软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.006973]

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

一种基于威胁模型的安全测试用例生成框架和工具*

付昌兰[1,2], 张 贺[1,2], 李凤龙[3], 匡宏宇[1,2]

[1](南京大学 软件学院, 江苏 南京 210023)

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

[3](华为云计算技术有限公司, 浙江 杭州 310053)

通信作者: 张贺, E-mail: hezhang@nju.edu.cn

摘 要: 近年来, 软件系统安全问题正引发越来越多的关注, 系统存在的安全威胁容易被攻击者所利用, 攻击者通常采用各种攻击技术诸如口令暴力破解、网络钓鱼、SQL 注入等对系统进行攻击. 威胁建模是一种结构化分析、识别并处理威胁的方法, 传统的测试主要集中在测试代码缺陷, 处于软件开发后期, 不能很好地对接前期威胁建模分析成果以构建安全的软件, 业界威胁建模工具缺少进一步生成安全测试的功能. 为了应对此问题, 提出一种从威胁模型生成安全测试用例的框架, 并设计和实现工具原型. 为了便于测试, 对传统的攻击树模型进行改进, 对构建的模型进行规范性检查, 从该模型中可以自动生成测试线索. 根据攻击节点发生概率对测试线索进行评估, 优先检测概率较高的威胁的测试线索. 对防御节点进行评估, 选择收益性较高的防御方案缓解威胁, 以改进系统安全设计. 通过为攻击节点设置参数可以将测试线索转换成具体的测试用例. 在软件开发早期阶段以威胁建模识别出的威胁作为输入, 通过框架和工具可以生成测试, 指导后续的安全开发和安全测试设计, 将安全技术更好地嵌入到软件设计和开发之中. 案例研究部分将该框架和工具运用于极高危风险的安全测试生成, 并说明了其有效性.

关键词: 威胁模型; 威胁建模; 攻击树模型; 测试用例生成; 测试线索

中图法分类号: TP311

中文引用格式: 付昌兰, 张贺, 李凤龙, 匡宏宇. 一种基于威胁模型的安全测试用例生成框架和工具. 软件学报. http://www.jos.org.cn/1000-9825/6973.htm

英文引用格式: Fu CL, Zhang H, Li FL, Kuang HY. Threat Model-based Security Test Case Generation Framework and Tool. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/6973.htm

Threat Model-based Security Test Case Generation Framework and Tool

FU Chang-Lan[1,2], ZHANG He[1,2], LI Feng-Long[3], KUANG Hong-Yu[1,2]

[1](Software Institute, Nanjing University, Nanjing 210023, China)

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

[3](Huawei Cloud Computing Technologies Co. Ltd., Hangzhou 310053, China)

Abstract: In recent years, software system security issues are attracting increasing attention. The security threats existing in systems can be easily exploited by attackers. Attackers usually attack systems by using various attacking techniques, such as password brute force cracking, phishing, and SQL injection. Threat modeling is a method of structurally analyzing, identifying, and processing threats. Traditional tests mainly focus on testing code defects, which take place in the late stage of software development. It is not able to well connect the results from early threat modeling and analysis for building secure software. Threat modeling tools in the industry lack the function of generating security tests. In order to tackle this problem, this study proposes a framework that is able to generate security test

* 基金项目: CCF-华为胡杨林基金-软件工程专项 (CCF-HuaweiSE2021003); 国家自然科学基金 (62072227, 62202219); 国家重点研发计划 (2019YFE0105500); 江苏省重点研发计划 (BE2021002-2); 南京大学计算机软件新技术国家重点实验室创新项目 (ZZKT2022A25); 海外开放课题 (KFKT2022A09)

收稿时间: 2022-08-28; 修改时间: 2022-10-26, 2023-02-17, 2023-03-16; 采用时间: 2023-05-23; jos 在线出版时间: 2023-09-27

cases from threat models and designs and implements a tool prototype. In order to facilitate tests, this study improves the traditional attack tree model and performs compliance checks. Test scenarios can be automatically generated from the model. The test scenarios are evaluated according to the probabilities of attack nodes, and the scenarios of the threats with higher probabilities will be tested first. The defense nodes are evaluated, and the defense scheme with higher profit is selected to alleviate the threats, so as to improve the system's security design. By setting parameters for attack nodes, test scenarios can be specified as test cases. In the early stage of software development, with the inputs of the threats identified by threat modeling, test cases can be generated through this framework and tool to guide subsequent security development and test design, which improves the integration of security technology in software design and development. The case study applies this framework and tool in test generation for very high security risks, which shows their effectiveness.

Key words: threat model; threat modeling; attack tree model; test case generation; test scenarios

随着计算机与互联网技术的快速发展, 软件已经渗入到人们生活、工作的各个方面. 与此同时, 互联网软件攻击技术也在不断演进, 软件面临着极大的安全问题. Web 是互联网的核心, 是云计算和移动互联网的最佳载体, 因此 Web 安全也是互联网公司安全业务中最重要的组成部分[1]. 早在 2003 年, Web 技术的成熟使得 Web 应用的功能越来越强大, 最终成为了互联网的主流, 成为黑客攻击目标. SQL 注入的出现是 Web 安全史上的一个里程碑, 它最早出现大概是在 1999 年, 并很快就成为 Web 安全的头号大敌. SQL 注入漏洞至今仍然是 Web 安全领域中的一个重要组成部分. Web 安全领域常见的攻击技术还有跨站脚本攻击 (XSS)、跨站点请求伪造 (CSRF) 等攻击技术. 各种攻击技术主要针对软件存在的威胁和漏洞, 软件缺陷容易被攻击者所利用, 给人们的生命财产安全造成了极大的损失. 近年来, 软件安全越来越受到人们的重视, 被学术界和工业界所广泛关注[2].

美国国家标准与技术研究院 (NIST) 报告软件安全缺陷修复成本巨大, 当应用程序部署并进入其运行环境时, 要显著提高其安全性是非常困难和昂贵的[3]. 如果是在项目发布后再执行漏洞修复计划, 其修复成本相当于在设计阶段执行修复的 30 倍[4]. 软件安全问题需要在软件开发过程的早期阶段就被考虑, 在早期发现安全威胁有助于从源处解决安全问题, 减轻安全威胁发生所带来的损失和修复成本.

威胁建模 (threat modeling)[5]是一种通过对系统进行建模和结构化分析, 识别系统潜在的威胁和漏洞, 并正确应对这些威胁和漏洞从而保证系统安全的有效方法. 威胁建模作为安全开发生命周期 (SDL) 中的重要一环, 具有重要的价值, 包括早期发现安全漏洞, 确定安全需求, 设计和交付更安全的产品等. 威胁建模特别提供了一种系统的方法来识别业务场景中可能危及安全的威胁, 而从威胁建模生成测试有助于提供安全测试的设计和执行的指导, 检测和缓解威胁, 尽早应对安全问题, 将安全嵌入到软件设计和实现中, 提高构建软件的安全性. 基于威胁建模的测试设计是被业界认可采用的手段, 威胁建模的分析结果作为测试需求提供给测试人员进行安全测试. 传统的测试主要集中在代码测试, 处于软件开发后期, 不能很好地对接前期威胁建模分析成果以构建安全的软件. 现有的威胁建模技术与软件测试的结合非常有限, 业界已有的威胁建模工具均不具备进一步生成测试的功能, 难以有效指导后续开发和测试设计. 目前业界主要是人工手动设计测试方案和用例, 安全测试设计专业知识要求高, 往往需要专业测试人员联合开发人员及其他人员一起设计, 急需从威胁建模生成测试, 指导和辅助安全测试设计, 提升安全测试设计的全面性和效率.

攻击树模型[5]是一种以树型结构描述攻击步骤的威胁模型, 根节点即为攻击目标, 可以对接威胁建模识别的威胁作为根节点, 通过该模型可以直观和全面地从黑客视角分析达成该目标威胁的攻击途径. 根据达成目标威胁的攻击途径检测威胁, 可以测试威胁是否存在. 但是传统的攻击树模型较为抽象, 存在一定局限性, 现有研究从原始攻击树生成测试较为抽象, 没有考虑如下几个方面: 如何用于实际场景中具体安全测试, 如何实际对接到威胁建模, 提供从威胁建模到具体测试的完整流程和工具, 攻击途径较多时如何高效检测到威胁, 针对性地防御和缓解威胁. 因此, 难以运用于工业界进行安全测试并改进安全设计.

针对上述问题, 本文基于攻击树模型进行优化设计, 定义改进的攻击树模型, 基于该模型提出了从威胁模型生成测试用例的框架, 并实现了工具原型. 本文的研究概括图如图 1 所示. 在软件开发早期阶段以威胁建模识别出的威胁作为输入, 通过模型构建对接威胁, 对构建的模型进行规范性检查, 提供详细信息, 具体贴合实际场景测试, 通过本框架和工具可以生成具体的测试用例, 通过对生成的测试评估优先级以提高测试效率, 检测威胁的同时防御

和缓解威胁, 指导后续的安全开发和测试设计, 将安全技术更好地嵌入到软件设计和开发之中. 本文所构建的数据集范围主要是 Web 安全领域, 本工具提供了 Web 安全领域常见威胁的测试数据, 可以直接用于分析 Web 安全威胁测试生成, 检测威胁, 提高软件安全性.

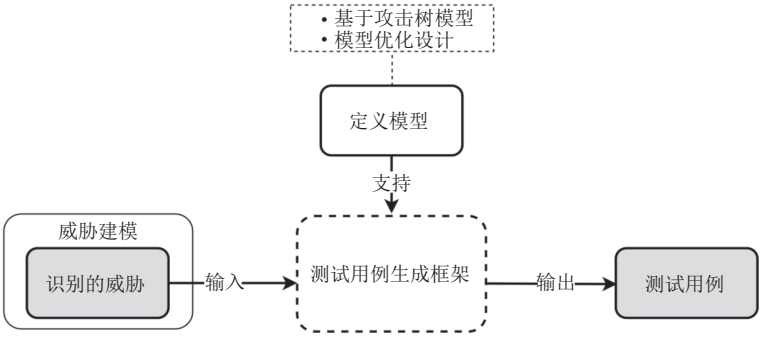


图 1　测试用例生成整体研究概括图

本文的主要贡献如下.

1) 提出一种基于威胁模型的安全测试用例生成框架. 该框架可以对接前期威胁建模分析结果, 具备从威胁生成测试用例的完整流程, 并提供了 Web 安全领域常见威胁的测试数据, 辅助安全测试.

2) 提出一种改进的攻击树模型, 该模型支撑测试用例生成框架以及框架中的元素, 更贴合实际场景中的测试.

3) 设计和实现了工具原型, 帮助企业和其他组织进行威胁分析、安全开发和测试设计. 通过对被业界广泛关注的极高危风险 log4j2 进行案例研究和分析, 验证了本框架和工具的有效性.

本文第 1 节介绍背景及相关工作. 第 2 节介绍了改进的攻击树模型. 第 3 节给出了基于威胁模型生成测试的具体框架和流程. 第 4 节实现了自动化工具原型. 第 5 节通过案例研究验证了本文所提出的框架和工具的有效性. 第 6 节总结全文并提出了下一步工作.

1　背景及相关工作

1.1　威胁建模背景

威胁建模是一种通过对系统进行建模和结构化分析, 识别系统潜在的威胁和漏洞, 并正确应对这些威胁和漏洞从而保证系统安全的有效方法. 威胁建模在系统整体安全模型的设计中起着重要作用, 因为它可以帮助确保将安全性内置到应用程序中, 早期发现安全缺陷, 理解安全需求, 设计和交付更安全的产品等. 威胁建模是安全开发生命周期 (SDL) 中重要的一环, 需要专业的安全知识和渗透测试视角, 从攻击者角度分析系统或特性的安全风险, 并给出恰当的处置, 其主要包括系统建模、识别威胁、缓解威胁、验证威胁被缓解这几个步骤. 传统的软件安全威胁建模和安全风险分析多依赖于分析人员安全技术水平, 技术门槛较高, 分析质量参差不齐. 目前已有一些威胁建模识别技术和工具, 用于提升威胁建模的质量, 降低技术门槛, 提高系统安全性.

STRIDE[5]方法是一种流行和实用的威胁建模技术, 该方法由 Microsoft 员工 Kohnfelder 等人[6]发明, 用于发现软件系统的安全缺陷. STRIDE 方法基于数据流图 (DFD) 对系统建模, DFD 比较适合对系统的实体、实体交互、数据流、资产/数据存储等各个方面进行表征, 结合信任边界, 更加全面直观地从威胁分析的角度对系统业务进行刻画, 满足业务需求. 该方法将威胁分为仿冒 (spoofing)、篡改 (tampering)、抵赖 (repudiation)、信息泄露 (information disclosure)、拒绝服务 (denial of service) 和权限提升 (elevation of privilege) 这 6 类, DFD 元素与 STRIDE 这 6 类威胁之间存在映射关系. STRIDE 可用于通过分析 DFD 中定义的每个接口来识别威胁, 此框架和助记符旨在帮助软件开发人员识别软件容易遭受的攻击的类型[5]. 微软已将安全性很好地集成到安全开发生命周期 (SDL) 中[7], 威胁建模是 Microsoft 安全开发生命周期的核心元素, SDL 使用 STRIDE 集成了系统的安全威胁建模方法. 微软的 MTMT (Microsoft threat modeling tool) (https://www.microsoft.com/en-us/securityengineering/

sdl/threatmodeling) 工具基于 STRIDE 方法进行威胁识别. 相较于其他工具如 Threat Dragon (https://github.com/OWASP/threat-dragon)、IriusRisk (https://www.iriusrisk.com/threat-modeling-platform)、ThreatModeler (https://threatmodeler.com/) 而言, 微软的 MTMT 工具威胁分析质量以及自动化程度会高一些. 目前已有的工具主要侧重于识别系统潜在的威胁, 缺少对威胁发生的原因、条件和场景进行具体分析, 均不具备进一步生成测试的功能, 不能很好地指导从根本上解决安全问题.

1.2 基于威胁模型的测试生成

目前可以用于威胁建模生成测试的模型主要有 3 种, 分别是基于树、基于网、基于图的模型. 基于树的威胁模型, 主要分为故障树[8]、攻击树[9], 故障树通过对可能造成系统故障的各方面因素进行分析, 画出故障原因的各种可能组合和发生概率, 主要用于分析大型复杂系统的可靠性. 攻击树 (attack trees)[9]类似于故障树[8], 是用树状结构来表示攻击者的攻击路径和手段. 根节点描述了高级攻击, 是攻击目标, 该高级攻击在较低级别的攻击分支中进一步分解为攻击措施. 通过对造成威胁目标的所有攻击措施进行分析, 可以分析触发威胁目标的所有攻击路径. Wysopal 等人[10]使用攻击树进行基于风险的安全测试, 从开源库漏洞树中学习知识, 通过威胁建模分析风险行为, 基于数据流图识别威胁路径, 基于高风险的组件查找存在组件中的漏洞, 但是并没有从攻击树系统地生成安全测试. Marback 等人[11]基于原始攻击树模型生成安全测试序列, 给出了测试序列生成算法, 简要概括了实现的测试生成技术的思路, 没有提供实现细节, 其思路为将测试序列中的事件映射到适用的单元测试, 将指定的输入参数添加到测试序列中, 生成包括有效和无效输入的测试输入, 以及生成执行具有所分配的输入值的所有测试序列的测试脚本. 其生成的测试序列较为抽象, 未考虑实际场景中威胁攻击的具体信息, 包括攻击阶段、前置条件、攻击事件发生概率等, 未实际对接到威胁建模识别的威胁, 未提供数据支持以及完整的从威胁建模识别出的威胁到具体测试用例生成的完整框架和自动化工具, 难以运用于实际场景的安全实践.

基于网的威胁模型主要有基于 Petri 网[12,13]的威胁网, 具有数学和图形两种表达方式, 是一种网状信息流模型, 包括条件和事件两类节点, 其中可添加状态信息, 可以用于描述和分析系统中的控制流和信息流, 适合于描述异步的、并发的计算机系统模型. 以网状结构来建模和分析系统威胁, 验证系统行为的安全性是否满足安全目标. 威胁网的复杂性较高, 使用威胁网构建威胁模型需要形式化方法方面的专业知识. Xu 等人[12]基于 Petri 网提出了一种通过使用表示为 Predicate/Transition 网络的形式化威胁模型来自动生成安全测试的方法, 以网状结构对系统功能行为进行形式化建模, 分析系统功能行为中存在的安全威胁. 该方法根据威胁模型-实现描述 (TMID) 规范自动生成可执行的安全测试代码. TMID 规范由威胁模型和模型-实现映射 (MIM) 描述组成. 威胁模型描述了攻击者如何进行攻击以违反安全目标. MIM 描述将威胁模型的各个元素映射到其实现结构. 给定 TMID 规范, 该方法可以从威胁模型中生成所有攻击路径, 然后根据 MIM 描述将其转换为可执行代码.

基于图的威胁模型, 主要分为攻击图[14]、UML 时序图[15]. 攻击图是一种有向图, 展示了攻击者可能发动的攻击顺序和攻击效果, 由顶点和有向边两部分构成. 通过模拟攻击者对存在安全漏洞的网络攻击过程, 找到所有能够到达目标的攻击路径, 同时将这些路径以图的形式表现, 这种图就是网络攻击图, 简称攻击图. Malzahn 等人[14]介绍了一种用于识别和利用漏洞的端到端的过程和工具即自动漏洞和风险分析 (AVRA) 工具, 旨在用于网络风险评估. 所提出的方法可以同时分析整个网络, 比传统的漏洞扫描更全面. AVRA 自动生成网络及其各个组件的详细模型, 用于创建攻击图. 然后, AVRA 遵循攻击图中的某条攻击路径自动发起攻击以达到特定目标. UML 时序图描述了对象之间传递消息的时间顺序, 它用来表示用例中的行为顺序, 是强调消息时间顺序的交互图. UML 时序图描述了为了实现场景的行为而在对象之间交换的消息序列, 可以用于对威胁场景的事件序列进行建模, 发现系统运行时的威胁行为. Wang 等人[15]提出了一种威胁模型驱动的安全测试方法, 用于在运行时检测不良威胁行为. 对安全策略的威胁使用 UML 时序图建模. 从设计级威胁模型中提取一组威胁跟踪, 每个威胁跟踪都是在系统执行期间不应发生的事件序列. 如果执行跟踪是威胁跟踪的一个实例, 则会报告安全违规. 其关键技术是验证 UML 时序图中描述的威胁规范与代码实现之间的一致性, 将程序的执行视为方法调用和方法执行的事件序列, 如果任何事件序列在威胁模型中调用了可能的事件序列, 那么模型中的威胁仍然存在于代码中, 并且将生成错误消息以报告

故障. 经过对比分析, 这几种模型中攻击树模型更加直观且易于使用.

