正确检出.

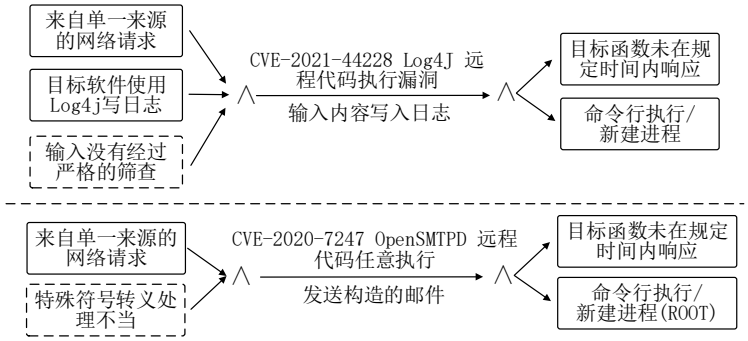

图 14 Log4J 远程代码执行漏洞利用以及 OpenSMTP 远程代码执行漏洞利用的指标依赖模型

由于攻击者通常会使用不同的攻击机制来达到相同或者相似的攻击目的, 因而它们对系统造成的影响往往是相似的. 本小节的实验也证实, 在 APT 攻击场景下, 相对于传统方法关注于变化多样的攻击行为, 本文提出的方法利用攻击造成的影响来检测攻击同样具有可行性. 针对本文方法的攻击检测能力, 后续 5.4.2 小节将进一步设计实验以评估其有效性.

小结：本文提出的基于指标依赖模型的攻击检测方法着眼于漏洞利用后对系统的影响而非变化多样的攻击行为, 因而可以根据攻击对系统造成影响的相似性, 实现 APT 攻击变种的检测.

5.4.2 针对 RQ2: 攻击及其变种的攻击检测实验

APT 攻击往往需要多个漏洞组合利用, 为了验证本文所提出的方法对 APT 攻击检测的有效性, 本小节将使用业界广泛采用的 DARPA 透明计算项目[30–32]以及分别涉及 Log4J, 心脏滴血, 拒绝服务漏洞的 APT 攻击作为攻击样本. 为了进一步地扩充攻击样本, 我们将跟据表 6 中每一个攻击模式所涉及到的漏洞利用, 用表 7 中复现的同种类漏洞利用进行组合替代, 形成新的攻击变种. 例如 TC-CADETS-1 这一 APT 攻击主要通过“利用 Nginx 实现 C&C.”, “下载攻击程序 1, 提权打开并进行 C&C, 实现 ROOT 访问.”来实现对目标机器的高权限访问, 它们分别对应“远程代码执行”, “本地提权”这两个漏洞种类. 当 TC-CADETS-1 的“利用 Nginx 实现 C&C.”无法实现时, 将从表 7 中复现的“远程代码执行”漏洞利用中分别选用 Log4J 代码执行漏洞(CVE-2021-44228), Webmin 远程代码执行漏洞(CVE-2020-35606)等漏洞利用组合成新的 TC-CADETS-1 攻击变种. 同理当 TC-CADETS-1 的本地提权无法成功实施时, 将从表 7 中的本地提权漏洞选用对应的本地提权漏洞利用组合成新的 TC-CADETS-1 攻击变种. 利用上述方法, 可组合成 722 个攻击变种用于评估.

表 7 本研究所复现的漏洞利用

序号	CVE 编号	漏洞名称	漏洞类型	漏洞分数
1	CVE-2021-44228	Log4J 代码执行漏洞	远程代码执行漏洞	9.3
2	CVE-2013-4547	Nginx 权限绕过漏洞	权限绕过漏洞	7.5
3	CVE-2017-7529	Nginx 整型溢出漏洞	溢出类漏洞/信息泄露漏洞	5.0
4	CVE-2014-0160	心脏滴血漏洞	溢出类漏洞/信息泄露漏洞	5.0
5	CVE-2022-0778	OpenSSL 拒绝服务攻击	拒绝服务漏洞	5.0
6	CVE-2020-7247	OpenSMTPD 漏洞	远程代码执行/本地提权漏洞	10
7	CVE-2020-15778	OpenSSH 劫持	远程代码执行漏洞	6.8
8	CVE-2018-12386	Firefox 远程代码执行漏洞	远程代码执行漏洞	5.8
9	-	MySQL 提权漏洞	权限提升漏洞	-
10	CVE-2020-35606	Webmin 远程代码执行漏洞	远程代码执行漏洞	9
11	CVE-2019-10149	Exim 权限提升漏洞	权限提升漏洞	10