Hersén 等人[16]采用行为设计研究方法评估威胁建模语言模型中测试覆盖攻击模拟的程度, 用于分析测试系统内攻击者的行为以及评估测试覆盖率, 检查和确保所创建的威胁建模语言的质量. 该研究中的测试主要是用攻击模拟的方式检查语言的期望行为, 用于分析系统内的威胁, 而本文的测试对接威胁建模识别出来的威胁, 针对威胁进行测试, 用于指导后期的测试设计. Marksteiner 等人[17]对工业物联网 (IIoT) 提出了一种从威胁建模到自动化测试过程的方法, 将威胁建模和自动测试用例生成集成到工业化软件安全测试中, 以缩小威胁建模和自动化测试用例生成之间的差距. 所提出的方法旨在根据物联网系统中使用的协议生成交互序列, 指定一个协议消息命令列表, 并应用来自功能测试的测试生成器来创建交互序列. 生成序列的方法包括基于随机方法、基于搜索的策略、遗传算法和强化学习. 在线反馈定向测试用于确定可行序列, 并避免生成违反协议约束的非法序列. 但基于该研究发表的短文仅给出了大致的思路, 并没有提供详实的内容.

1.3 其他基于模型的测试生成

近几年基于威胁建模生成测试的研究相对较少, 此外还有一些基于模型生成测试、测试生成以及从早期阶段生成测试的相关研究.

Martin 等人[18−20]研究了从用 XACML (OASIS 可扩展访问控制标记语言) 编写的访问控制策略规范生成测试的技术. XACML 是用于编写访问控制策略的语言规范标准, 表示特定领域的访问控制策略、访问请求和访问响应, 运用软件测试技术发现基于 XACML 编写的策略规范中的错误, 确保策略规范的正确性. 他们定义了策略覆盖标准[18]和 XACML 访问控制策略[19]的变异测试框架. 为了从策略规范生成测试, 他们将输入整合到变化影响分析工具[20], 给定两个策略, 变化影响分析工具会对这两个策略的不同反应进行评估并给出反例, 根据这些反例生成请求, 用于检查策略规范的正确性. Masood 等人[21,22]研究了一种基于状态的方法来测试生成基于角色的访问控制 (RBAC) 策略. 他们的方法首先构建 RBAC 策略的有限状态模型, 然后从状态模型驱动测试. 通过状态模型对 RBAC 规范的结构和行为进行建模, 使用该模型为相应的实现生成测试套件, RBAC 规范的结构和行为代表实现的预期行为, 检查实现所表现的行为是否符合 RBAC 指定的预期行为, 针对策略规范中可能的约束/规则验证实现, 检查实现中的行为是否与访问控制规范一致. Zafar 等人[23]提出了一个用于嵌入式系统测试的三阶段工具支持的测试生成工作流, 该工作流从需求规范到测试. 工作流程中, 需求工程师基于 Gherkin-like 风格的领域特定语言 (DSL) 规范需求并提取关于模型元素即状态和转换的信息, 创建符合嵌入式系统行为方面的模型, 有助于对测试系统进行图形化建模. 工作流的后期阶段会生成一个可执行的测试脚本, 该脚本在特定领域的仿真平台上运行, 该工作对需求规格进行了测试. Qiu 等人[24]提出了一种基于形式化需求模型的测试用例生成方法, 以在需求和测试之间建立桥梁. 相对于从代码开始生成测试用例集来说, 该方法有助于分析软件是否满足需求设计. 通过形式化需求模型 (VRM) 对软件系统需求进行建模, 然后通过分析模型的输入和输出结构, 提取从输入变量到输出变量的约束路径, 以支持测试用例的错误类型分析. 通过自行开发的测试用例生成工具结合一个案例研究, 说明了所提出方法的有效性. Mihret 等人[25]提出了一种基于 PAT 模型检查器的协作系统测试用例生成方法, 该方法使用通信顺序过程 (CSP) 建模语言对选定协作系统的交互行为进行建模, 通过模型检查推理验证系统资产, 对特定和假定的攻击者行为进行建模, 并将其作为模型检查过程的输入模型, 生成攻击者驱动的反例 (违反系统模型指定属性的有序事件). 设计了测试规范规则/算法来将反例转换为测试用例, 转换旨在减少反例和相应测试用例之间的语义差距. Nayak 等人[26]提出了一种从输入 UML 活动图生成和优化测试序列的方法, 具体描述了一种称为测试序列生成统一建模语言 (UMLTSG) 的算法, 使用基于搜索的算法, 称为使用蚁群优化 (TSP ACO) 的测试序列优先化来生成和优化测试序列. 这些算法克服了处理复杂决策活动 (如条件活动、分叉活动和加入活动) 的现有限制. 优化过程有助于减少处理节点的数量, 从而使时间和成本最小化. Rocha 等人[27]提出了一种从 UML 模型生成测试的系统化过程, 该过程使用模型驱动工程 (MDE) 的概念, 将 UML 序列图形式化为扩展的有限状态机, 并为其提供精确的语义, 应用 ModelJUnit 和 JUnit 库来自动生成测试用例, 通过案例研究评估了其适用性. Santiago 等人[28]提出了一种基于机器学习 (ML) 的测试流生成方法, 该方法结合了语言规范, 其中包括可用于描述测试流的语法和可训练的测试流生成模型, 以便以可训练、可跨应用程序重用的方式生成测试, 以帮助弥合人类和机器测试能力之间的差

距. Erdem 等人[29]提出了一个电子商务工作流, 利用隐马尔可夫模型自动化 UI 测试. 所提出的工作流旨在基于用户浏览行为生成测试脚本, 能够生成在历史 Web 使用数据中看不到的浏览模式. 通过分析实际浏览行为和新生成的浏览行为之间的相似性评价新生成的测试脚本的质量. 该工作流能够生成与实际用户浏览行为具有高度相似性的新的和不可见的用户浏览行为. 所提供的原型实现能够生成高质量的新测试脚本, 可以提供对 Web 应用程序的全面测试. Grano 等人[30]设想了一组性能代理用于提供对测试执行成本即运行时和内存使用的合理估计, 提出了一种称为 aDynaMOSA 的自适应策略, 它通过扩展 DynaMOSA (单元测试中最先进的进化算法) 来利用这些代理, 与 DynaMOSA 相比, 通过此自适应方法生成的测试套件在运行时间和堆内存消耗方面获得统计意义上的显著改进. 以上研究中, 有安全测试生成的相关研究, 但不是威胁建模相关的测试生成, 有从早期需求阶段生成测试的研究以及基于模型生成测试的相关研究, 但不是关于安全测试生成, 其他测试生成相关的研究不是基于可视化模型生成的安全相关的测试, 没有对接到早期设计阶段威胁建模识别的威胁.

传统的测试主要集中在代码测试, 处于软件开发后期, 不能很好地对接前期威胁建模分析成果以构建安全的软件. 尽管有少量基于威胁模型生成测试的研究, 但没有提供从威胁生成测试用例的完整流程和框架, 缺少真正系统地生成安全测试并且可以实际落地于工业界的研究. 本文旨在基于威胁模型对识别出的威胁进一步生成测试, 通过生成的测试指导后续的安全开发和测试设计, 检测并缓解威胁, 将安全问题更好地更全面地嵌入到软件安全设计和实现之中, 提高软件的安全性.

2 改进的攻击树模型

本文所提方法主要基于攻击树模型来生成测试, 本节介绍了传统攻击树模型并对其进行分析, 设计了改进的攻击树模型, 并给出了模型定义.

2.1 传统攻击树模型分析

传统的攻击树模型[31]是由 Schneier 在 1999 年提出的一种分析威胁的形式化建模方法. 该模型提供了一种正式而条理清晰的方法来描述系统所面临的威胁和系统可能受到的多种攻击[32]. 该模型的构建思想基于从攻击者的角度出发, 将系统中存在的威胁描述为一系列可能的攻击操作. 其中, 树的根节点通常用来表示威胁, 叶子节点用来表示攻击者实施的具体攻击操作, 除根节点之外的非叶子结点通常用来表示多种攻击措施结合所达到的临时结果. 由于攻击措施结合方式不同, 攻击树一般被定义成与或树 (AND-OR Tree) 的形式.

如图 2 所示, 攻击树模型呈树型结构, 节点一般定义为三元组 TreeNode=(G, T, C). 其中, G 表示节点的攻击目标, T 表示节点的类型, C 表示节点的孩子列表. T 的类型有两种, 分别为 AND 类型和 OR 类型, AND 类型表示该节点可被划分为一系列串行的攻击操作, OR 类型表示该节点可被划分为一系列并行的攻击操作, 即实现 AND 类型节点的目标, 必须完成 C 中所有节点的操作, 而要想实现 OR 类型节点的目标, 只需完成 C 中的任一节点的操作即可. 一个攻击步骤可以与一个或多个后续攻击步骤连接, 以及根据攻击步骤之间的 OR 类型或 AND 类型关系, 从而创建一个攻击路径[33]. 图 2 所示模型中存在 4 条攻击路径, 分别是{G7, G8, G9, G5}, {G7, G8, G9, G6}, {G4, G5}, {G4, G6}, 路径中节点的执行顺序按照从左到右的顺序有序执行.

从图 2 中可以看出, 传统的攻击树模型的定义简单, 模型中的节点语义单一, 难以详细描述攻击操作的关键信息, 例如生命周期、层次等. 攻击操作存在一定的抽象性, 难以描述贴近业务场景的具体操作过程.

因此, 通过对传统攻击树模型进行分析, 总结如下需要改进的地方.

(1) 节点语义简单, 难以应用于实际场景. 攻击树模型中的节点仅是作为一种攻击操作的描述, 并没有揭露攻击者实施操作的具体信息. 其中, 攻击操作所需要的输入参数、输出参数、前置条件、后置条件、属性等相关信息并没有得到有效描述. 此外, 模型并未提供刻画攻击阶段的字段, 用户难以理解攻击操作处于攻击生命周期的哪一阶段, 不便于用于实际场景中具体的安全测试.

(2) 节点间的关联关系存在局限性. 在攻击树模型中, 节点有 AND、OR 两种关联类型, 当用户划分某一攻击操作时, 只能将操作定义为 AND 或 OR 其中的一种类型, 即节点关联的子节点列表间关联关系必须一致. 但在实

际应用中, 有些攻击操作可能被划分为若干不同关联关系的子节点, 即某一节点下的子节点中, 部分节点间是AND 关联关系, 部分节点间是 OR 关联关系. 如图 3 所示, 为实现 G0 的目标, 需要同时实现 G1 和 G2 或者单独实现 G3. 对于图 3 展示的关联场景, 原始攻击树模型定义的关联类型无法满足.

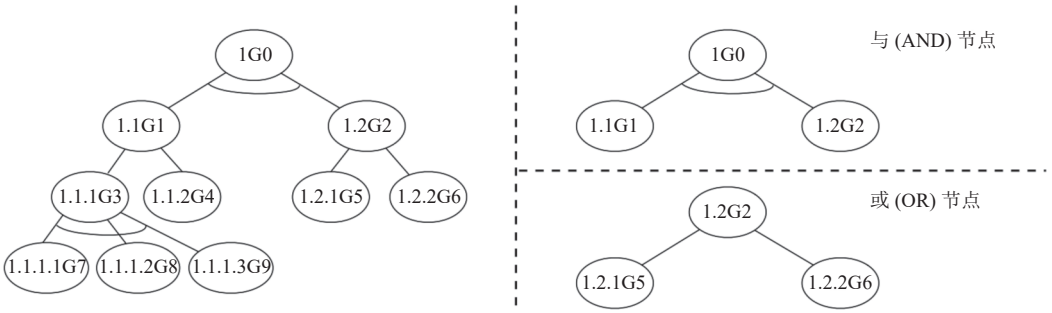


图 2　传统攻击树模型

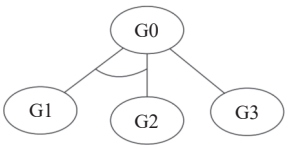


图 3　不同关联类型结合的节点类型

(3) 未体现消减措施. 攻击树模型仅对攻击操作进行简单描述, 并未体现应对这些攻击操作的消减措施. 在实际场景中, 用户通常对检测到的威胁同时具有对其进行消减的需求. 检测威胁的目的其实是为了发现威胁以及解决威胁, 消减措施可以反馈到安全设计和实现中去, 从而有助于避免威胁的发生, 构建安全的软件.

2.2　模型优化设计

针对传统攻击树模型中存在的局限性以及便于实际安全测试, 本文对模型进行了优化设计.

(1) 优化设计 1: 丰富攻击节点语义

攻击树模型中的节点仅仅是一种语义描述, 简单阐述了攻击操作的名称, 但并没有说明攻击操作的详细信息, 因此为了详细地阐述攻击操作的具体信息, 为攻击节点绑定名称、描述、层级、阶段、前置条件、后置条件等相关属性, 如表 1 所示.

表 1　攻击节点属性描述

属性	含义
名称	攻击操作的简要概括
描述	攻击操作的具体详细信息
层级	节点当前所在的层级, 用于阐述节点间的上下级关系
阶段	攻击操作当前所在的攻击生命周期环节
前置条件	触发攻击操作所需要某些资源或条件
后置条件	描述攻击操作产生的影响

本文将节点层级划分为顶层 (top level)、状态层 (state level)、事件层 (event level) 这 3 层结构[34]. 顶层描述攻击或防御的最终目标, 在攻击树模型中顶层中的节点往往只有一个, 即根节点; 状态层描述攻击树模型中存在中间状态的节点, 即根节点往往由若干状态层的节点组成, 状态层的节点可以继续划分为事件层的节点; 事件层描述攻击者实施的具体攻击操作或防御者实施的具体防御举措, 该层节点处于攻击树模型中的最底层, 且不可被划分.

本文基于网络杀伤链 (cyber-kill-chain) 模型[35]定义攻击阶段的取值. 网络杀伤链模型是由洛克希德·马丁公司提出的, 用来针对性描述攻击生命周期不同阶段的攻击分析模型. 由于攻击树模型基于攻击者的角度进行构建, 攻击路径代表了攻击者实现攻击目标的一次完整性操作, 所以路径中的节点可以映射到网络杀伤链模型中的不同阶段. 其中, 网络杀伤链模型的 7 个阶段[36]中, 前两个阶段 Reconnaissance 与 Weaponization 讲述的是攻击实施前的准备工作, 包括识别系统信息, 准备恶意脚本等. 中间的 4 个阶段 Delivery、Exploitation、Installation、Command and Control 讲述的是攻击者如何对目标系统实施具体攻击, 包括向目标发送恶意信息、利用恶意信息、安装后门程序并控制系统. 最后的阶段 Actions on Objectives 讲述的是攻击者实施完攻击后, 后续所采取的相关操作, 例如窃取敏感数据、消除踪迹等. 因此, 基于网络杀伤链模型, 可定义攻击节点阶段模型如图 4 所示.

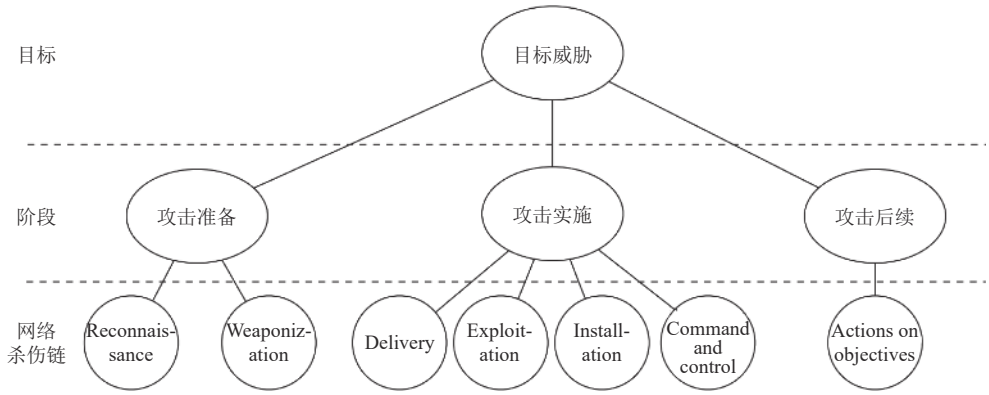


图 4 攻击节点阶段模型

在攻击节点阶段模型中, 底层为网络杀伤链模型, 展现了攻击者为实现目标威胁所进行攻击操作的 7 个阶段. 然而, 在实际应用中, 基于网络杀伤链模型来划分攻击阶段仍然比较繁琐. 因此, 本文对其 7 个阶段进一步概括, 得到攻击准备、攻击实施、攻击后续 3 个粗粒度的阶段, 即阶段的取值有 3 种: 攻击准备 (prepare)、攻击实施 (action)、攻击后续 (follow_up).

(2) 优化设计 2: 添加了操作符、防御节点

由于模型节点间的关联关系存在局限, 对于某些场景, AND 和 OR 关联类型无法满足. 因此, 本文设计了一种新的节点类型为操作符, 通过操作符对节点进行关联. 如图 5 所示, 操作符用菱形表示, 通过在 G0 下面挂载 AND 和 OR 类型的操作符, 能够将图 3 中节点间的关联关系进行解耦, 解决了节点单一类型的局限性问题. 添加的操作符抽离了节点的关联语义, 使节点更关注于攻击操作的描述.

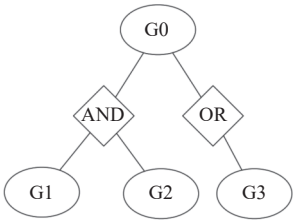


图 5 操作符解决节点关联关系局限性示意图

此外, 在攻击树模型中, AND 和 OR 类型均未描述攻击操作的有序性. 但在现实生活中, 攻击者实施的某些操作是严格有序的, 攻击顺序的错乱可能会导致攻击失败. 并且, 因为节点不具备有序性, 如果用户构建模型时, 没有按照攻击操作的有序性关联节点, 那么模型生成的攻击路径会杂乱不堪, 难以起到指导作用. 而有序性的特征是从前到后依次执行, 其蕴含操作均执行的前提. 由于 AND 类型可以描述操作均执行的特征, 所以本文基于 AND 类型再补充一种节点类型 Sequential-AND, 简称为 SAND[37], 如图 6 所示, 展示了 3 种攻击节点的关联类型.

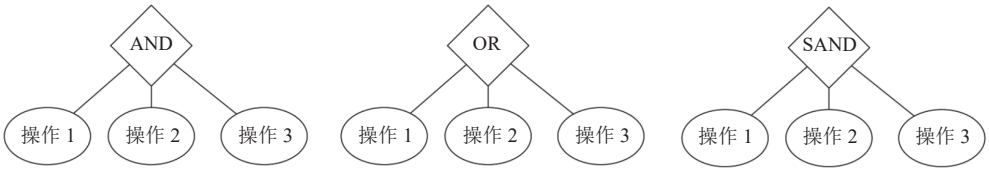


图 6　攻击节点的关联类型

添加的防御节点主要是用来表示攻击节点所对应的防御方案, 实施防御方案是消减威胁的有效措施. 通过添加防御节点有助于对检测的威胁进行消减, 指导安全设计和实现, 防御威胁的发生. 防御节点间的关联关系和攻击节点一致, 存在 AND、OR 和 SAND 这 3 种关联类型, 本文添加的防御节点用矩形表示.

(3) 优化设计 3: 为攻击节点增加评估属性和测试属性

评估属性用于评估测试线索的优先级, 本文采用对攻击节点进行评估的思想[38], 通过为节点添加评估属性, 计算节点的发生概率, 进而确定生成的测试的优先级. 本文采用 CVSS 评估法, 为攻击节点添加 CVSS 指标, 来确定节点的发生概率.

为攻击节点增加测试属性, 用于生成测试用例. 基于攻击节点形成的攻击路径往往是抽象的, 未能体现攻击者的详细操作, 用户凭借抽象的指导难以对威胁进行检测. 因此, 本文为攻击节点添加测试相关的属性信息, 包含测试操作的调用方法、输入参数、预期结果.

2.3　模型定义

本节展示改进的攻击树模型的定义和元模型. 改进的攻击树模型的元模型如图 7 所示, 其中包含节点类型、节点间的关联关系以及节点包含的属性信息. 模型的具体定义如下.

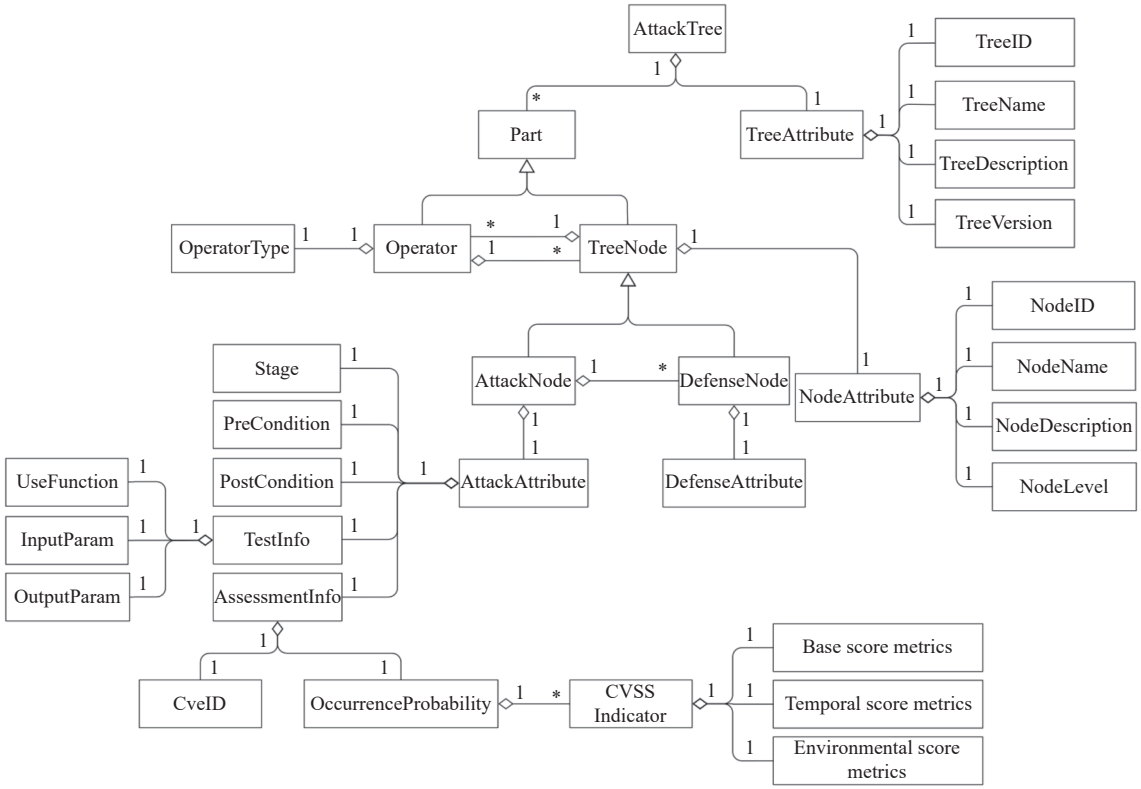


图 7　攻击树模型的元模型

定义 1. 攻击树模型. 攻击树模型由一系列构件 Part 和相关属性 TreeAttribute 构成, 有关系式 ATM=(Pt, Ta), 其中: (1) Pt 是一个非空有限集合, 囊括了模型的所有构件信息. Pt 中的元素 Part 有两种类型, 分别是操作符构件 Operator、树节点构件 TreeNode. (2) Ta 是模型的相关属性集合. 该集合中包含 4 种属性: TreeID 表示模型的 ID 标识; TreeName 描述模型的名称, 一般将威胁的描述作为该字段的属性值; TreeDescription 阐述模型的具体信息; TreeVersion 用作模型的版本管理. 其中, TreeName 和 TreeVersion 的组合, 用于唯一标识模型. 当用户创建模型时, 本文开发的工具, 会通过这两个字段在数据库中检索, 检查创建的模型是否与已存在的模型发生命名冲突.

定义 2. 构件 (Part). 构件是模型的核心, 在定义 1 中提到, Part 的类型有两种: Operator、TreeNode. 该模型与传统的攻击树模型相比, 最主要的区别在于其结构由 Part 组成, 而传统攻击树模型由攻击节点组成.

(1) Operator 用来特指节点间的关联关系, 有关系式 Operator=(Tn, Ot), Tn 表示 Operator 关联的节点列表, Ot 表示 Operator 的类型. OperatorType 的类型有两大类, 共 6 种, 其中攻击节点的关联以 A_开头, 防御节点的关联以 D_开头, 即 OperatorType∈{A_AND, A_OR, A_SAND, D_AND, D_OR, D_SAND}, 相同类型的节点前缀保持一致.

(2) TreeNode 用来描述节点的具体信息, 有关系式 TreeNode=(Op, Na), 其中 Op 表示节点关联的 Operator 列表, Na 表示节点的属性描述. TreeNode 的类型有两种, 分别是攻击节点 AttackNode 和防御节点 DefenseNode; Na 表示 AttackNode 和 DefenseNode 的通用属性信息, 有关系式 Na={NodeID, NodeName, NodeDesciption, NodeLevel}, 其中 NodeID 表示节点的 ID 标识, NodeName 描述节点的名称, NodeDescription 阐述节点的具体信息, NodeLevel 代表节点的层级. 在 Part 集合中, Operator 和 TreeNode 相互关联, 共同构成了模型的树形结构. 其中, 用户在构建模型时, 以攻击目标作为根 TreeNode, 自顶向下划分, 直到目标全部分解为具体的攻击/防御操作.

定义 3. 节点层级 (NodeLevel). 节点层级用来描述节点间的上下级关系, 可分为顶层 (top level)、状态层 (state level)、事件层 (event level) 这 3 层结构, 有关系式 NodeLevel∈{Top, State, Event}.

定义 4. 攻击节点 (AttackNode). 攻击节点用来描述攻击操作信息, 有关系式 AttackNode=(Aa, Dn). 其中 Aa 表示节点的属性列表, Dn 表示节点关联的 DefenseNode 列表, 代表该节点对应的防御方案列表. 如图 8 所示, AttackNode 与 DefenseNode 通过虚线连接.

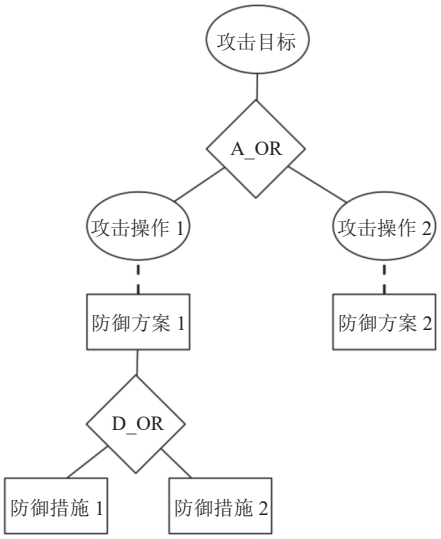


图 8　攻击节点与防御节点衔接示意图

定义 5. 防御节点 (DefenseNode). 防御节点用来描述防御方案信息, 有关系式 DefenseNode=(Da). 其中 Da 表示节点的属性列表, 与 AttackNode 不同, 其关系式中不存在指向 AttackNode 的关联. 根据 DefenseNode 的定义, 当 AttackNode 不存在时, DefenseNode 不会孤立存在. 由于 AttackNode 中已经存在指向 DefenseNode 的引用, 因此 DefenseNode 未重复定义指向 AttackNode 的引用.

定义 6. 攻击属性 (AttackAttribute). 攻击属性用来描述攻击节点特有的属性信息, 包括攻击阶段 (生命周期)、前置条件、后置条件、测试信息、评估信息这 5 方面内容, 有关系式 AttackAttribute={Stage, PreCondition, PostCondition, TestInfo, AssessmentInfo}.

(1) Stage 用来描述攻击阶段, 即攻击操作处于攻击生命周期的哪个环节, Stage∈{Prepare, Action, Follow_Up}. 如图 9 所示, 攻击阶段严格有序, 该特征可被用来修正攻击节点间的有序性.

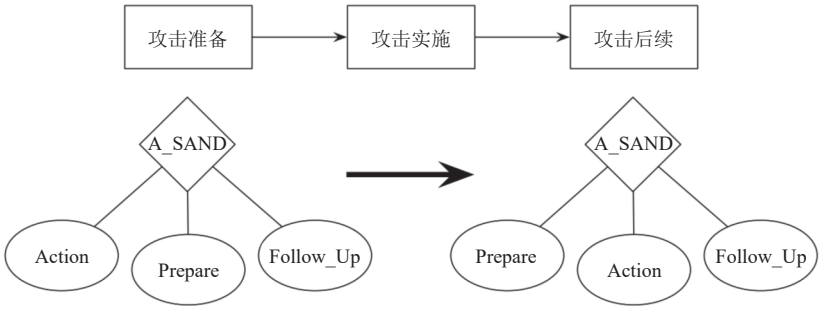


图 9　攻击阶段的有序性

(2) PreCondition 用来描述攻击操作的前置条件, 例如必须需要某些资源或必须完成某些操作才能导致该操作的发生.

(3) PostCondition 用来描述实施攻击操作后产生的影响. 例如实施 XSS 攻击, 成功拿到了用户的登录凭证.

(4) TestInfo 用来描述节点的测试属性信息, 有关系式 TestInfo={UseFunction, InputParam, OutputParam}, UseFunction 表示测试调用的具体方法, InputParam 表示输入参数, OutputParam 表示预期结果. 设置 TestInfo 字段的原因在于将抽象化的攻击路径转换为可供参考的测试用例.

(5) AssessmentInfo 用来描述节点的评估属性信息, 有关系式 AssessmentInfo={CveID, OccurrenceProbability}, CveID 表示利用漏洞的 CVE 标识, OccurrenceProbability 表示节点发生概率的数值. OccurrenceProbability 的数值由一系列的 CVSS 指标通过特定算式得到.

定义 7. 防御属性 (DefenseAttribute). 防御属性用来描述防御节点特有的属性信息, 主要是描述防御方案方面的内容, 这个属性为一扩充属性, 用户根据需求进行设置和重用.

3　基于威胁模型的安全测试用例生成框架

本文所提出的基于威胁模型的安全测试用例生成框架如后文图 10 所示, 主要包括从威胁建模工具识别的威胁作为输入, 构建威胁对应的攻击树模型, 可以基于模板自动构建, 也可以通过可视化构建、XML 文本构建等方式手动构建; 对模型进行规范性检查, 分为对 XML 文件的校验和对可视化图形的校验; 基于符合规范性的模型采用算法生成攻击路径, 生成的攻击路径即为指导威胁检测的测试线索; 对生成的路径进行评估并选择防御威胁的方案, 该环节主要是对模型的节点进行评估, 通过对攻击节点进行评估从而计算攻击路径的发生概率, 通过对防御节点进行评估从而选择收益性较高的防御方案, 其中模型节点评估也可以选择在构建了规范的模型以后就进行; 优先选择发生概率大的测试线索进行测试, 通过设置测试参数并加载测试用例模板, 将测试线索转换为测试用例. 通过该框架可以生成测试, 用于检测威胁是否会发生, 选择收益性较高的防御方案, 有助于对威胁进行防御和缓解, 改进系统安全设计, 提高系统的安全性. 接下来对测试用例生成框架流程进行具体介绍.

3.1　模型构建

本文提供了包含常见威胁的内置模板库以支持模型的自动化构建, 提高构建效率. 同时本工具还支持模型的手动构建方式来作为补充和选择.

(1) 基于模板的自动构建. 自动构建方式中可以根据输入的威胁自动匹配内置模板库中符合需求的模板, 自动

生成模型. 具体步骤为: 通过威胁解析引擎解析输入的威胁数据; 获取威胁对应的 CWE_ID、CAPEC_ID、类别字段信息; 然后根据获取的相关信息去内置模板库中检索对应或相似的内置模板. 威胁的内置模板库基于 CAPEC 攻击库构建, 而 CAPEC 库中列举了 500 多种攻击类型, 常见的攻击模式有: 注入攻击、资源耗尽攻击、认证攻击等, 其结构和 CWE 漏洞分类库的结构类似, CAPEC 包含的 Related Weaknesses 字段能够关联威胁列表中的 CWE_ID 字段, 因此, 能够通过该字段从内置模板库匹配模板, 根据模板渲染模型.

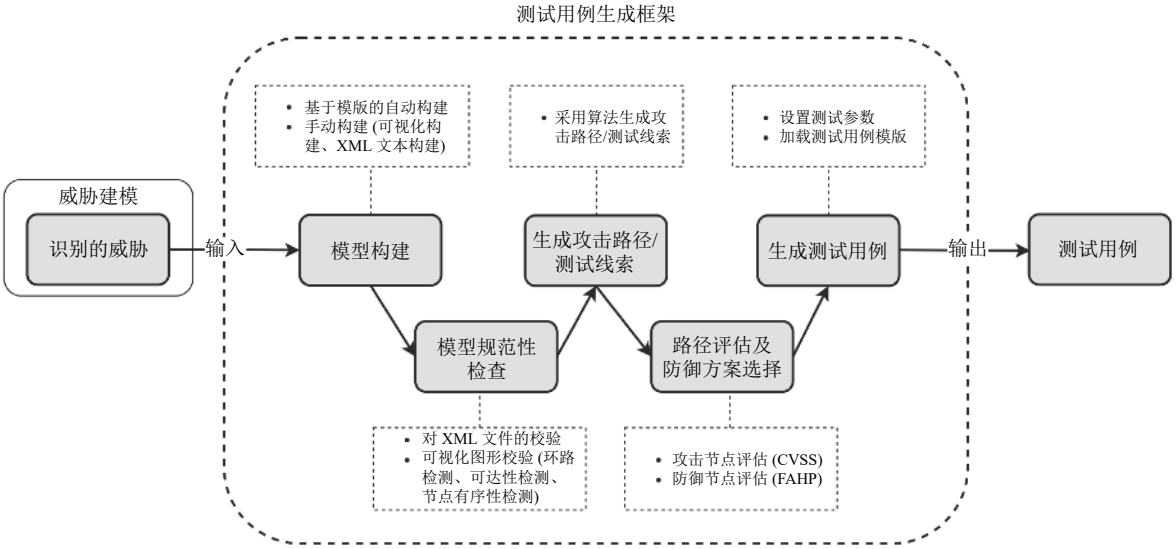


图 10 基于威胁模型的测试用例生成框架图

内置模板库是工具为帮助用户快速构建模型而设立的辅助仓库, 在该仓库中提供了 Web 安全领域常见威胁的攻击树模型. 用户在构建模型时, 可以从模板库中选择跟分析的威胁较为匹配的模板, 无需从零开始构建, 节省了大量的时间. 本文对 OWASP Top Ten (https://owasp.org/www-project-top-ten/) 和 CAPEC (https://capec.mitre.org/data/index.html) 库进行分析处理得到 Web 安全领域常见威胁的内置模板. OWASP Top Ten 是由开放 Web 应用程序安全项目 (open Web application security project, OWASP) 发布的威胁文档, 记录了 Web 应用程序中最为关键的威胁类型, 目前最新的 2021 年文档将常见的 Web 安全领域的威胁分为 10 大类, 共计 182 种; CAPEC (common attack pattern enumeration and classification) 是由美国国土安全部 (United States Department of Homeland Security, DHS) 发起的供软件界共享攻击模式的分类数据集, 包含了进行欺骗性互动、操作数据结构、颠覆访问控制等 9 类攻击机制 (mechanisms of attack), 共计 546 种攻击模式. OWASP Top Ten 和 CAPEC 之间互通, OWASP Top Ten 数据集通过 Related Attack Patterns 字段为每种威胁绑定了对应的 CAPEC 列表, 同理, CAPEC 也可以通过 Related Weaknesses 字段找到攻击模式对应的威胁列表.