在本实验中, 我们分别使用 Elastic Security 以及本文提出的基于指标依赖的攻击检测方法对上述得到的 722 次攻击进行检测, 以评估本方法的有效性. 与现有工作类似[3,7,8,33], 当每次发起攻击时, 我们都会在同一

个机器上模拟多种正常的用户行为, 例如浏览网页, 下载文件, 安装软件包, 编译程序等. 需要注意的是, 在实验过程中, Elastic Security 开启了全部的检测规则, 并且启用了基于机器学习的异常检测功能, 两种检测机制同时工作, 若其中之一检测到攻击则认为 Elastic Security 具有检测该攻击的能力. 而本文提出的基于指标依赖的攻击检测方法则开启了表 4 中总结的监控项所对应的监控器.

表 8 Elastic Security 与本文方法针对 722 次攻击的检出率结果对比

检测机制	Accuracy	F1-Score
Elastic Security (基于检测规则+基于机器学习)	96.37%	98.15%
基于指标依赖的攻击检测方法	99.30%	99.65%

表 9 Elastic Security 与本文方法的误报率结果对比

检测机制		总警报数	有效警报数	无效警报数	误报率
Elastic Security	基于检测规则	1735	1735	0	0
	基于机器学习	678	92	586	86.43%
基于指标依赖的攻击检测方法		1936	1936	0	0

漏报分析: Elastic Security 以及本文提出的方法对 722 次攻击的识别结果如表 8 所示, 结果表明 Elastic Security 可以正确识别到其中 96.37%的攻击, 而本文提出的方法能正确识别到其中 99.30%的攻击. 本文提出的方法识别正确率高, 主要是由于本文关注于攻击造成的影响而非变化多样的攻击行为, 因而在攻击检测上比基于事件分析的 Elastic Security 更具有泛化性.

误报分析: Elastic Security 以及本文提出方法的误报情况如表 9 所示. 由于 Elastic Security 中存在基于检测规则的检测技术以及基于机器学习的异常检测机制两种截然不同的攻击检测机制, 且不同的检测机制误报率差别较大, 因此此处将对这两个机制进行分别讨论.

Elastic Security 中基于规则的攻击检测机制与本文所提出的方法均规定了详细的攻击模式, 例如图 13 中 Elastic Security 的攻击事件检测规则, 以及图 14 中本文提出的指标依赖模型. 根据这些攻击模式可以较为准确地识别攻击, 因而在本实验中暂未发现由 Elastic Security 基于规则的检测机制以及本文方法引发的误报.

然而, Elastic Security 同时内置了基于机器学习的异常检测技术. 该技术在实验中的确弥补了基于规则的攻击检测机制泛化性不足, 易漏报的缺点, 然而也导致了大量的误报信息. 例如 Webmin 远程代码执行漏洞（CVE-2020-35606）, 该漏洞由 Webmin 的更新模块中过滤规则较弱导致, 使得攻击者可以轻易绕过并执行任意指令. Webmin 的更新模块自身也会正常执行“apt-get”命令以更新系统中的一些软件包, 然而 Elastic Security 的异常检测机制, 将其识别为异常, 并报告了大量的警报, 反而给人工排查真正的 Webmin 远程代码执行漏洞利用带去更大的工作量. 根据统计, Elastic Security 机器学习异常检测引擎在本实验中共检测到 678 个警报, 误报率达到 86.43%.

小结: 相较于现有的基于攻击事件分析的攻击检测方法 Elastic Security, 一方面本方法着眼于漏洞利用后对系统的影响而非变化多样的攻击行为, 可达到 99.30%的检测精度, 与其基于规则的检测方法相比具有更高的泛化性. 另一方面, 本方法根据攻击模型指导进行攻击检测, 与 Elastic Security 中基于异常的检测机制相比, 可使攻击检测的误报率更低.