OWASP Top Ten 中的 182 种威胁对应 CAPEC 中的 318 种攻击模式, 本文结合目前广泛采用基于 Web 方式开发系统软件的现状[39], 对 OWASP Top Ten 匹配的 318 种攻击模式进行处理, 并生成对应的内置模板. CAPEC 提供的攻击模式是通过分析攻击者利用漏洞时经常使用的机制分层组织而来, 其为树状结构, 共分为 4 层: 类别模式 (category)、元模式 (meta)、标准模式 (standard)、详细模式 (detailed), 层级按照从抽象到具体依次细化, 本文参照 CAPEC 层级的定义将模板的级别划分为 4 层, 分别与 CAPEC 的 4 层攻击模式相对应. 确定的内置模板层级和个数如表 2 所示.

模板的制定首先以详情级别的攻击模式为出发点, 分析其攻击实施方式、发生条件、结果等信息, 构建具体威胁的模板; 之后对同属同一标准 (standard) 的其他攻击模式进行分析, 抽离共性, 定制第 2 层该标准 (standard) 的模板; 最后按照同样的准则, 构建第 1 层、第 0 层模板. 图 11 展示了自底向上制定模板的示意图.

表 2　内置模板层级和个数

内置模板层级	CAPEC层级	描述	个数
0层	类别模式	类别模式是基于某些共同特征的攻击模式的集合. 更具体地说, 它是基于效果/意图的攻击模式的聚合, 是基于一些通用标准的模式分组	7
1层	元模式	元模式是攻击中使用的特定方法或技术的绝对抽象特征, 通常没有特定技术或实现, 旨在提供对高级方法的理解, 是相关标准级别攻击模式组的概括. 元模式对于架构和设计级别威胁建模练习特别有用	30
2层	标准模式	标准模式专注于攻击中使用的特定方法, 通常被视为完全执行攻击的单一部分, 旨在提供足够的细节来理解特定方法以及它如何尝试实现期望的目标, 是一种元模式的特定类型	97
3层	详细模式	详细模式提供了低级别的详细信息, 通常针对特定技术, 表达其完整的执行流程. 该模式比元模式和标准模式更具体, 通常会利用许多不同的标准级别攻击模式链接在一起来实现目标	191

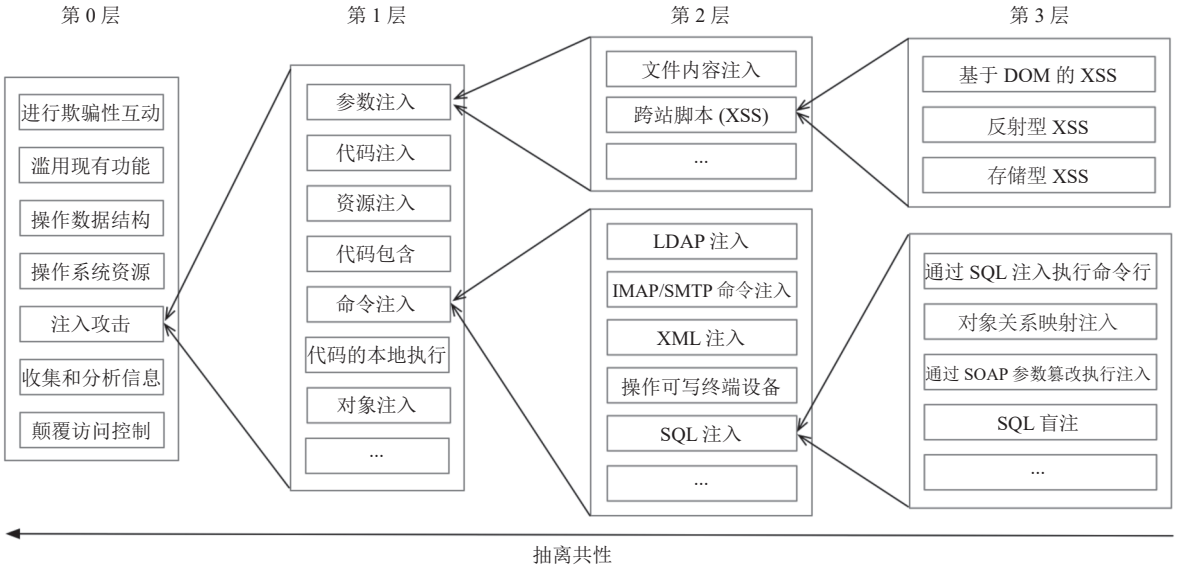


图 11　模板制定示意图

CAPEC 中所包含的 Execution Flow 字段用来描述攻击的实施方式, 适合作为模型的攻击节点数据; Mitigations 字段用来描述攻击的消减方案列表, 适合作为模型的防御节点数据. 由于在 CAPEC 数据集中, 两字段的属性值均为一段语义描述, 因此需要对其进行处理, 将其转化为可利用的数据, 处理两字段的流程为: 通过预处理程序, 分别对两字段的属性值进行分词整理, 得到初始的处理数据; 对初始的处理数据进行拆分, 得到模板的节点列表; 编制列表中节点的属性值, 并绑定节点间的关联关系, 得到内置模板文件; 将内置模板文件加入到模板库中. 以 Execution Flow 字段为例, 该属性值由 Step、Phase、Description 关键词进行分隔. 本文通过分析该字段的组成特征, 编制预处理程序, 对特殊字词进行正则匹配, 提取需要的关键信息, 得到了攻击准备阶段、攻击实施阶段、攻击后续阶段的操作描述. 之后, 根据模型的定义将处理后的操作描述拆分为具体的攻击节点, 并绑定节点的关联关系, 得到无防御节点的模板文件. 同理, 对 Mitigations 字段的处理也类似, 当将其转化为一系列防御节点后, 根据节点的描述与对应的攻击节点进行绑定, 最终可以得到包含攻击节点和防御节点的完整模型. 图 12 为制定模板的流程图.

(2) 手动构建. 手动构建方式分为可视化构建方式和 XML 文本构建方式两种. 可视化构建方式是通过拖拽模型元素进行可视化构建, 模型元素包括操作符和节点构件以及属性信息. XML 文本构建方式为将建模元素以 XML 格式进行编写, 用户通过快速编写 XML 文件的方式, 将文件导入到工具中, 经工具解析后得到对应的模型.

在模型构建中, 用户可以选择其中一种方式进行构建, 也可以先通过文本构建方式构建模型的雏形结构, 然后再通过可视化构建方式进行修改补充.

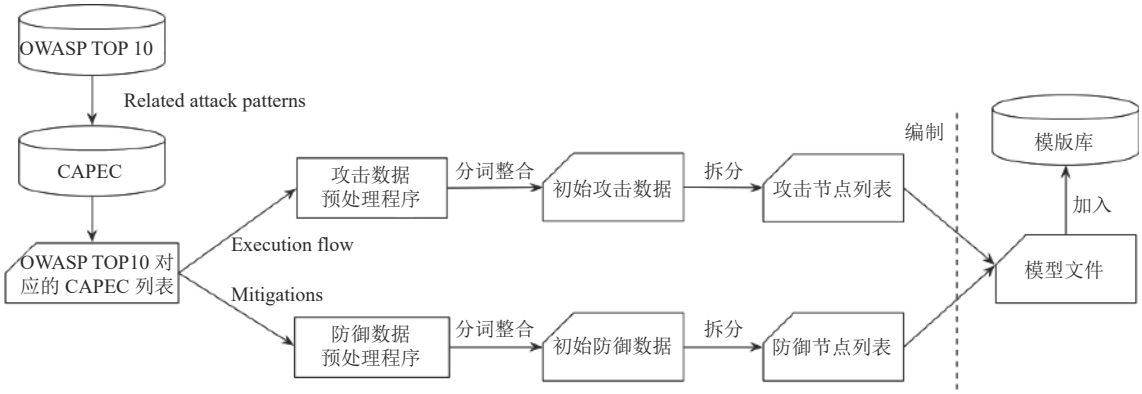


图 12 模板制定流程图

3.2 模型规范性检查

对构建的模型进行规范性检查, 以保证模型的规范性, 确保基于该模型生成的测试的正确性. 根据模型的定义制定基本的规则约束内容, 分别为 XML 文件语义规则和可视化图形结构规则, 规范性检查引擎通过规范性检查算法校验模型是否符合这两方面的语义约束, 对模型进行规范性检查.

模型的手动构建方式有两种, 分别是基于 XML 的文本构建方式和基于 Web 界面的可视化构建方式, 因此校验算法也分为两种形态, 一种是对 XML 文件的校验, 另一种是对可视化图形的校验.

● 对 XML 文件的校验: 定义 XML 语义规则如表 3 所示. 对 XML 文件进行解析, 如果 XML 文件格式不符合语义规则, 向用户抛出解析错误提示; 如果符合语义规则, 则经工具渲染模块, 渲染成可视化模型.

表 3 XML 语义规则

规则序号	描述
rule-1	根标签必须是<AT/>
rule-2	根标签下只有一个攻击节点标签<attackNode/>
rule-3	攻击节点标签下存在两种标签类型, 操作符标签<operator/> 和防御节点标签<defenseNode/>, 数量不做限制
rule-4	防御节点标签下仅存在操作符标签<operator/>, 数量不做限制
rule-5	操作符标签下存在两种标签类型, 攻击节点标签<attackNode/> 和防御节点标签<defenseNode/>, 数量不做限制
rule-6	操作符标签type属性仅有6种取值: A_AND、A_SAND、A_OR、D_AND、D_SAND、D_OR
rule-7	攻击节点标签level属性仅有3种取值: TOP、STATE、A_EVENT
rule-8	仅有level属性值为A_EVENT的攻击节点标签具有stage属性的取值
…	…

● 对可视化图形的校验: 主要检查模型是否存在环路, 以及检查模型的可达性和节点有序性, 运用的算法有环路检测算法、可达性分析算法和节点重排序算法.

(1) 环路检测

模型的图形化展示是一种树形结构, 规范的树形结构不应存在环路. 工具采用从根节点遍历到叶子结点的方式对模型进行分析, 确定模型中除根节点之外的所有节点仅有唯一的入度. 当节点间存在直接相互引用或间接相互引用关系时, 工具会无休止地重复遍历某些节点, 一直处于解析环路中, 引发内存堆栈溢出, 无法正常运行. 环路检测可以避免这种情况发生, 确保模型的规范性.

(2) 可达性检查

可达性检查确定模型不存在孤立节点的情况. 用户在构造模型时, 可能会疏忽部分节点间的关联, 导致节点是

孤立的状态. 然而, 在工具实现中, 孤立的节点并不参与运算, 导致构建的模型可能与用户预期不符. 从模型根节点出发进行深度优先遍历 (DFS), 判断访问节点个数是否等于模型节点的个数, 从而判断模型节点是否均可达.

(3) 节点有序性检查

节点有序性检查用于确定构建的模型中的节点顺序是否符合攻击生命周期约束, 运用节点重排序算法检查和调整模型中攻击节点的有序性. 如果用户未依据 Stage 语义设置 A_AND/A_SAND 操作符关联节点顺序, 则可能导致攻击路径中节点顺序有误, 不符合攻击生命周期, 影响威胁检测的正确性. 工具在生成测试时, 会对操作符关联的攻击节点按照 Stage 语义从左到右的顺序重新排序, 以确保模型节点的有序性.

3.3 基于模型的测试线索生成

本文基于传统的白盒测试和黑盒测试, 采用动态测试用例生成方法. 用例被理解为用户和系统之间的一段互动, 一些研究人员和从业者从许多方面将用例具体概括为所有功能和行为[40], 基于攻击步骤得到的攻击路径即为一组用例. 攻击路径是攻击者实现攻击目标可能执行的操作步骤和途径, 根据攻击路径进行测试可以用于检测威胁. 因此, 攻击路径的序列是指导安全威胁测试的测试线索, 每条攻击路径即为一条测试线索. 线索[41]在针对不同问题时的看法是不一样的, 可以将线索看成一般使用的场景、系统级测试用例、对象消息和方法执行的交替序列、系统状态机描述中的转换序列、指令序列、决策路径和原子系统功能序列等. 线索有不同的层次, 可以分为用例级和单元级线索. 攻击路径对应的序列是用例级的线索. 例如根据攻击路径节点集合{A, B, C, D}可以形成测试线索 start → A → B → C → D → end . 测试人员可以根据攻击路径形成的测试线索来设计测试用例, 使得威胁检测更具方向性.

本研究中的测试线索基于改进攻击树模型生成, 可以在攻击路径提取的同时加入攻击和测试属性作为该条测试线索执行的条件参数, 为威胁攻击检测和分析提供更多的信息, 测试员可以根据这些信息来设计测试用例. 此外可以根据节点添加的评估属性对测试线索按照发生概率进行优先级排序.

攻击树模型可以生成若干攻击路径, 而攻击路径由模型中达成攻击目标的攻击步骤的最小集合组成, 表示为攻击者实现攻击目标可能执行的操作途径, 所以计算模型的攻击路径本质上是找到模型中攻击步骤的全部最小集合. 本文对模型进行了优化设计, 将集合包含的节点类型由叶子结点转化为 A_EVENT 类型的攻击节点, 即集合的结构由一个或多个 A_EVENT 类型的攻击节点组成. 基于改进攻击树模型得到攻击路径需要从模型的根节点出发, 根据攻击节点类型和操作符类型的不同, 将节点生成的最小集合进行相互组合. 计算模型的最小集合过程整体为一递归回溯流程.

3.4 路径评估及防御方案选择

对生成的路径进行评估并选择防御威胁的方案主要是对模型的节点进行评估, 通过对攻击节点进行评估从而计算攻击路径的发生概率, 通过对防御节点进行评估从而选择收益性较高的防御方案, 其中模型节点评估也可以选择在构建了规范的模型以后就进行, 本文提供了攻击评估策略和防御评估策略用于对模型节点进行评估.

(1) 攻击节点评估

通用漏洞评分系统 (common vulnerability scoring system, CVSS, https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator)[42]是一个开放的框架, 用于评估软件漏洞的特征和严重性, 通过为模型中的攻击节点设置 CVSS 指标, 可以计算得到节点的 CVSS 得分, 进而确定攻击路径的发生概率, 从而根据发生概率的大小对路径进行优先级排序.

利用系统漏洞是一种众所周知的攻击方式, 攻击者实施的攻击举措往往是从已被披露的漏洞着手, 利用操作系统或常用软件探明的漏洞对系统发起攻击[43]. 目前, 业界权威的漏洞披露库有 CVE (https://www.cve.org/)、NVD (https://nvd.nist.gov/) 等. 在 CVE 和 NVD 中, 常常会对披露的漏洞从描述、严重程度、相关的解决方案等方面来进行介绍. 其中, 最为关键的是漏洞的严重程度, 严重程度决定了漏洞的风险量度. 风险量度共有 5 种级别, 分别是无、低级、中级、高级、严重级, 其间接反映了漏洞的威胁发生概率, 通常风险量度越高的漏洞, 往往发生概率越高, 越容易被攻击者利用. 判断漏洞风险级别基于 CVSS 标准, 对攻击措施发生概率的度量便可转换为对利用漏洞严重程度的分析. 当攻击措施不是利用漏洞的方式来施行时, 对攻击措施发生概率的度量无法通过对利用漏

洞严重程度来进行分析. 这种情况需要用户依据 CVSS 标准来设置节点的 CVSS 指标, 评估攻击措施的风险发生概率. 本文将基于 CVSS 标准对攻击节点的发生概率进行评估, 攻击措施的发生概率与漏洞的严重程度两者间成正比例关系. 此外, 基于 CVSS 的得分来量化攻击节点的发生概率.

在得到所有攻击节点发生概率以后, 可以进一步得到攻击路径的发生概率, 攻击路径的发生概率由路径中节点的发生概率乘积开根后得到. 评估攻击路径的发生概率, 能够让用户了解攻击者最可能会采取哪一种攻击途径, 优先根据发生概率高的攻击路径形成的测试线索进行测试, 能够提高测试效率.

(2) 防御节点评估

模糊层次分析法 (fuzzy analytic hierarchy process, FAHP)[44]是一种改进的层次分析法 (analytic hierarchy process, AHP). AHP[45]是美国著名运筹学家 Satty 教授在 20 世纪 70 年代提出的一种定性和定量分析相结合的系统分析方法. 这种方法的特点在于对决策问题的本质、指标及其内在关系等因素进行深入研究的基础上, 利用较少的定量信息使决策的思维过程数学化, 从而为多目标、多准则或无结构特性的复杂决策问题提供简便的决策方法. 该方法通过对难以定量的复杂系统构建决策模型, 由专家进行指标打分, 确定决策指标的权重, 进而确定最合适的决策方案. AHP 存在不容易满足一致性的缺陷, 因此 FAHP 用模糊一致性判断矩阵[46]来对 AHP 的缺陷进行弥补.