5.4.3 针对 RQ3: 目标软件性能损耗

由于本方法需要对目标软件进行插装以部署部分必要的监控器, 为了尽可能降低程序插装对软件性能造成的损耗, 本文主要采取了两点措施:（1）与通用的插装技术不同, 本文采用指标依赖模型指导的插装, 因而仅在必要的程序点进行插装即可, 大大减少插装需求, 从而降低性能损耗.（2）根据不同的监控项的特性, 采

用用户态/内核态旁路监控, 插装监控结合的多层次监控技术, 尽可能减少插装需求, 以降低性能损耗.

本小节将评估部署的监控器对目标软件是否会造成严重的性能影响. 首先, 本文使用未经过任何处理的程序进行性能评估, 通过对特定功能进行运行并计时, 以获取其所需的时间成本. 随后, 按要求对目标软件进行监控器插入处理, 同样对特定功能进行运行并计时. 最终通过计算前后的时间差以获得我们所需要的性能开销.

使用上述方法, 本实验对 Nginx, Log4J 等实验所涉及的目标程序进行监控器插入前后的运行时间测算, 并反复进行 100 次, 采用平均值. 表 10 结果为各目标程序插装前后的性能下降情况, 平均单个监控器造成的性能开销均低于 1%, 属于可接受的范围内.

小结: 由于多层次插装技术的应用以及指标依赖模型的指导, 使得本文提出的基于指标依赖模型的攻击检测方法对插装的需求大大降低, 从而使得本方案对目标软件每处插装所导致的平均性能影响处于 1%一下, 属于可接受的范围.

表 10 各目标程序性能下降情况

目标程序	监控器插入前 平均耗时（ms）	监控器插入后 平均耗时（ms）	监控器耗时(毫秒)	性能下降百分比
Exim	100065886	100155419	89533	0.089%
Log4J	100555434	100613573	58139	0.058%
OpenSSL	100121489	100221578	100089	0.099%
OenSMTPD	99185821	99896709	710888	0.717%

5.4.4 针对 RQ4: 系统性能损耗

本实验用于评估攻击检测方法对系统的性能损耗. SPEC2006 是一种广泛用于评估计算机性能的标准性能测试基准[34], 包括了整数运算(INT)和浮点运算(FP)两部分性能测试. 因此, 本实验采用 SPEC2006 来评估本文方法的系统性能损耗.

在本实验中, 我们让 SPEC2006 在以下两种不同的场景下运行, 并分别记录耗时:

- 场景 1, 系统中仅运行原始的程序, 不运行攻击检测系统.
- 场景 2, 系统中部署了攻击检测系统.

我们分别记录了 SPEC2006 程序在这两种场景下的时间成本. 数据如图 15 所示. 在此图中, 蓝色矩形表示场景 1 中执行 SPEC2006 的时间成本, 橙色矩形表示场景 2 中 SPEC2006 中的程序运行时间成本. 从图中可以看到, 在两个场景下, SPEC2006 耗时非常靠近. 经过计算, 可得平均性能成本仅为 5.497%. 这说明本文提出的检测方法对系统性能影响较小, 是可以接受的.

小结: 由上述实验可知, 本文所提出的基于指标依赖模型的攻击检测方法对系统的性能损耗为 5.497%, 同样处于可接受的范围内.

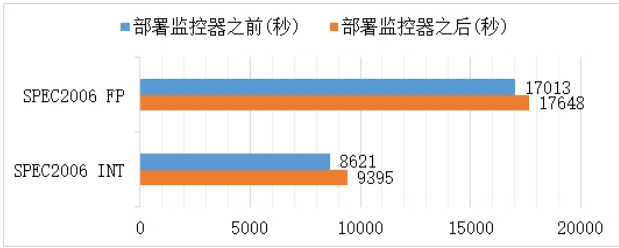

图 15 SPEC2006 执行时间对比图

6 相关工作

6.1 攻击检测技术

当前的 APT 攻击检测技术主要分为三类, 分别为基于日志审计的攻击检测[2–5], 基于二进制插装的攻击检测[6–10]以及两者的混合检测技术[11]. 基于日志审计的攻击检查通过记录系统级的事件, 例如文件 IO 等, 并利用系统中各程序运行时的日志记录, 分析并判定与攻击相关的事件序列.

基于日志审计的攻击检测方法主要由两类, 一类是基于异常检测的方法, 另一类为基于攻击特征的方法. 基于异常检测的方法常通过提取分析系统正常行为日志中的模式, 来识别偏移这些模型的异常行为. 例如文章 Nodoze[35]首先通过处理日志文件构建成 APT 攻击溯源图（Provenance Graph）, 随后分析溯源图中程序之间的信息流, 即上一个程序的输出作为当前程序的输入. 根据这些程序的输入输出依赖关系获得其异常分, 以此来识别异常分高的异常行为. 此外 Unicorn[32]和 ProvDetector[36]也通过学习系统中各良性程序正常执行的溯源图或者执行路径来识别 APT 攻击的异常行为. 国内也有一些相关工作, 例如 SeqNet[37], 通过将系统运行溯源图序列转化为特征向量序列, 并训练得到系统的良性行为模型, 以有效识别攻击过程中的长期关联关系.

基于攻击特征的方法, 例如 Holmes[5] 提出可引入专家知识, 通过事先制定的依赖规则将高层 TTP（Tactics, Techniques, and Procedures, 技术、战术和过程）与底层零散事件进行映射, 从而实现攻击检测. 类似的, ATLAS[3]认为攻击之间通常会共享一些相同的攻击行为, 因此可通过识别提取攻击序列, 并利用相似度来识别 APT 攻击. 此外, 国内也有工作提出可利用沙盒提取 APT 攻击的进程行为, 注册表行为, 网络行为等特征, 基于提取到的动态特征与目标程序的 PE 头静态特征相结合, 可进一步进行高效的攻击检测以及组织分类[38].

基于日志的攻击检测方法仅能通过分析通用系统事件来分析攻击行为, 因此, 随着 APT 攻击的隐蔽性日益增强, 基于日志的攻击检测方法由于无法感知程序内部信息而产生语义鸿沟问题[9], 从而造成漏报率的上升. 例如攻击程序可以恶意注入良性程序, 以劫持良性程序进行一些非法操作, 从而避免被发现, 该场景下若能结合良性程序的内部信息, 将使得攻击检测更精准.

基于二进制插装技术的攻击检测方法则通过对程序内部进行插装, 以获得更细粒度以及更精确的程序运行记录. 例如 libdft[38]提出基于 Intel Pin 二进制动态插装技术的动态数据流追踪系统, 可对大型二进制程序进行动态插装, 并追踪其数据以应对零日攻击等更具威胁性的攻击. 类似的, DTA++[39]以及 Minemu[40]采用经过优化的动态污点追踪技术来精确检测内存相关的攻击.

二进制插装技术不受编程语言限制, 较为通用, 然而过多的插装, 将使得目标程序的运行效率大大下降. 目前也有工作尝试将日志信息与程序内部信息相结合, 降低插装需求并进行协同工作, 以达到攻击检测的最佳效果. 例如 BEEP[6]通过检测程序中的循环, 并以循环为单位进行插装. 插装所获取的程序信息可与系统日志结合, 提高攻击检测的精度. 相较于先前以指令为单位的插装方法, BEEP 以循环为单位的插装方法可以有效降低插装带来的性能损耗. TRACE[11]又进一步, 通过共享数据结构识别及动态数据流追踪, 可以更精准地保留所需的程序内部信息, 从而降低攻击检测过程中的性能与存储开销. 此外, OmegaLog[9]认为可以仅对软件自带的日志函数进行插装并提取日志信息, 与系统日志进行融合分析从而达到攻击检测的目的. 该方案确实可以进一步降低插装带来的性能损耗, 但也存在一定的局限性, 例如目前大部分软件的日志是功能性导向的, 主要用于排查故障, 可能不会记录攻击检测所需要的安全相关的事件.

6.2 攻击模型

攻击模型常用于描述单个或者多个攻击, 用以评估系统的脆弱性或者攻击的危害性. 常见的攻击模型主要有攻击树(Attack Tree), 攻击图(Attack Graph).

攻击树是由根节点与众多子节点组成的多层次树结构[41], 每一个节点均表示一个攻击行为. 攻击树自下而上, 具有同一父节点的兄弟节点之间存在“与”, “或”关系. 具有“与”关系的子节点必须同时满足才可触发父节点, 具有“或”关系的子节点之间则只需满足其一即可触发父节点. 如图 16 所示, 如果攻击者想要

非法登录目标邮件系统, 则需要通过暴力破解或者通过密码窃取的方式来获取密码. 而若想要进行密码窃取, 则还需要同时安装键盘记录器并且监控用户窗口以准确地获取目标邮件系统密码.