用户筛选出发生概率较高的攻击路径后, 需要针对性地为路径中的某些节点制定防御措施来进行消减. 针对某种攻击的消减措施有多种, 例如 CAPEC 攻击库, 收集了 500 多种攻击类型, 并为每种类型绑定若干消减措施, 因此, 如何选择最理想的措施需要用户根据实际情况进行判断. 目前, 业界并没有提供一种好的防御推荐标准, 对防御措施的选择多数以工作经验为主. Wang 等人[47]和 Ji 等人[48]利用经济学中投资回报率 ROI 的概念, 来评估防御措施的好坏. 但是这种评估方法的前提是, 需要用户具体落实各项措施. 只有落实全部措施后, 才能得知哪种措施回报率最高, 本质上讲该评估方法属于一种负反馈的评估机制. 虽然该方法根据系统的反馈能够清晰地获知哪种措施最好, 但是其成本过大, 需要对每一种措施进行测试, 因此并不适用于实际场景. 本文将基于人工经验和 FAHP 结合的方式对防御措施进行评估.

在现实情况中, 针对防御措施的评估, 有多种考虑维度. 例如考虑实施的成本, 一般公司更青睐于购买防御服务少、消耗人力资源低、可快速建设的措施; 考虑防御实施的难度, 难度较低的策略, 只需要初级安全人员就可以进行实施; 考虑防御措施的稳定时间, 在绝大多数公司中更青睐稳定性高的策略, 例如一个措施相比其他措施而言可能花费的成本较高, 但是其保护系统的时间更长, 结合系统收益而言, 扣除成本后系统取得的平均收益反而更高. 所以考虑维度不同, 防御措施的选择也会千差万别. 因此, 需要综合各种考虑维度, 并选择最为合适的方案.

此类问题为典型的决策问题, 本文采用 FAHP 并基于 3 种考虑维度: 成本、难度、措施的稳定时间来对防御方案进行选择. 其中, 决策模型中的目标层为选择最合适的防御措施、准则层为防御评估的 3 个指标、方案层为各防御措施, 如图 13 所示. 按照防御评估策略的流程对防御措施进行评估: 参照 0.1–0.9 标度法制定成本、难度、能力指标的模糊判断矩阵; 确定成本、难度、能力指标的权重; 制定方案层中防御措施针对各个指标的模糊判断矩阵; 确定防御措施针对各个指标的权重; 计算各防御措施的总目标得分. 按照 FAHP 对防御措施进行层次分析, 能够得到各防御措施的总目标得分, 根据得分的大小, 能够明确哪些防御措施被优先采纳.

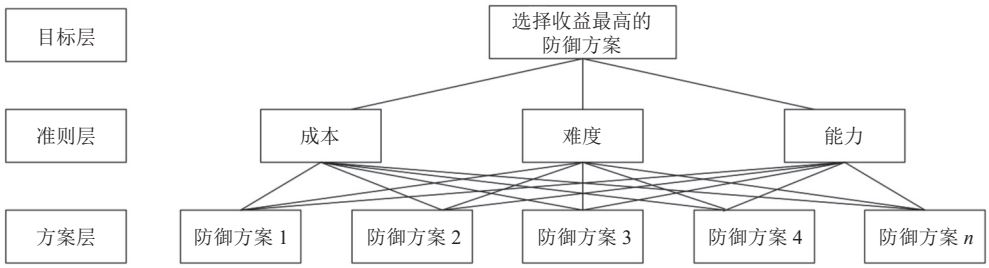


图 13 防御评估策略的 AHP 决策模型

3.5 测试用例生成

测试用例生成的目的在于将抽象的攻击路径转换为具体的测试用例, 便于用户参考. 优先选择发生概率大的攻击路径形成的测试线索, 将其转换为测试用例, 有助于提高检测威胁的效率. 由上文得知, 攻击路径由一系列攻击操作级联而成, 虽然其能够给用户提供初步的指导建议, 使用户模仿攻击者对系统实施类似的操作, 去测试系统中是否存在该威胁. 然而, 这并不是一种明确的测试用例, 路径仅仅描述了测试用例的操作步骤, 通常而言, 在常见的测试用例中, 除操作步骤外, 还应包含测试名称、前置条件、预期结果、测试结果等属性.

这对于攻击路径而言, 并不能满足上述测试相关的属性信息, 因此与符合业务场景的测试用例相比, 仍有所欠缺. 为解决上述问题, 本文设计了一种测试用例生成策略, 如图 14 所示, 该策略基于攻击路径的指导, 贴合业务场景的安全测试需求, 能够利用攻击节点绑定的测试参数信息和测试用例模板, 将攻击路径转换为测试用例, 并以报表的形式呈现给用户.

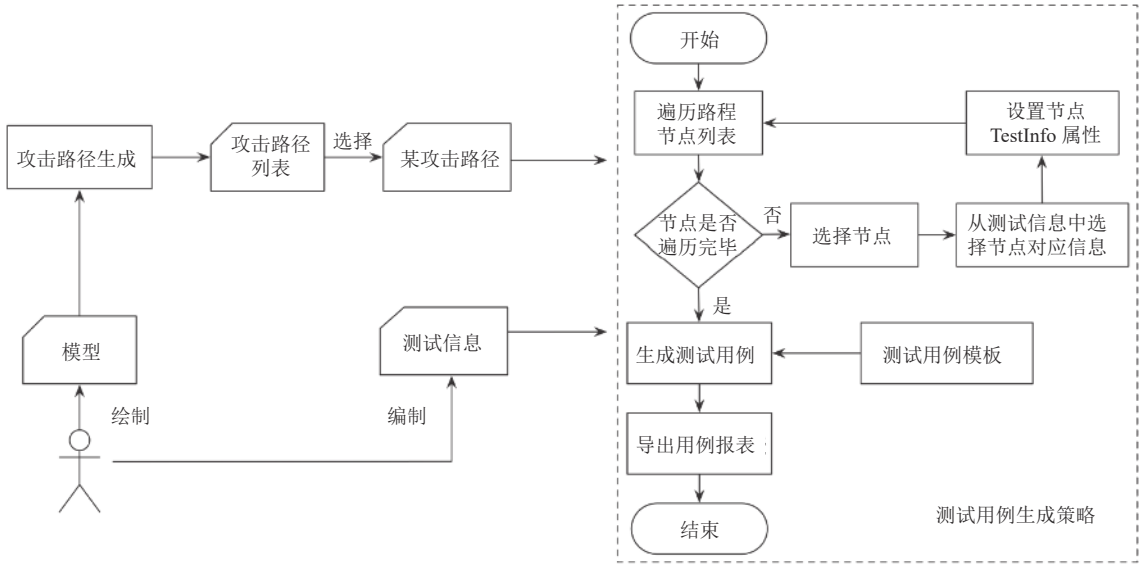
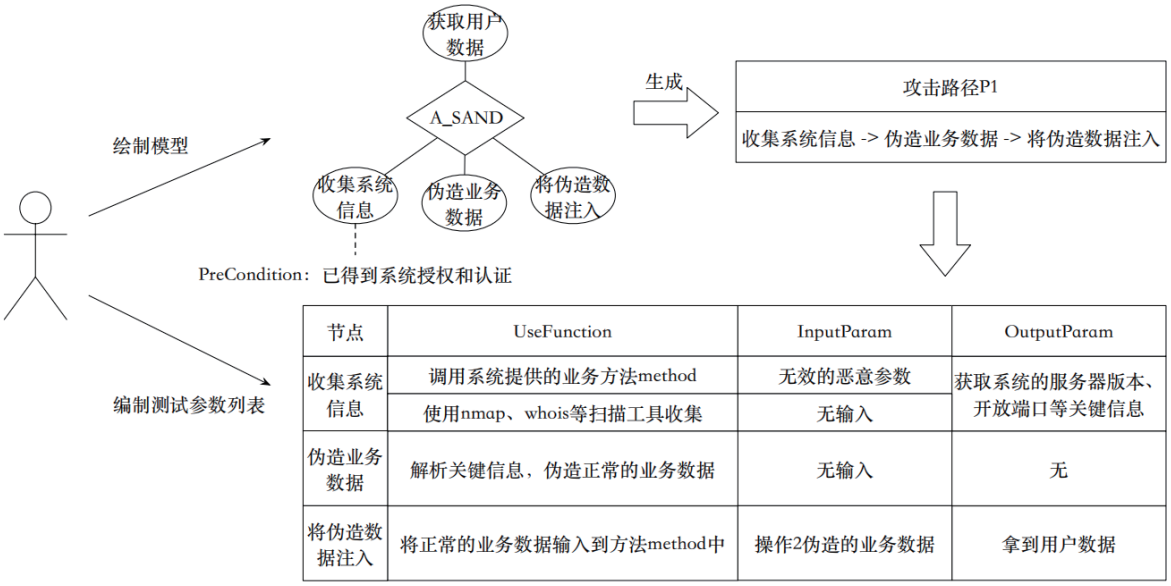


图 14 测试用例生成流程

图 14 中, 测试信息 TestInfos 的格式为散列表结构, 键为节点 Node, 值为节点对应的参数列表 TestParamList. 参数列表中的每一项 TestParam 为三元组结构, 包含调用方法 UseFunction、输入参数 InputParam、预期结果 OutputParam 这 3 个字段, 与节点测试字段 TestInfo 中的属性一一对应. 在测试用例生成策略中, 同一节点绑定的测试参数默认仅有唯一的预期结果, 即在测试参数列表中, 仅设置一次 OutputParam 字段即可.

当用户将具体的测试信息绑定到节点的测试属性字段后, 利用测试用例生成策略中提供的测试用例模板, 可以将节点的测试参数信息解析到模板中, 进而将攻击路径转化为可供参考的测试用例. 该策略还可将用例导出, 以报表的形式呈现给用户, 辅助用户检测威胁.

如图 15 测试用例生成样例所示, 攻击者的目的在于获取用户的系统数据, 有一条攻击路径 P1 可以实施: 收集系统信息 → 伪造业务数据 → 将伪造数据注入. 从路径 P1 可以看出, 该路径较为抽象, 不能够为用户提供具体的测试指导. 因此, 用户通过为路径 P1 的节点绑定测试参数列表, 进一步将攻击操作具像化. 例如收集系统信息操作, 有两种具体的手段可以实施, 第 1 种为调用系统提供的业务方法 method, 输入一些无效的恶意参数, 导致系统报错, 通过系统展示的错误报告可以获取系统的服务器版本信息; 第 2 种为使用 nmap、whois 等专业的扫描工具, 对系统进行扫描, 拿到系统 IP、开放端口等关键信息. 在设置完测试参数列表后, 工具会加载测试用例模板, 将攻击路径和测试参数信息进行渲染, 进而得到路径 S1 对应的测试用例报表. 最后, 用户参考该用例, 去调用自动化检测脚本或执行相关操作, 检测系统是否存在泄漏用户数据的风险.



节点	UseFunction	InputParam	OutputParam
收集系统信息	调用系统提供的业务方法method	无效的恶意参数	获取系统的服务器版本、开放端口等关键信息
	使用nmap、whois等扫描工具收集	无输入	
伪造业务数据	解析关键信息，伪造正常的业务数据	无输入	无
将伪造数据注入	将正常的业务数据输入到方法method中	操作2伪造的业务数据	拿到用户数据

测试用例模版	
用例编号	路径{pathId}-TC
测试名称	测试攻击路径{pathId}
前置条件	1、节点1的{PreCondition} 2、节点2的{PreCondition} … n、节点n的{PreCondition}

操作步骤	1、节点1的{UseFunction}//输入：节点1的{InputParam}
	2、节点2的{UseFunction}//输入：节点2的{InputParam}
	…
	n、节点n的{UseFunction}//输入：节点n的{InputParam}

预期结果	1、节点1的{OutputParam} 2、节点2的{OutputParam} … n、节点n的{OutputParam}
测试结果	

渲染 → 生成

用例编号	路径P1-TC
测试名称	测试攻击路径P1
前置条件	已得到系统授权和认证
操作步骤	1、调用系统提供的业务方法method//输入：无效的恶意参数； 或者 使用nmap、whois等扫描工具收集 2、解析敏感信息，伪造正常的业务参数 3、将正常的业务数据输入到方法method中//输入：操作2伪造的业务数据
预期结果	1、获取系统的服务器版本、开放端口等关键信息 2、拿到用户数据
测试结果	

图 15　测试用例生成样例

4　工具原型

为了对接威胁建模识别和分析成果, 本文设计和实现了测试用例生成工具, 以支持领域专家、业界软件设计人员、开发人员以及测试人员在早期进行威胁建模并生成测试的安全实践, 用于指导后续开发和测试设计等. 本工具架构根据典型的 4 层架构进行设计和划分, 整体架构图如图 16 所示. 第 1 层表现层负责向用户展示业务流程和数据, 包含绘图管理、模型导入导出、模型检查等功能; 第 2 层服务层, 提供模型存取、文件解析、模型检查、节点评估等服务; 第 3 层持久层主要负责保存用户和模型信息, 将用户编辑的模型以及模型中涉及的节点、测试参数等信息存储到数据库或文件系统中; 第 4 层数据层主要负责模型结果的保存和读取. 各层的具体介绍如下.

表现层: 该层为用户提供访问入口, 用户通过该层可以直观地使用可视化建模、模型规范性检查、模型节点评估等工具核心功能. 工具前端采用 Vue 框架结合 Element UI 组件进行开发, 整个页面整洁美观, 交互良好; 对于图形的绘制, 采用可视化引擎 G6-Editor 作为底层支持, G6-Editor 拥有优秀的交互和卓越的性能, 能有效支持灵活

的图形编辑和拖拽; Axios 作为基于 Promise 的 HTTP 库, 可以用于发起 Ajax 请求, 调用服务层提供的 RESTful API, 让前端开发更集中于数据处理和页面逻辑.

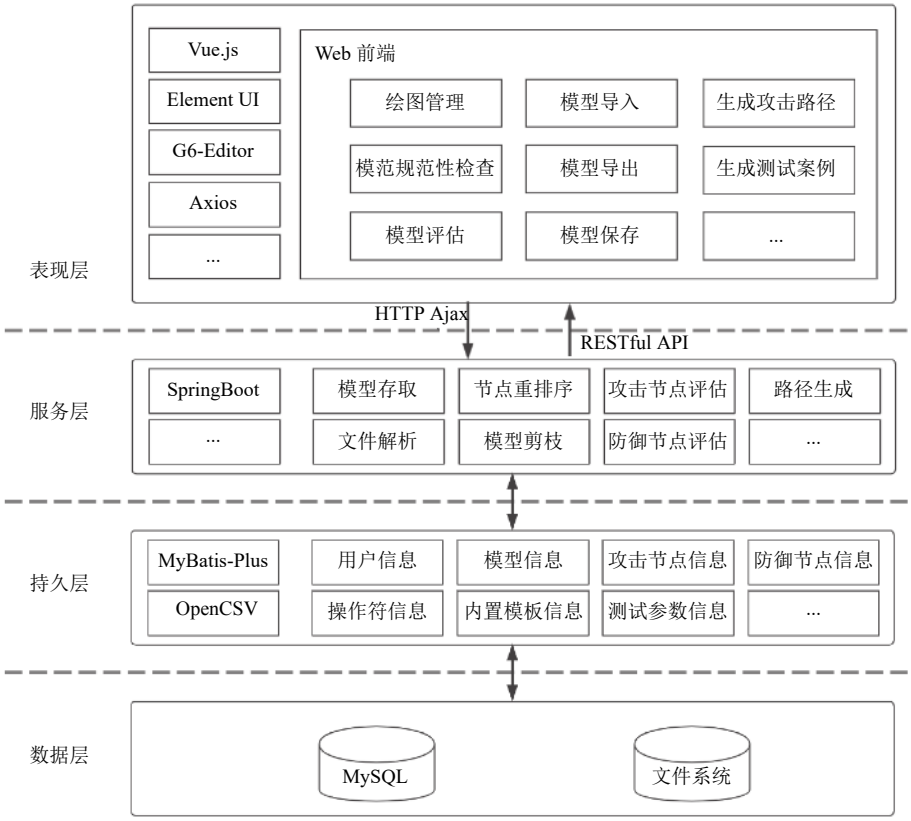


图 16 工具架构

服务层: 该层主要提供模型存取、模型校验、节点评估、对文件进行解析等服务. 后端整体采用 Spring Boot 框架进行开发, 对对象实体和业务逻辑进行封装, 并提供了核心业务的 RESTful 接口, 方便表现层进行调用.

持久层: 该层主要充当数据层与服务层间交互的媒介, 将工具涉及的对象实体信息转化为底层数据. 持久化存储采用 MyBatis-Plus 将数据存储到 MySQL 数据库、采用 OpenCSV 将数据转化为 CSV 文件存储到文件系统.

数据层: 该层描述底层的数据存储依赖, 用户和模型中的数据均存放于 MySQL 数据库的表中. 同时, 也可根据用户需要以 XML、CSV 文件等形式直接存储到本地文件系统.

本工具是一个前后端分离的 Web 应用. 前端采用 Vue 框架结合 Element UI、G6-Editor 等组件进行开发实现, 后端采用 Spring Boot 框架将实体和业务逻辑进行封装, 并以 MySQL 和文件系统作为持久化机制进行实现. 其中, 前后端采用 RESTful 风格进行交互.

工具功能主要包括模型构建模块、模型规范性检查模块、模型节点评估模块、路径和测试用例生成模块以及模型管理模块, 核心测试生成模块效果图如图 17、图 18 所示. 在测试生成模块中, 用户通过攻击路径生成功能, 确定模型对应的攻击路径列表. 用户可以选择某条攻击路径, 对路径中的节点依次设置测试参数列表, 通过测试用例生成功能得到路径转换的测试用例. 测试生成模块利用工具提供的攻击路径生成算法和测试用例生成策略, 能够将模型转化为一系列攻击路径和测试用例, 辅助用户对系统威胁进行检测.