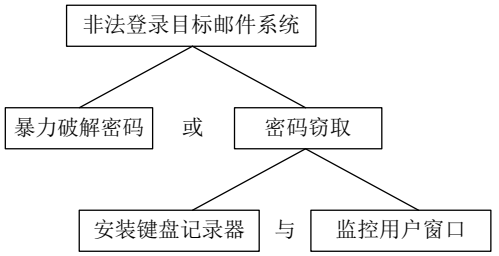

图 16 非法窃取目标邮件系统的攻击树小例

随着网络攻击的发展, 用于分析计算机网络脆弱性的攻击图也应运而生. 它可以生成攻击者在网络中的所有攻击路径, 这使得每个节点对系统整体安全的重要性更加明确, 大大地补充了原先只能够依靠漏洞的数量和评级对整个网络进行安全性分析的方法. 攻击图主要分为状态攻击图与属性攻击图, 状态攻击图[42–45]中的节点表示节点编号, 端口情况, 权限等系统状态, 有向边则主要表示攻击行为. 状态攻击图可以表示当攻击者实施某攻击后, 系统的整体状态将如何发生变迁. 状态攻击图能够十分简洁直观地描述攻击者在目标网络系统中的所有潜在的攻击路径, 便于安全人员分析网络中的弱点. 但是状态攻击图的每一个结点都需要表示系统全局状态, 因此一些本身不参与攻击的结点的状态将成为攻击图中的冗余信息, 这使得状态攻击图不适用于节点众多的网络中, 具有较大的局限性.

为了适应日益扩大的网络规模, 属性攻击图被提出. 属性攻击图[46,47]的节点可分别表示漏洞或属性, 属性节点表明攻击者当前所具有的权限, 漏洞节点表示存在漏洞的服务. 漏洞节点的入度为条件边, 当满足条件时, 该节点所表示的漏洞才能被利用. 属性攻击图由于只涉及与攻击有关的属性变化, 因此相对于表示系统全局状态变化过程的状态攻击图要简洁, 但是由于属性攻击图只显示必要的属性, 因此不如状态攻击图直观, 难以轻松理解. 针对不同的应用场景, 属性图也可进行扩展, 例如也有人将攻击图与贝叶斯图结合, 实现贝叶斯攻击图, 这类攻击图能够用于计算攻击者攻击成功的概率[48]. 甚至也有研究将其移植到处理器微架构, 用于表示微架构层面的攻击[49].

7 讨论

7.1 攻击变种检测

如前文所述，为了在检测变种的同时保证攻击检测的精确度, 现有的基于事件分析的攻击检测方案通常同时使用两类攻击检测技术. 一类是基于攻击信号识别的攻击检测技术, 通过提取和识别攻击特征来发现攻击. 另一类是基于异常行为定位的攻击检测技术, 该方案通过定位异常活动来发现攻击. 而本文根据观察发现, 攻击者通常会采用不同的攻击机制实现同一攻击目标. 因此本文方法利用攻击后对系统状态的影响, 而非这些层出不穷的攻击机制, 作为检测指标来实现攻击检测. 不同攻击检测方法对攻击变种检测的效果主要受攻击检测方法本身及其所针对的目标影响, 因此本小节将从这两方面来讨论攻击变种检测问题.

1. **攻击检测方法的泛化性**: 基于攻击信号识别的攻击检测技术通过将观察到的运行数据与已有的攻击模式库进行匹配以实现攻击检测, 因此可以很好地拟合已知攻击, 然而由于攻击模式库需要手动补充, 因为对未知攻击的泛化性不足. 而基于异常行为定位的攻击检测技术则通过学习系统良性行为来检测攻击造成的异常, 以弥补基于攻击信号识别的攻击检测技术对未知攻击泛化性不足的问题. 然而, 来自单一来源的数据输入缺乏攻击导向性, 容易使训练得到的模型与系统中频繁出现的良性行为过拟合, 从而导致对一些低频次的用户操作产生误报. 本文采用的方法则结合了上述两种方法

的优点, 利用不同攻击手段的攻击目标的相似性, 采用其对系统造成的影响这一更具通用性的指标来构建攻击模型, 一方面缓解了攻击检测方法对已知攻击模式的过拟合问题, 提升了模型的泛化性以实现对攻击变种的识别和感知, 另一方面基于模型的指导, 降低了攻击检测的误报率.

2. **针对攻击行为的检测 VS 针对攻击目标的检测:** 基于攻击信号识别的攻击检测技术以及基于异常行为定位的攻击检测技术分别通过识别收集的信息中是否存在攻击信号或者异常行为以进行攻击检测. 上述两种检测方法主要关注于攻击行为, 然而攻击手段变化多样, 很难完全覆盖. 本文通过观察发现, 一方面系统中需要重点关注与保护的对象有限, 另一方面, 虽然攻击与其变种日益增多, 但它们所利用的缺陷以及利用后对系统的影响并没有随之大量增长. 因此相比于多变的攻击手段, 本文方法更关注数量相对有限的攻击目标, 即通过攻击后对目标系统及软件造成的影响, 来进行攻击检测. 本文提出指标依赖模型, 并根据模型指导在目标系统与软件中有针对性地插入安全导向的监控器以使得威胁感知能力尽可能覆盖关键资源（系统部件, 关键数据, 重要软件等）. 这些关键资源通常是攻击过程中不可避免的目标, 因此即便出现新的攻击手段以及攻击场景, 也依然可以被本文所提出的攻击检测方法所感知.

需要注意的是, 为了实现上述检测效果, 本文的攻击检测方法需要在目标系统及软件中进行插装使用. 由于其不影响现有的防御体系, 同时具有可接受的性能损耗, 因此也可以与已有的防御措施协同工作, 从而更全面地保护系统.

7.2 攻击建模方案

为适用于不同的场景, 很多攻击模型方案均已提出, 例如攻击树, 攻击图等. 这些攻击模型主要用于可视化网络中存在的攻击路径, 以帮助安全人员分析当前网络的脆弱性. 然而上述的这些模型, 多需要手动输入大量数据, 例如网络中每台机器中存在的漏洞, 网络架构等等. 此外现有的攻击模型仅包含粗粒度的信息, 没有详细的指标, 因此难以用于攻击检测.

本文提出的指标依赖模型则是专为攻击检测设计的, 然而一方面, 当前的 CVE 漏洞库中关于漏洞的描述信息过于简单, 缺乏细节, 且其它来源的资料较为杂乱, 没有统一的格式, 因而难以从中提取有效信息用于明确监控漏洞利用所需的预警指标以及关键指标. 另一方面, 由于不同系统以及软件的异常指标不同（例如同一程序中不同变量的正常取值范围, 不同程序对函数执行时间的容忍程度等均不同）, 异常的指标阈值难以通过自动化分析等方式获取. 综上, 我们很难利用自动化的方式为漏洞利用及攻击提取到对应的指标依赖模型, 当前工作中, 漏洞利用的模型以及多个漏洞利用组合的攻击模型依然需要手动构建. 为了降低人工成本, 后续也将探索自动化建模方法. 后续建模方法将尝试从 CVE 漏洞库以及网络威胁情报等已有的知识体系上, 进一步细粒度分析, 挖掘, 并融合成新的攻击知识图谱. 基于该知识图谱, 一方面可以根据网络威胁情报提供的详细攻击信息有效提取到检测相关的指标, 以半自动化或者部分场景自动化的方式组建攻击检测所需的指标依赖模型. 另一方面, 可以根据漏洞利用及其在网络攻击中的上下文提取其对应的攻击逻辑, 以指导自动化组合漏洞利用的指标依赖模型, 从而降低组合指标依赖模型所需要搜索的状态空间, 有效避免路径爆炸问题.

8 总 结

基于漏洞的重复性和相似性, 本文提出了一种新的指标依赖模型, 并应用于攻击检测. 该方法着眼于漏洞的根因以及利用后对系统状态的影响而非变化多样的攻击行为, 构建通用且具有高泛化性的指标依赖模型. 基于模型的关键指标指导进行监控, 可实现攻击与变种的精确检测. 实验表明, 本文提出的方法在攻击及变种检测中具有较好的效果, 优于现有的方案, 并且具有更少的误报以及可接受的性能损耗.

References:

[1] FIREEYE M. 2021 FIREEYE MANDIANT Service Special Report. 2022.

https://www.arrow.com/ecs-media/16352/fireeye-rpt-mtrends-2021.pdf.