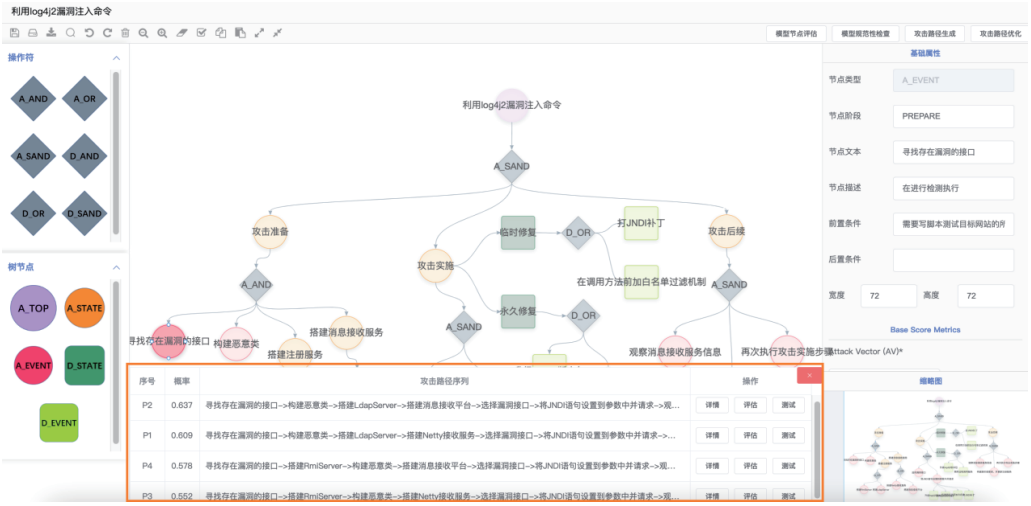


图 17　攻击路径生成效果图

生成测试用例

名称	测试参数列表
寻找存在漏洞的接口	[["调用扫描脚本查询", "系统域名", "获取存在log4j2漏洞的Api列表"], ["手动点击系统页面的提交、搜索、查询等按钮", "", ""]]
构建恶意类	[["构造恶意类Test.class（查询对方系统的文件信息）", "", ""]]
搭建LdapServer	[["搭建LDAP服务", "ip: {ldapServerIp}，port: {ldapServerPort}", "LDAP服务搭建成功"]]
搭建消息接收平台	[["搭建SpringBoot项目MsgServer，提供接收消息的接口/api/receiveMessage", "ip：{msgServerIp}，port：{msgServerPort}", "消息接收平台搭建成功"]]
选择漏洞接口	[["从log4j2漏洞的Api列表中随机选取一个Api", "", ""]]
将JNDI语句设置到参数中并请求	[["往漏洞targetApi的参数param中注入JNDI命令语句", "curl http://{targetIp}:{targetPort}/{targetApi} -H 'param: ${jndi:ldap://{ldapServerIp}:{ldapServerPort}/test}'", ""]]
观察消息接收服务信息	[["查看MsgServer项目/api/receriveMessage接口是否显示被攻击系统的文件信息", "", "成功收到被攻击系统的文件信息"]]
构建新的恶意类，并更新注册服务	[]
再次执行攻击实施步骤	[]

(a) 路径 P2 中攻击节点的测试参数

标头	信息
用例编号	路径P2-TC
测试名称	测试攻击路径P2
前置条件	1、需要写脚本测试目标网站的所有接口 2、需要构建恶意类 3、需要搭建LDAP服务 4、需要搭建消息接收服务
操作步骤	1、调用扫描脚本查询//输入：系统域名; 或者 手动点击系统页面的提交、搜索、查询等按钮 2、构造恶意类Test.class（查询对方系统的文件信息） 3、搭建LDAP服务//输入：ip: {ldapServerIp}，port: {ldapServerPort} 4、搭建SpringBoot项目MsgServer，提供接收消息的接口/api/receiveMessage//输入：ip：{msgServerIp}，port：{msgServerPort} 5、从log4j2漏洞的API列表中随机选取一个API 6、根据漏洞targetApi的参数param，注入JNDI命令语句//输入：请求: http://{targetIp}:{targetPort}/{targetApi}, param值：${jndi:ldap://{ldapServerIp}:{ldapServerPort}/test} 7、查看MsgServer项目/api/receriveMessage接口是否显示被攻击系统的文件信息
预期结果	1、获取存在log4j2漏洞的Api列表 2、LDAP服务搭建成功 3、消息接收平台搭建成功 4、成功收到被攻击系统的文件信息
测试结果	

(b) 生成的测试用例

图 18　测试用例生成效果图

5 案例研究及分析

本节应用本文所提出的测试用例生成框架对业界广泛关注的极高危风险进行案例研究, 验证本框架和工具可以有效地从威胁生成测试.

5.1 案例描述

在 2021 年 12 月 9 日, 国内多家安全公司检测到业界流行的 Java 日志库 log4j2 存在远程代码执行漏洞. 该漏洞允许未经身份验证的远程代码执行 (remote code execute, RCE), 影响 2.0 至 2.14.1 版本. 该漏洞利用原理为攻击者在请求系统业务接口时, 通过往接口参数中设置 JNDI 请求, 使 log4j2 组件在记录日志时, 通过内置的解析引擎, 解析执行参数中的 JNDI 请求, 进而达到对象注入的目的. 此外, 据调查发现, 目前业界大量的开源组件均使用 log4j2 组件, 如 Spring-Boot-strater-log4j2、Redis、Flink、Kafka 等, 所以该漏洞的影响面极广. 正因此, CVSS 官网根据其利用的影响 (完全服务器控制), 以及利用的程度, 给该漏洞打出了满分 10 分的评分, 编号为 CVE-2021-44228, 众多媒体也将该漏洞形容成“史诗级”“核弹级”漏洞, 可见该漏洞的影响相当严重, 被定义为极高危漏洞. 因此, 基于本工具对 log4j2 威胁进行分析并生成测试, 以对该威胁进行检测并及时处理, 表明本工具可以有效运用于实际场景, 帮助业界分析和检测威胁.

5.2 案例研究过程

案例研究使用本 DAT 工具对 log4j2 威胁进行分析. 攻击树模型和测试用例生成流程的相关知识可以从工具帮助页面获得, 如图 19 所示. 图 19 中给出了可视化模型及其元素的标注, 其中, 攻击节点均由椭圆形表示, 不同颜色代表不同层级, 紫色表示顶层攻击节点, 淡黄色表示状态层攻击节点, 红色表示事件层攻击节点; 防御节点均由矩形表示, 由于防御节点必须依附攻击节点, 没有顶层节点概念, 因此防御节点的颜色只有两种, 即浅绿色表示状态层防御节点, 绿色表示事件层防御节点; 同类别节点的关联通过实线表示, 防御节点和攻击节点的关联通过虚线表示; 橙色框标识节点所在的阶段; 菱形表示操作符, 框内内容标识了操作符的类型. 图 19 中还给出了攻击节点以及防御节点的含义和属性、CVSS 评估指标信息、操作符包括攻击节点和防御节点关联操作符类型.

图 19 工具帮助页面

● 模型构建. 基于相似的内置模板构建 log4j2 威胁的攻击树模型. 通过加载内置模板功能, 加载相似的模板; 修改节点的属性信息, 包括节点的描述、阶段、评估指标等信息; 补充节点或操作符, 通过拖拽左侧元素面板类中的节点、操作符加入到画布中, 修改节点信息并进行连线. 为选择相似的内置模板, 通过查看 CVE-2021-44228 的详细分析报告, 发现 log4j2 威胁对应的 CWE 标号为 CWE-502 不可信数据的反序列化 (deserialization of untrusted

data), 其中 CWE-502 的 Related Attack Patterns 属性字段为 CAPEC-586 对象注入 (object injection), 即 log4j2 威胁是一种对象注入攻击模式. 而在 CAPEC 中, 对象注入的层级为元模式 (Meta), 因此 log4j2 威胁可以在对象注入模板的基础上, 进行拓展. 同时, log4j2 威胁的原理是利用 JNDI 远程调用漏洞, 而在此之前, 同为 Java 类的日志组件 logback 也产生过类似漏洞, 同为一种对象注入攻击模式, 其 CVE 标号为 CVE-2021-42550, 因此为进一步节省构建 log4j2 威胁的攻击树模型, 本案例分析基于 logback JNDI 远程执行漏洞威胁的攻击树模型作为基准模板, 快速生成 log4j2 威胁的攻击树模型. 其中, 加载的 logback JNDI 远程执行漏洞威胁的攻击树模型如图 20 所示.

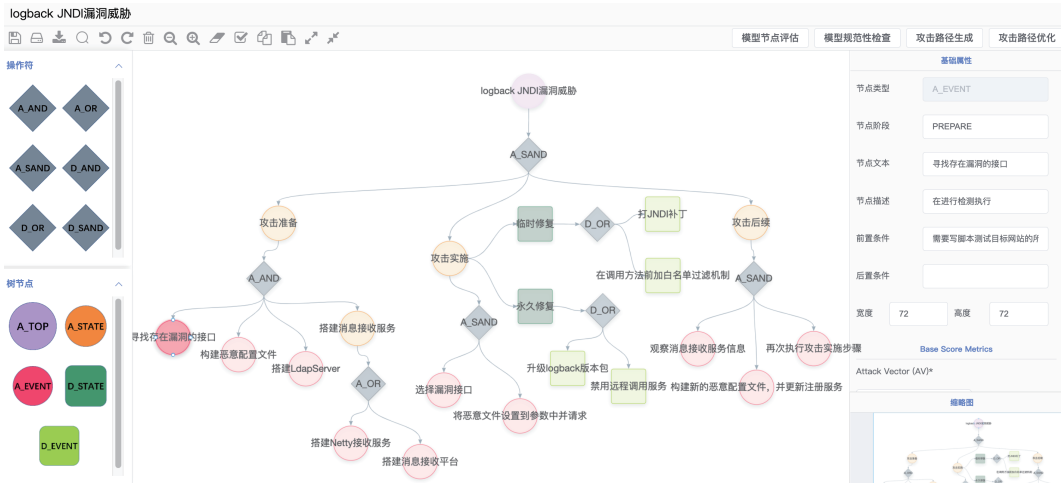


图 20 logback JNDI 远程执行漏洞威胁的攻击树模型

logback JNDI 远程执行漏洞威胁源于未对用户输入做有效的过滤, 攻击者可利用该漏洞精心设计恶意配置允许执行从 LDAP 服务器加载的任意代码. JNDI 是一种标准的 Java 命名系统接口, 提供统一的客户端 API, 为开发人员提供了查找和访问各种命名和目录服务的通用、统一的接口. 攻击者需要寻找存在漏洞的接口, 构造恶意配置文件, 替换被攻击系统的 logback 配置, 其中包含 JNDI 注入语句, 搭建 LDAP 服务器以允许使用 LDAP 读取或存储恶意配置文件中的信息, 搭建消息接收服务便于查看被攻击的系统是否成功执行 JNDI 注入命令, 将恶意文件设置到参数中并提交文件请求以后可以利用该漏洞攻击系统, 该漏洞可以采用打补丁、在调用方法前加白名单过滤机制对用户输入的内容进行过滤、升级 logback 版本包和禁用远程调用服务方案进行防御. 从图 20 能够看到, 模型的根节点为“logback JNDI 漏洞威胁”, 根结点按照攻击生命周期划分为攻击准备 PREPARE、攻击实施 ACTION、攻击后续 FOLLOW_UP 这 3 大阶段, 为了达到根节点所代表的威胁目标, 则需要按顺序依次执行这 3 大阶段中完整的攻击操作, 阶段的语义通过 A_SAND 操作符进行强约束, 下面具体介绍该模型各阶段的信息.

首先, 在攻击准备阶段中, 需要进行“寻找存在漏洞的接口”“构造恶意配置文件”“搭建 LdapServer”“搭建消息接收服务”这 4 种准备工作, 由于准备工作没有严格的顺序关系, 因此只需由 A_AND 操作符进行关联. 其中, “寻找存在漏洞的接口”“构造恶意配置文件”和“搭建 LdapServer”为不可被划分的操作, 被定义为 A_EVENT 节点类型, 而“搭建消息接收服务”可被划分为“搭建 Netty 接收服务”和“搭建消息接收平台”两种广为采用的消息接收服务方式, 定义为 STATE(A_STATE) 节点类型, 并由 A_OR 操作符进行关联. 其中, 搭建消息接收平台可以采用 SpringBoot 进行开发, 编写接收消息的 Controller 控制器, 并对外提供接收消息的 API 接口. 可在画布的右侧对节点设置属性信息, 图 20 右侧显示的是“寻找存在漏洞的接口”节点的基础属性信息, 其前置条件为“需要写脚本测试目标网站的所有接口”, 节点阶段为“PREPARE”. 其次, 在攻击实施阶段中, 该阶段被划分为“选择漏洞接口”和“将恶意文件设置到参数中并请求”两步具体操作. 由此看出, 对于攻击者而言, 利用 logback JNDI 漏洞实施攻击操作极为简单. 模板中对该攻击进行防御的方案主要有两种, 分别为“临时修复”方案和“永久修复”方案, 由于这两种方案还可被划分, 因此均被定义为 STATE(D_STATE) 节点类型. 其中, 在“临时修复”方案中, 可划分为“打 JNDI 补

丁”和“在调用方法前加白名单过滤机制”两种具体的防御方案, 同理在“永久修复”方案中, 也可划分为“升级 logback 版本包”和“禁用远程调用服务”两种, 这些具体的防御方案达到的效果是等价的, 均由 D_OR 操作符进行关联, 并定义为 D_EVENT 类型. 最后, 在攻击后续阶段中, 只需要在该阶段“观察消息接收服务信息”, 查看被攻击的系统是否成功执行 JNDI 注入命令, 并根据接收消息做后续操作. 其中, 判断的依据为查看消息接收服务中是否有预期的返回结果, 如果有预期结果, 那么攻击者可以重新“构建新的恶意配置文件, 并更新注册服务”, 并“再次执行攻击实施步骤”进行新一轮的攻击. 利用 log4j2 的远程执行代码漏洞进行攻击, 攻击者需要寻找存在漏洞的接口, 构建恶意类, 类中包含攻击者编写的恶意代码, 可以通过 LDAP 和 RMI 两种远程调用方式搭建注册服务, 以允许通过 LDAP 或 RMI 协议下载恶意代码在本地执行, 将恶意类的地址 JNDI 语句设置到接口参数中并通过请求的方式执行, 使 log4j2 组件在记录日志时, 通过内置的解析引擎解析执行参数中的 JNDI 请求, 进而达到对象注入的目的. 搭建消息接收服务便于查看被攻击的系统是否成功执行 JNDI 注入命令. 通过对 log4j2 威胁的分析发现, 该攻击模式与 logback JNDI 漏洞威胁有些许不同, 主要分为 4 点, 如表 4 所示.

表 4　两种威胁的不同

不同点	具体描述
构建的恶意对象	利用log4j2的远程执行代码漏洞进行攻击, 需要构建的恶意对象为恶意类, 类中包含攻击者编写的恶意代码, 但不包含JNDI注入语句; 而利用logback漏洞则是构建恶意的配置文件, 替换被攻击系统的logback配置, 其中包含JNDI注入语句
JNDI利用方式	log4j2支持LDAP和RMI两种远程调用方式, 而logback仅支持LDAP一种
实施方式	log4j2威胁是通过请求的方式执行, 只要是请求操作均可; 而利用logback该漏洞的前提是必须是提交文件请求
防御方案	log4j2威胁的防御方案, 不再是升级logback版本包, 而是升级log4j2版本包

如图 21 所示, 根据表 4 的分析结果, 对 logback JNDI 远程执行漏洞威胁的攻击树模型进行修改, 得到 log4j2 威胁的攻击树模型. 图中红色方框标注出了两种威胁的不同以及修改之处, 模型的根节点变成“log4j2 威胁”, 同样分为攻击准备、攻击实施、攻击后续 3 大阶段, A_SAND 操作符表示这 3 大阶段按顺序执行. 根据表 4 中两种威胁构建的恶意对象和利用方式存在的区别, 对应到模型的位置, 攻击准备阶段的 4 个攻击操作的其中两个操作不同, 将 logback JNDI 远程执行漏洞威胁中的“构造恶意配置文件”和“搭建 LdapServer”节点分别修改为“构建恶意类”和“搭建注册服务”节点, 其中“搭建注册服务”是为了便于对 log4j2 威胁所支持的两种调用方式进行划分而抽象出的新节点, 该节点可进一步划分为“搭建 RmiServer”和“搭建 LdapServer”两种远程调用方式, 这两种方式其中任一种均可以完成“搭建注册服务”的操作, 因此用 A_OR 操作符进行关联. 根据表 4 可知两种威胁的实施方式存在不同, 攻击实施阶段 log4j2 威胁是通过请求的方式执行, 只要是请求操作均可, 而利用 logback 漏洞的前提必须是提交文件请求, 因此将 logback JNDI 漏洞模型中攻击实施阶段的“将恶意文件设置到参数中并请求”节点修改为“将 JNDI 语句设置到参数中并请求”节点. 根据表 4 中两种威胁的防御方案不同, 将防御方案中的“永久修复”中的“升级 logback 版本包”节点修改为“升级 log4j2 版本包”节点. 攻击后续阶段中由于 log4j2 威胁构建的恶意对象为恶意类, 而利用 logback 漏洞则是构建恶意的配置文件, 因此将“构建新的恶意配置文件, 并更新注册服务”节点修改为“构建新的恶意类, 并更新注册服务”节点.

● 模型规范性检查. 模型构建完毕后, 对模型进行规范性检查, 根据模型的定义约束、模型校验算法、节点重排序算法检查模型的规范性. 由于图 21 所示的攻击树模型基于模板而来, 因此是规范的, 工具无告警提示, 模型规范性检查通过.

● 模型节点评估. 设置攻击节点的 CVSS 指标, 确定攻击节点的发生概率, 以便后续用于评估攻击路径优先级; 选中评估的防御节点, 设置 FAHP 模糊判断矩阵指标, 根据 FAHP 方法确定各防御节点的收益性, 以便选择收益性较高的防御方案, 为后续防御威胁提供参考.