[2] Yagemann C, Pruett M, Chung S P, et al. ARCUS: Symbolic Root Cause Analysis of Exploits in Production Systems. In: Proceedings of the 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, 2021. 1989-2006.

[3] Alsaheel A, Nan Y, Ma S, et al. ATLAS: A sequence-based learning approach for attack investigation. In: Proceedings of the 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, 2021. 3005-3022.

[4] Yu L, Ma S, Zhang Z, et al. ALchemist: Fusing Application and Audit Logs for Precise Attack Provenance without Instrumentation. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 21). The Internet Society, 2021.

[5] Milajerdi S M, Gjomemo R, Eshete B, et al. Holmes: real-time apt detection through correlation of suspicious information flows. In: Proceedings of the IEEE Symposium on Security and Privacy (SP). IEEE, 2019. 1137-1152.

[6] Lee K H, Zhang X, Xu D. High Accuracy Attack Provenance via Binary-based Execution Partition. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 13). The Internet Society, 2013.

[7] Kwon Y, Wang F, Wang W, et al. MCI: Modeling-based Causality Inference in Audit Logging for Attack Investigation. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 18). The Internet Society, 2018.

[8] Ma S, Zhai J, Wang F, et al. MPI: Multiple perspective attack investigation with semantic aware execution partitioning. In: Proceedings of the 26th USENIX Security Symposium (USENIX Security 17). USENIX Association, 2017. 1111-1128.

[9] Hassan W U, Noureddine M A, Datta P, et al. OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 20). The Internet Society, 2020.

[10] Ma S, Zhang X, Xu D. Protracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 16). The Internet Society, 2016.

[11] Irshad H, Ciocarlie G, Gehani A, et al. Trace: Enterprise-wide provenance tracking for real-time apt detection. IEEE Transactions on Information Forensics and Security, 2021, 16: 4363-4376.

[12] Elastic. Open security platform unifying SIEM, endpoint & cloud | Elastic. 2022. https://www.elastic.co/security.

[13] Hutchins E M, Cloppert M J, Amin R M. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues in Information Warfare & Security Research, 2011, 1: 80-106.

[14] DARPA. Transparent Computing. 2022. https://www.darpa.mil/program/transparent-computing.

[15] Canella C, Van Bulck J, Schwarz M, et al. A systematic evaluation of transient execution attacks and defenses. In: Proceedings of the 28th USENIX Security Symposium (USENIX Security 19). USENIX Association, 2019. 249-266.

[16] Wang T, Wei T, Gu G, et al. TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection. In: Proceedings of the IEEE Symposium on Security and Privacy. IEEE, 2010. 497-512.

[17] Lin G, Wen S, Han Q L, et al. Software vulnerability detection using deep neural networks: a survey. Proceedings of the IEEE, 2020, 108(10): 1825-1848.

[18] Farris K A, Shah A, Cybenko G, et al. Vulcon: A system for vulnerability prioritization, mitigation, and management. ACM Transactions on Privacy and Security (TOPS), 2018, 21(4): 1-28.

[19] Duan R, Alrawi O, Kasturi R P, et al. Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 21). The Internet Society, 2021.

[20] Syed N F, Shah S W, Trujillo-Rasua R, et al. Traceability in supply chains: A Cyber security analysis. Computers & Security, 2022, 112: 102536.

[21] MITRE. CWE - Detection Methods. 2022. https://cwe.mitre.org/community/swa/detection_methods.html.

[22] MITRE. MITRE ATT&CK®. 2022. https://attack.mitre.org/.

[23] Su T, Wang J, Su Z. Benchmarking automated GUI testing for Android against real-world bugs. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2021. 119-130.

[24] MURAUS T. Detecting which process is creating a file using LD_PRELOAD trick. 2022.

https://www.tomaz.me/2014/01/08/detecting-which-process-is-creating-a-file-using-ld-preload-trick.html.

[25] Sysdig. Security Tools for Containers, Kubernetes, & Cloud. 2022. https://sysdig.com/.

[26] Everson D, Cheng L, Zhang Z. Log4shell: Redefining the Web Attack Surface. In: Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb). The Internet Society, 2022.

[27] Darktrace. Detecting and responding to Log4Shell in the wild. 2022. https://www.darktrace.com/en/blog/detecting-and-responding-to-log-4-shell-in-the-wild.