下面以攻击节点“寻找存在漏洞的接口”为例, 介绍节点 CVSS 指标的设置依据. 如图 22 所示, 该节点的 CVSS 指标设置为 AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:L/E:P/RL:X/RC:U/CR:X/IR:X/AR:X. 由于该节点代表的操作通

过远程网络形式施行, 仅需低用户权限, 无需与其他用户进行交互, 所以设置 AV 为 N、PR 为 L、UI 为 N; 此外, 自该漏洞公布后, 业界厂商立即采取应急方案对漏洞进行修复, 所以该操作成功难度较大, 设置 AC:H; 与此同时, 该操作是一种查询行为, 不会影响系统其他组件和系统的可用性, 所以设置 S 为 U、A 为 L; 但是, 该操作会对系统机密性和完整性造成严重的损失, 所以设置 C 为 H、I 为 H; 同理, 鉴于 log4j2 漏洞的火爆程度, 目前业界已有指导代码可用, 但这些代码多为概念验证性代码, 对实际系统并不适用, 所以设置 E 为 P; 而且, 查询行为并未涉及漏洞的修复概念, 所以设置 RL 为 X; 查询报告的可信度为未知, 所以设置 RC 为 U; 最后, 由于查询操作并未影响系统的 IT 资产, 所以对于环境度量组的指标 CR、IR、AR 指标, 均设置为 X. 计算得到该节点的发生概率为 0.56.

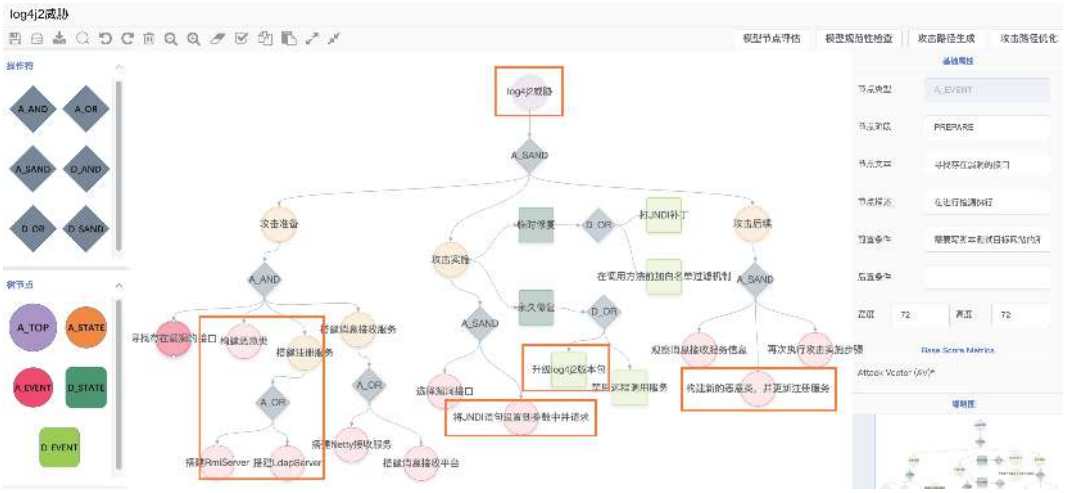


图 21 log4j2 威胁的攻击树模型

图 22 “寻找存在漏洞的接口”节点的发生概率

同理, 设置其他攻击节点的 CVSS 指标, 表 5 列出了 log4j2 威胁树模型中所有 A_EVENT 类型节点的 CVSS 评估描述和对应的发生概率.

表 5 A_EVENT 类型节点的发生概率

节点名称	CVSS评估描述	发生概率
寻找存在漏洞的接口	AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:L/E:P/RL:X/RC:U/CR:H/IR:H/AR:L	0.42
构建恶意类	AV:X/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L/E:F/RL:X/RC:X	0.53
搭建RmiServer	AV:X/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:L	0.26
搭建LdapServer	AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:L/E:P	0.63
搭建Netty接收服务	AV:L/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:L	0.4
搭建消息接收平台	AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:L/E:P	0.6
选择漏洞接口	AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	0.73
将JNDI语句设置到参数中并请求	AV:N/AC:H/PR:L/UI:N/S:C/C:H/I:H/A:H/E:F/RL:O/RC:C/CR:H/IR:H/AR:H	0.79
观察消息接收服务信息	AV:L/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:L	0.44
构建新的恶意类, 并更新注册服务	AV:L/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:H/CR:H/IR:H/AR:H	0.7
再次执行攻击实施步骤	AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H	0.87

选中模型中的 4 个防御节点, 并根据 FAHP 方法和 0.1–0.9 标度法设置模糊判断矩阵, 计算各方案的收益得分, 如图 23 所示, 本次评估从成本、难度、能力这 3 方面进行考虑. 由于 log4j2 威胁的影响面巨大, 所以在此优先选择防御能力最高的方案, 其次考虑成本, 最后才考虑方案实施的难度. 因此, 在设置指标的模糊判断矩阵时, 重要程度的比重为: 能力> 成本> 难度, 如图 23 所示, 能力、成本、难度指标所占权重为 50%, 36.7%, 13.3%. 在得到指标的权重后, 接下来分别设置 4 种防御方案针对成本、难度、能力指标的模糊判断矩阵, 并计算每种方案在各个指标中的得分, 并基于该得分计算各方案收益总得分. 以满分 100 分为例, 从图 23 中能够看出“升级 log4j2 版本包”的收益得分最高, 为 31.87 分, 其次为“打 JNDI 补丁”, 为 27.52 分, 再其次为“禁用远程调用服务”, 为 22.96 分, 最后为“在调用方法前加白名单过滤机制”, 为 17.65 分. 因此, 如果用户根据后续生成的攻击路径和测试用例的指导, 检测出某系统中存在 log4j2 漏洞, 那么建议优先选择“升级 log4j2 版本包”的防御方案, 对该漏洞进行永久修复.

● 生成测试线索. 在评估完攻击节点的发生概率和防御节点的收益性后, 将模型转化为攻击路径序列. 得到图 21 转化的 4 条攻击路径序列, 如表 6 所示. 计算路径的发生概率, 选择最高发生概率的攻击路径 P2, 为该路径中的攻击节点绑定测试参数, 并生成测试用例.

生成测试用例. 如图 24 所示, 为路径 P2 中的攻击节点绑定测试参数信息并生成测试用例. 以节点“寻找存在漏洞的接口”为例, 该节点设置了两种不同的测试参数信息, 一种偏自动化, 通过调用扫描脚本对系统域名的所有接口进行扫描, 另一种偏手工, 人工手动点击系统页面的提交、搜索、查询等关键按钮, 来尝试发现存在 log4j2 漏洞的 API. 在对节点设置完测试参数后, 点击“生成测试用例”按钮, 工具会对攻击路径 P2 和设置的测试参数进行解析, 生成路径 P2 对应的测试用例. 测试人员可以根据生成的测试用例对系统进行检测, 查看系统中是否存在 log4j2 威胁. 如果威胁存在, 那么用户可以参考步骤 4 中推荐的防御方案, 对系统进行安全防护. 用户可以将建模结果进行保存或导出, 方便模型重用.

5.3 案例研究结果和分析

案例研究对极高危风险 log4j2 威胁进行分析生成测试, 验证了本文所提出的框架的有效性. 研发的 DAT 工具能够对接威胁, 支撑用户的可视化建模需求, 支撑复杂设计的模型结构; 能够对用户构建的模型进行规范性检查, 验证模型的规范性, 为后续模型节点评估、攻击路径生成等操作提供模型正确性支撑; 能够评估攻击节点的发生概率, 为后续攻击路径评估提供支撑, 还能够评估防御节点的收益性, 为用户选择防御方案提供指导; 能够将模型转换为攻击路径序列, 能够为路径中的节点设置测试参数, 将路径转换为测试用例; 能够对模型进行保存或导出, 提高模型的可复用性和扩展性. 通过以上分析, 本研究提出的框架和工具可以有效地从威胁生成测试用例, 支持威胁分析、测试设计等安全实践.

计算防御收益性

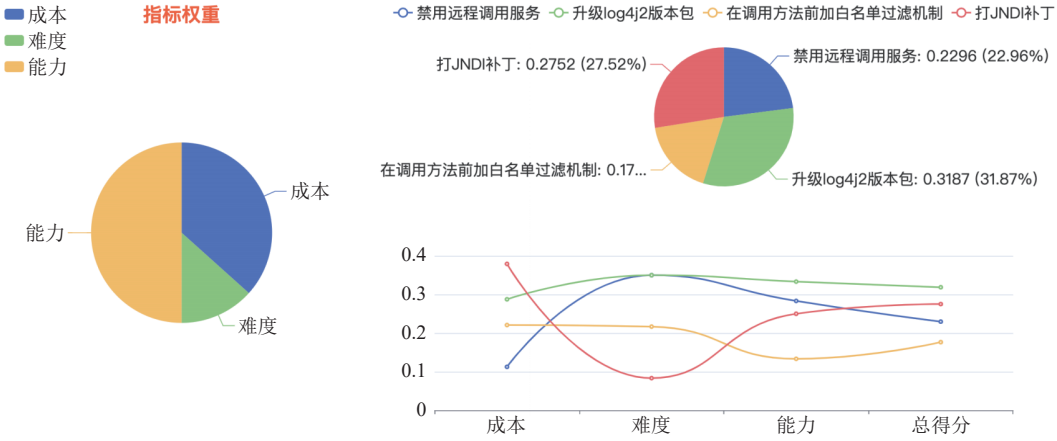
指标判断矩阵	成本	难度	能力
成本	0.5	0.75	0.35
难度	0.25	0.5	0.15
能力	0.65	0.85	0.5

成本判断矩阵	禁用远程调用服务	升级log4j2版本包	在调用方法前加白名单过滤机制	打JNDI补丁
禁用远程调用服务	0.5	0.3	0.2	0.1
升级log4j2版本包	0.7	0.5	0.6	0.35
在调用方法前加白名单过滤机制	0.5	0.4	0.5	0.35
打JNDI补丁	0.9	0.65	0.65	0.5

难度判断矩阵	禁用远程调用服务	升级log4j2版本包	在调用方法前加白名单过滤机制	打JNDI补丁
禁用远程调用服务	0.5	0.5	0.7	0.9
升级log4j2版本包	0.5	0.5	0.7	0.9
在调用方法前加白名单过滤机制	0.3	0.3	0.5	0.7
打JNDI补丁	0.1	0.1	0.3	0.5

能力判断矩阵	禁用远程调用服务	升级log4j2版本包	在调用方法前加白名单过滤机制	打JNDI补丁
禁用远程调用服务	0.5	0.3	0.8	0.6
升级log4j2版本包	0.7	0.5	0.7	0.6
在调用方法前加白名单过滤机制	0.2	0.3	0.5	0.3
打JNDI补丁	0.4	0.4	0.7	0.5

(a) 为防御节点设置模糊判断矩阵



(b) 计算防御方案收益得分

图 23 防御节点收益性

表 6 图 21 攻击树模型转化的攻击路径序列

序号	发生概率	攻击路径序列/测试线索
P1	0.609	寻找存在漏洞的接口 → 构建恶意类 → 搭建LdapServer → 搭建Netty接收服务 → 选择漏洞接口 → 将JNDI语句设置到参数中并请求 → 观察消息接收服务信息 → 构建新的恶意类, 并更新注册服务 → 再次执行攻击实施步骤
P2	0.637	寻找存在漏洞的接口 → 构建恶意类 → 搭建LdapServer → 搭建消息接收平台 → 选择漏洞接口 → 将JNDI语句设置到参数中并请求 → 观察消息接收服务信息 → 构建新的恶意类, 并更新注册服务 → 再次执行攻击实施步骤
P3	0.578	寻找存在漏洞的接口 → 搭建RmiServer → 构建恶意类 → 搭建消息接收平台 → 选择漏洞接口 → 将JNDI语句设置到参数中并请求 → 观察消息接收服务信息 → 构建新的恶意类, 并更新注册服务 → 再次执行攻击实施步骤
P4	0.552	寻找存在漏洞的接口 → 搭建RmiServer → 构建恶意类 → 搭建Netty接收服务 → 选择漏洞接口 → 将JNDI语句设置到参数中并请求 → 观察消息接收服务信息 → 构建新的恶意类, 并更新注册服务 → 再次执行攻击实施步骤

相关工作中, Wang 等人[15]提出了一种威胁模型驱动的安全测试方法, 用于在运行时检测不良威胁行为. 对安全策略的威胁使用 UML 时序图建模. 从威胁模型中导出消息序列, 并从每个消息序列中导出威胁跟踪. 测试程序代码时, 可以使用威胁模型作为指导对其进行检测, 以记录运行时与威胁场景相关的方法调用和方法执行的痕迹, 如果执行跟踪是威胁跟踪的一个实例, 则会报告安全违规. 将运行时执行跟踪与威胁跟踪相匹配, 以确定威胁是否

仍然存在. 其关键技术是验证 UML 时序图中描述的威胁规范与代码实现之间的一致性, 将程序的执行视为方法调用和方法执行的事件序列, 如果任何事件序列在威胁模型中调用了可能的事件序列, 那么模型中的威胁仍然存在于代码中, 并且将生成错误消息以报告故障. 该研究实质是将访问控制场景的威胁模型 UML 时序图中的威胁场景作为指导, 通过事件跟踪匹配的方式检测模型威胁是否存在于代码中, 从而检测安全故障. Masood 等人[21,22]研究了一种基于状态的方法来测试生成基于角色的访问控制 (RBAC) 策略. 他们的方法首先构建 RBAC 策略的有限状态模型, 然后从状态模型驱动测试. 通过状态模型对 RBAC 规范的结构和行为进行建模, 使用该模型为相应的实现生成测试套件, RBAC 规范的结构和行为代表实现的预期行为, 检查实现所表现的行为是否符合 RBAC 指定的预期行为, 针对策略规范中可能的约束/规则验证实现. 以上研究均是对代码进行测试, 检查实现中的行为是否与访问控制规范一致, 看其是否违背安全策略, 相比之下, 本文基于微软的 STRIDE 威胁建模方法, 其中使用数据流图对业务场景进行建模, 更加贴近业务需求, 识别出场景中系统可能存在的安全威胁之后, 使用树型模型对达成威胁的攻击途径进行结构化分析, 系统地从威胁模型生成安全测试. 在开发早期识别威胁并生成测试, 指导安全测试设计, 从设计阶段过渡到测试阶段. Martin 等人[18−20]研究了从用 XACML (OASIS 可扩展访问控制标记语言) 编写的访问控制策略规范生成测试的技术. XACML 是 OASIS 设计的语言规范标准, 可用于编写访问控制策略, 表示特定领域的访问控制策略、访问请求和访问响应, 运用软件测试技术发现基于 XACML 编写的策略规范中的错误, 确保策略规范的正确性. 他们定义了策略覆盖标准[18]和 XACML 访问控制策略[19]的变异测试框架, 为了从策略规范生成测试, 他们将输入整合到变化影响分析工具[20], 给定两个策略, 变化影响分析工具会对这两个策略的不同反应进行评估并给出反例, 根据这些反例生成请求. 在策略测试中, 测试输入是访问请求, 测试输出是访问响应, 根据策略变化生成请求, 主要是用于检查策略规范的正确性. 而本文的工作主要是对系统潜在的威胁通过威胁模型进行结构化分析并生成测试.

Xu 等人[12]提出了一种通过使用表示为 Predicate/Transition 网络的形式化威胁模型来自动生成安全测试的方法, 以网状结构对系统功能行为进行形式化建模, 分析系统功能行为中存在的安全威胁, 测试出的威胁会有遗漏现象, 而本文的测试对接威胁建模识别出的威胁, 具体而言是通过对系统业务场景进行威胁建模识别出潜在的威胁, 然后以识别出的威胁作为指导, 通过本文提出的框架和工具生成测试以检测威胁是否存在, 对系统安全威胁的识别和测试更为全面和系统. 网络的形式化威胁模型对专业知识要求较高, 本文采用的树型结构直观易用, 并且可以对接威胁建模识别的威胁. Wysopal 等人[10]介绍了基于风险的安全测试, 从开源库漏洞树中学习知识, 识别安全缺陷, 但是并没有从攻击树系统地生成安全测试. 作者介绍了一些抽象的安全测试策略, 大致概述了威胁建模的过程, 收集信息枚举应用程序的攻击面, 通过威胁建模分析风险行为, 基于数据流图识别威胁路径, 基于高风险的组件查找存在组件中的漏洞, 并没有实际提出如何生成测试. Marback 等人[11]基于原始攻击树模型生成安全测试序列, 从原始攻击树生成测试较为抽象, 不便于实际场景中具体安全测试使用, 没有实际对接到威胁建模的分析结果, 未提供从威胁建模到具体测试用例生成的完整流程和工具, 也没有提供数据支持, 没有考虑攻击途径较多时如何高效检测到威胁, 并针对性地防御和缓解威胁. 因此, 难以运用于工业界进行安全测试并改进安全设计. 本文提供改进的攻击树模型, 基于该模型提出了从威胁模型生成测试用例的完整框架, 并实现了工具原型. 通过模型构建对接威胁建模分析的结果, 对构建的模型进行规范性检查以提高生成测试的正确性, 提供详细信息, 具体贴合实际场景测试, 通过本文提出的框架和实现的工具可以生成具体的测试用例, 通过对生成的测试评估优先级以提高测试效率, 检测威胁的同时防御和缓解威胁, 指导后续的安全开发和测试设计, 将安全技术更好地嵌入到软件设计和开发之中. 本文提供了 Web 安全领域常见威胁的测试数据, 可以用于系统分析 Web 安全威胁测试生成. 总体而言, 相较于其他相关研究, 本文提供的测试用例生成框架更加完整, 生成的测试更为具体, 模型校验使得测试正确性更有保证, 衔接威胁建模识别的潜在威胁, 识别的威胁均可以通过该测试框架进行测试, 加上数据和工具的支持使得测试更加系统, 可运用于真实场景的安全实践, 便于在工业界落地.

本文提出了基于威胁模型生成测试的框架和其工具实现, 测试人员可以通过该框架和工具生成测试, 以设计阶段威胁建模分析的成果有效指导后期的安全测试. 后续测试用例是基于构建的威胁模型生成, 为了避免由于构建的威胁模型不完备而导致生成的测试用例不足的问题, 本文以直观可行的模型结构用于生成测试, 使用了基于

模板的模型构建方式, 提供了 Web 领域常见的结构化数据支持, 且数据较为全面. 同时, 本文还提供了文本构建和可视化构建方式作为补充, 测试人员可以采用多种构建方式相结合. 另外, 本文还提供了模型校验功能, 可以对构建的模型进行规范性检查. 如果测试人员漏掉某些细节, 可以在后续需要这些细节时进行补充和调整, 比如攻击节点的前置条件可以在生成测试用例时检查并进行修改. 通过对上述威胁建模各环节提供支持, 本文工作尽可能充分地对构建的模型的完备性提供保障. 未来计划进一步研究和完善如何更自动化智能化地构建威胁模型.

设置节点——寻找存在漏洞的接口的测试参数列表 确定 新增

序号	执行操作	输入参数	预期结果	操作
1	调用扫描脚本查询	系统域名	获取存在log4j2漏洞的Api列表	修改 删除
2	手动点击系统页面的提交、搜索、查询等按钮			修改 删除

生成测试用例

名称	测试参数列表
寻找存在漏洞的接口	[["调用扫描脚本查询", "系统域名", "获取存在log4j2漏洞的Api列表"], ["手动点击系统页面的提交、搜索、查询等按钮", "", ""]]
构建恶意类	[["构造恶意类Test.class（查询对方系统的文件信息）", "", ""]]
搭建LdapServer	[["搭建LDAP服务", "ip: {ldapServerIp}，port: {ldapServerPort}", "LDAP服务搭建成功"]]
搭建消息接收平台	[["搭建SpringBoot项目MsgServer，提供接收消息的接口/api/receiveMessage", "ip：{msgServerIp}，port：{msgServerPort}", "消息接收平台搭建成功"]]
选择漏洞接口	[["从log4j2漏洞的Api列表中随机选取一个Api", "", ""]]
将JNDI语句设置到参数中并请求	[["往漏洞targetApi的参数param中注入JNDI命令语句", "curl http://{targetIp}:{targetPort}/{targetApi} -H 'param: ${jndi:ldap://{ldapServerIp}:{ldapServerPort}/test}'", ""]]
观察消息接收服务信息	[["查看MsgServer项目/api/receriveMessage接口是否显示被攻击系统的文件信息", "", "成功收到被攻击系统的文件信息"]]
构建新的恶意类，并更新注册服务	[]
再次执行攻击实施步骤	[]

(a) 设置路径 P2 中攻击节点的测试参数

标头	信息
用例编号	路径P2-TC
测试名称	测试攻击路径P2
前置条件	1、需要写脚本测试目标网站的所有接口 2、需要构建恶意类 3、需要搭建LDAP服务 4、需要搭建消息接收服务
操作步骤	1、调用扫描脚本查询//输入：系统域名; 或者 手动点击系统页面的提交、搜索、查询等按钮 2、构造恶意类Test.class（查询对方系统的文件信息） 3、搭建LDAP服务//输入：ip: {ldapServerIp}，port: {ldapServerPort} 4、搭建SpringBoot项目MsgServer，提供接收消息的接口/api/receiveMessage//输入：ip：{msgServerIp}，port：{msgServerPort} 5、从log4j2漏洞的API列表中随机选取一个API 6、根据漏洞targetApi的参数param，注入JNDI命令语句//输入：请求: http://{targetIp}:{targetPort}/{targetApi}, param值：${jndi:ldap://{ldapServerIp}:{ldapServerPort}/test} 7、查看MsgServer项目/api/receriveMessage接口是否显示被攻击系统的文件信息
预期结果	1、获取存在log4j2漏洞的Api列表 2、LDAP服务搭建成功 3、消息接收平台搭建成功 4、成功收到被攻击系统的文件信息
测试结果	

(b) 生成测试用例

图 24　设置路径 P2 中攻击节点的测试参数并生成测试用例

6　总结与展望

威胁建模可以在软件开发早期阶段识别业务场景中系统潜在的安全威胁, 但是现有的威胁建模工具不具备生成测试的功能, 没有对威胁发生的原因、条件进行具体分析, 不能很好地指导测试威胁, 缓解威胁. 本文提出了一种基于威胁模型的测试用例生成框架, 并实现了工具原型, 可以从识别的威胁进一步生成测试, 能够辅助测试人员检测威胁, 进行安全测试并制定防御方案, 以便后期的测试能够很好地对接前期的威胁建模分析成果, 提高软件的安全性. 为了对实际攻击操作进行具体描述, 本文对传统的攻击树模型进行了改进, 以便生成的测试能够更好地贴

合实际场景. 本文提供了自动和手动构建模型的方式对威胁的攻击行为和手段进行建模, 对模型进行规范性检查, 以保证模型的正确性. 基于模型的攻击路径形成测试线索, 并进一步转换成测试用例. 案例研究表明本工具可以有效运用于实际场景中辅助分析和测试威胁.

目前, 本研究还存在一些有待改进之处, 例如, 攻击路径形成测试线索进一步转换成测试用例这一步骤中仍需要人工设置参数, 后续工作将研究如何自动转化并生成测试代码; 目前构建测试模型是基于模板的方式进行构建, 提供了文本构建和可视化构建方式作为补充, 后续工作将研究如何自动化构建测试模型; 增强工具自动化程度, 实现从设计阶段到测试阶段的平滑过渡.

References:

[1] Wu HQ. White Hats Talk About Web Security. Beijing: Electronic Industry Press, 2012 (in Chinese).

[2] Zheng MW. Research on security-critical software testing based on test coverage. Network Security Technology & Application, 2016(6): 106, 108 (in Chinese with English abstract). [doi: 10.3969/j.issn.1009-6833.2016.06.069]

[3] Tassey G. The economic impacts of inadequate infrastructure for software testing. Research Triangle Park: RTI, 2002.

[4] He XF. Security threat analysis and exploration on software design. Wireless Internet Technology, 2015, (20): 55–59 (in Chinese with English abstract). [doi: 10.3969/j.issn.1672-6944.2015.20.027]

[5] Shostack A. Threat Modeling: Designing for Security. John Wiley & Sons, 2014.

[6] Kohnfelder L, Garg P. The threats to our products, microsoft interface. 1999. http://blogs.msdn.com/sdl/attachment/9887486.ashx

[7] Johnstone MN. Threat modelling with stride and UML. In: Proc. of the 8th Australian Information Security Management Conf. Perth, 2012. 18–27.

[8] Schneier B. Secrets and Lies: Digital Security in a Networked World. New York: Wiley, 2000. 3.

[9] Budde CE, Kolb C, Stoelinga M. Attack trees vs. Fault trees: Two sides of the same coin from different currencies. In: Proc. of the 18th Int'l Conf. on Quantitative Evaluation of Systems. Paris: Springer, 2021. 457–467. [doi: 10.1007/978-3-030-85172-9_24]

[10] Wysopal C, Nelson L, Zovi DD, Dustin E. The Art of Software Security Testing: Identifying Software Security Flaws (Symantec Press). Boston: Addison-Wesley Professional, 2006.

[11] Marback A, Do H, He K, Kondamarri S, Xu DX. A threat model-based approach to security testing. Software: Practice and Experience, 2013, 43(2): 241–258. [doi: 10.1002/spe.2111]

[12] Xu DX, Tu MH, Sanford M, Thomas L, Woodraska D, Xu WF. Automated security test generation with formal threat models. IEEE Trans. on Dependable and Secure Computing, 2012, 9(4): 526–539. [doi: 10.1109/TDSC.2012.24]

[13] Murata T. Petri nets: Properties, analysis and applications. Proc. of the IEEE, 1989, 77(4): 541–580. [doi: 10.1109/5.24143]

[14] Malzahn D, Birnbaum Z, Wright-Hamor C. Automated vulnerability testing via executable attack graphs. In: Proc. of the 2020 Int'l Conf. on Cyber Security and Protection of Digital Services (Cyber Security). Dublin: IEEE, 2020. 1–10. [doi: 10.1109/CyberSecurity49315.2020.9138852]

[15] Wang LZ, Wong E, Xu DX. A threat model driven approach for security testing. In: Proc. of the 3rd Int'l Workshop on Software Engineering for Secure Systems. Minneapolis: IEEE, 2007. 10. [doi: 10.1109/SESS.2007.2]

[16] Hersén N, Hacks S, Fögen K. Towards measuring test coverage of attack simulations. In: Proc. of the 22nd Int'l Conf. on Enterprise, Business-process and Information Systems Modeling. Melbourne: Springer, 2021. 303–317. [doi: 10.1007/978-3-030-79186-5_20]

[17] Marksteiner S, Ramler R, Sochor H. Integrating threat modeling and automated test case generation into industrialized software security testing. In: Proc. of the 3rd Central European Cybersecurity Conf. Munich: ACM, 2019. 25. [doi: 10.1145/3360664.3362698]

[18] Martin E, Tao X, Yu T. Defining and measuring policy coverage in testing access control policies. In: Proc. of the 8th Int'l Conf. on Information and Communications Security. Raleigh: Springer, 2006. 139–158. [doi: 10.1007/11935308_11]

[19] Martin E, Xie T. A fault model and mutation testing of access control policies. In: Proc. of the 16th Int'l Conf. on World Wide Web. Banff: ACM, 2007. 667–676. [doi: 10.1145/1242572.1242663]

[20] Martin E, Xie T. Automated test generation for access control policies via change-impact analysis. In: Proc. of the 3rd Int'l Workshop on Software Engineering for Secure Systems. Minneapolis: IEEE, 2007. 5. [doi: 10.1109/SESS.2007.5]

[21] Masood A, Bhatti R, Gahfoor A, Mathur AP. Model-based testing of access control systems that employ RBAC policies. West Lafayette: Purdue University, 2005.

[22] Masood A, Ghafoor A, Mathur A. Scalable and effective test generation for access control systems that employ RBAC policies. West Lafayette: Purdue University, 2006.

[23] Zafar MN, Afzal W, Enoiu EP. Towards a workflow for model-based testing of embedded systems. In: Proc. of the 12th Int'l Workshop on Automating TEST Case Design, Selection, and Evaluation. Athens: ACM, 2021. 33–40. [doi: 10.1145/3472672.3473956]

[24] Qiu ZP, Wang LS, Kang JX, Gao ZJ, Hui W, Wei Y, Shen XY. A method of test case generation based on VRM model. In: Proc. of the 6th IEEE Int'l Conf. on Computer and Communication Systems (ICCCS). Chengdu: IEEE, 2021. 1099–1107. [doi: 10.1109/ICCCS52626.2021.9449261]

[25] Mihret Z, Liu LJ. Attack-driven test case generation approach using model-checking technique for collaborating systems. In: Proc. of the 2nd IEEE/ACM Int'l Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS). Madrid: IEEE, 2021. 1–8. [doi: 10.1109/EnCyCriS52570.2021.00008]

[26] Nayak G, Ray M. Model-based test sequence generation and prioritization using ant colony optimization. Journal of Information Technology Research, 2022, 15(1): 1–17. [doi: 10.4018/JITR.299946]

[27] Rocha M, Simão A, Sousa T. Model-based test case generation from UML sequence diagrams using extended finite state machines. Software Quality Journal, 2021, 29(3): 597–627. [doi: 10.1007/s11219-020-09531-0]

[28] Santiago D, Clarke PJ, Alt P, King TM. Abstract flow learning for Web application test generation. In: Proc. of the 9th ACM SIGSOFT Int'l Workshop on Automating TEST Case Design, Selection, and Evaluation. Lake Buena Vista: ACM, 2018. 49–55. [doi: 10.1145/3278186.3278194]

[29] Erdem I, Oguz RF, Olmezogullari E, Aktas MS. Test script generation based on hidden Markov models learning from user browsing behaviors. In: Proc. of the 2021 IEEE Int'l Conf. on Big Data. Orlando: IEEE, 2021. 2998–3005. [doi: 10.1109/BigData52589.2021.9671312]

[30] Grano G, Laaber C, Panichella A, Panichella S. Testing with fewer resources: An adaptive approach to performance-aware test case generation. IEEE Trans. on Software Engineering, 2021, 47(11): 2332–2347. [doi: 10.1109/TSE.2019.2946773]

[31] Schneier B. Attack trees: Modeling security threats. Dr Dobbs Journal, 1999, 24(12): 21–29.

[32] Nagaraju V, Fiondella L, Wandji T. A survey of fault and attack tree modeling and analysis for cyber risk management. In: Proc. of the 2017 IEEE Int'l Symp. on Technologies for Homeland Security (HST). Waltham: IEEE, 2017. 1–6. [doi: 10.1109/THS.2017.7943455]

[33] Zimba A, Chen HS, Wang ZS. Bayesian network based weighted APT attack paths modeling in cloud computing. Future Generation Computer Systems, 2019, 96: 525–537. [doi: 10.1016/j.future.2019.02.045]

[34] Daley K, Larson R, Dawkins J. A structural framework for modeling multi-stage network attacks. In: Proc. of the 2002 Int'l Conf. on Parallel Processing Workshop. Vancouver: IEEE, 2002. 5–10. [doi: 10.1109/ICPPW.2002.1039705]

[35] Yadav T, Rao AM. Technical aspects of cyber kill chain. In: Proc. of the 3rd Int'l Symp. on Security in Computing and Communications. Kochi: Springer, 2015. 438–452. [doi: 10.1007/978-3-319-22915-7_40]

[36] Hutchins EM, Cloppert MJ, Amin RM. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Leading Issues in Information Warfare & Security Research, 2011, 1(1): 80.

[37] Bryans J, Liew LS, Nguyen HN, Sabaliauskaite G, Shaikh S, Zhou FJ. A template-based method for the generation of attack trees. In: Proc. of the 13th IFIP WG 11.2 Int'l Conf. on Information Security Theory and Practice. Paris: Springer, 2019. 155–165. [doi: 10.1007/978-3-030-41702-4_10]

[38] Dong HY, Wang HW, Tao T. An attack tree-based approach for vulnerability assessment of communication-based train control systems. In: Proc. of the 2017 Chinese Automation Congress (CAC). Jinan: IEEE, 2017. 6407–6412. [doi: 10.1109/CAC.2017.8243932]

[39] Teixeira PG, Lopes VHL, dos Santos RP, Kassab M, Neto VVG. The status quo of systems-of-information systems. In: Proc. of the 7th IEEE/ACM Int'l Workshop on Software Engineering for Systems-of-Systems (SESoS) and the 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems. Montreal: IEEE, 2019. 34–41. [doi: 10.1109/SESoS/WDES.2019.00013]

[40] Jackson M. Engineering by software: System behaviours as components. In: Mazzara M, Meyer B. Present and Ulterior Software Engineering. Cham: Springer, 2017. 1–17. [doi: 10.1007/978-3-319-67425-4_1]

[41] Zhao C, Gao P. Research on the thread of test case generation. Computer and Digital Engineering, 2008, 36(10): 178–180 (in Chinese with English abstract). [doi: 10.3969/j.issn.1672-9722.2008.10.043]

[42] Li EX, Kang CQ, Huang DY, Hu MD, Chang FY, He LJ, Li XY. Quantitative model of attacks on distribution automation systems based on CVSS and attack trees. Information, 2019, 10(8): 251. [doi: 10.3390/info10080251]

[43] Allodi L, Banescu S, Femmer H, Beckers K. Identifying relevant information cues for vulnerability assessment using CVSS. In: Proc. of the 8th ACM Conf. on Data and Application Security and Privacy. Tempe: ACM, 2018. 119–126. [doi: 10.1145/3176258.3176340]

[44] Wang YM, Chin KS. Fuzzy analytic hierarchy process: A logarithmic fuzzy preference programming methodology. Int'l Journal of Approximate Reasoning, 2011, 52(4): 541–553. [doi: 10.1016/j.ijar.2010.12.004]

[45] Saaty TL, Kearns KP. The analytic hierarchy process. Analytical Planning, 1985. [doi: 10.1016/B978-0-08-032599-6.50008-8]

[46] Kubler S, Robert J, Derigent W, Voisin A, Le Traon Y. A state-of-the-art survey & testbed of fuzzy AHP (FAHP) applications. Expert Systems with Applications, 2016, 65: 398–422. [doi: 10.1016/j.eswa.2016.08.064]

[47] Wang P, Liu JC. Improvements of attack-defense trees for threat analysis. In: Proc. of the 2013 Int'l Computer Conf. on Advances in Intelligent Systems and Applications. Hualien: Springer, 2013. 91–100. [doi: 10.1007/978-3-642-35473-1_10]

[48] Ji X, Yu HQ, Fan GS, Fu WH. Attack-defense trees based cyber security analysis for CPSs. In: Proc. of the 17th IEEE/ACIS Int'l Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). Shanghai: IEEE, 2016. 693–698. [doi: 10.1109/SNPD.2016.7515980]

附中文参考文献:

[1] 吴翰清. 白帽子讲Web安全. 北京: 电子工业出版社, 2012.

[2] 郑明伟. 基于测试覆盖的安全关键软件测试研究. 网络安全技术与应用, 2016(6): 106, 108. [doi: 10.3969/j.issn.1009-6833.2016.06.069] [doi: 10.3969/j.issn.1009-6833.2016.06.069]

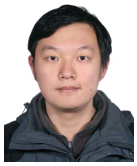
[4] 何欣峰. 软件设计安全威胁分析与探索. 无线互联科技, 2015, (20): 55–59. [doi: 10.3969/j.issn.1672-6944.2015.20.027]

[41] 赵翀, 高鹏. 基于线索的测试用例生成研究. 计算机与数字工程, 2008, 36(10): 178–180. [doi: 10.3969/j.issn.1672-9722.2008.10.043]

付昌兰(1990－), 女, 博士生, CCF 学生会员, 主要研究领域为软件安全, 威胁建模, 软件工程方法与理论.

李凤龙(1989－), 女, 硕士, 主要研究领域为网络安全, 云安全, 软件工程, 威胁建模.

张贺(1971－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件系统安全, 软件过程, 软件体系结构, 服务计算, 经验软件工程领域的科研和实践.



匡宏宇(1985－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为软件可追踪性, 程序理解, 文本分析.