[28] Synopsys. Heartbleed Bug. 2022. https://heartbleed.com/.

[29] Elastic. Detecting Exploitation of CVE-2021-44228 (log4j2) with Elastic Security. 2022. https://www.elastic.co/cn/security-labs/detecting-log4j2-with-elastic-security.

[30] Hossain M N, Milajerdi S M, Wang J, et al. SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In: Proceedings of the 26th USENIX Security Symposium (USENIX Security 17). USENIX Association, 2017. 487-504.

[31] Milajerdi S M, Eshete B, Gjomemo R, et al. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS 2019). ACM, 2019. 1795-1812.

[32] Han X, Pasquier T, Bates A, et al. UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats. In: Proceedings of the Network and Distributed Systems Security (NDSS 2020) Symposium 2020. The Internet Society, 2020.

[33] Pei K, Gu Z, Saltaformaggio B, et al. Hercule: Attack story reconstruction via community discovery on correlated log graph. In: Proceedings of the 32Nd Annual Conference on Computer Security Applications. ACM, 2016. 583-595.

[34] SPEC. SPEC CPU® 2006. 2022. https://www.spec.org/cpu2006/.

[35] Hassan W U, Guo S, Li D, et al. Nodoze: Combatting threat alert fatigue with automated provenance triage. In: Proceedings of the Network and Distributed Systems Security (NDSS 2019). The Internet Society, 2019.

[36] Wang Q, Hassan W U, Li D, et al. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In: Proceedings of the Network and Distributed Systems Security (NDSS 20). The Internet Society, 2020.

[37] Liang R Z, Gao Y, Zhao X B. Sequence feature extraction-based APT attack detection method with provenance graphs (in Chinese). Sci Sin Inform, 2022, 52: 1463–1480.

[38] Liang H, Li X, Yin N, et al. APT attack detection method combining dynamic behavior and static characteristics (in Chinese). Computer Engineering and Application. 2022.

[39] Kemerlis V P, Portokalidis G, Jee K, et al. libdft: Practical dynamic data flow tracking for commodity systems. In: Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments. ACM, 2012. 121-132.

[40] Kang M G, McCamant S, Poosankam P, et al. Dta++: dynamic taint analysis with targeted control-flow propagation. In: Proceedings of the Network and Distributed Systems Security (NDSS 11). The Internet Society, 2011.

[41] Bosman E, Slowinska A, Bos H. Minemu: The world's fastest taint tracker. In: Proceedings of the International Workshop on Recent Advances in Intrusion Detection. Springer, 2011. 1-20.

[42] Schneier B. Attack trees. Dr. Dobb's journal, 1999, 24(12): 21-29.

[43] Wing J M. Scenario graphs applied to network security. Information Assurance: Survivability and Security in Networked Systems, 2008: 247-277.

[44] Sheyner O M. Scenario graphs and attack graphs [Ph.D. Thesis]. Pittsburgh, Pennsylvania: Carnegie Mellon University, 2004.

[45] Sheyner O, Wing J. Tools for generating and analyzing attack graphs. In: Proceedings of the International Symposium on Formal Methods for Components and Objects. Springer, 2003. 344-371.

[46] Jha S, Sheyner O, Wing J. Two formal analyses of attack graphs. In: Proceedings of the 15th IEEE Computer Security Foundations Workshop. IEEE, 2002. 49-63.

[47] Bhattacharya S, Ghosh S K. An artificial intelligence based approach for risk management using attack graph. In: Proceedings of the

2007 International Conference on Computational Intelligence and Security (CIS 2007). IEEE, 2007. 794-798.

[48] Wang L, Yao C, Singhal A, et al. Interactive analysis of attack graphs using relational queries. In: Proceedings of the IFIP Annual Conference on Data and Applications Security and Privacy. Springer, 2006. 119-132.

[49] Wang L, Zhu Z, Wang Z, et al. Analyzing the security of the cache side channel defences with attack graphs. In: Proceedings of the 25th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2020. 50-55.

附中文参考文献:

[37] 梁若舟, 高跃, 赵曦滨. 基于序列特征提取的溯源图上 APT 攻击检测方法. 中国科学: 信息科学, 2022, 52: 1463–1480.

[38] 梁鹤，李鑫，尹南南，李超．结合动态行为和静态特征的 APT 攻击检测方法．计算机工程与应用, 2022.