中国科学院大学
University of Chinese Academy of Sciences

研究生学位论文开题报告

报告题目	面向二进制软件的复用漏洞检测技术研究		
学生姓名	王晓伟	学号	201818018670149
指导教师	霍玮	职称	研究员
学位类别	工学博士		
学科专业	网络空间安全		
研究方向	软件组成分析		
培养单位	中国科学院信息工程研究所		
填表日期	2022 年 7 月		

中国科学院大学制

报告提纲

一　选题的背景及意义

1.1　供应链安全事件

2021 年年末爆发的 Log4j 安全漏洞堪称互联网历史上破坏力最为惊人的漏洞之一，漏洞波及面和危害程度堪比 2017 年的“永恒之蓝”漏洞。从爆发至今，Log4j 漏洞影响的严重性、广泛性已经在各领域开始显现，并不断加大。

在美国网络安全审查委员会发布的首份报告《回顾 2021 年 12 月的 Log4j 事件》中明确指出 [1]，Log4j 是一款开源软件，开发者已经将其集成到数百万个系统中。这种无孔不入、无处不在的底层基础软件中的漏洞有能力影响全世界的公司和组织，也包括政府。Log4j 漏洞已成为一大“持续性流行漏洞”，将在未来多年引发持续引发风险，换言之，这种无所不在的软件库的未经修复版本，将在未来十年或更长时间内继续留存在各类系统当中，进而增加所处系统的安全风险。同时，美国网络安全审查委员会预测，鉴于 Log4j 的普遍存在，在未来十年中，易受攻击的版本仍将存在于实际运行的软件系统中，我们将看到利用漏洞的方式不断演进，所有组织都应具备发现和升级易受攻击软件的能力，以及长期维持这些漏洞管理的能力。

根据统计，有超过 35,863 个开源软件 Java 组件依赖于 Log4j，意味着超过 8% 的软件包里至少有一个版本会受此漏洞影响。漏洞在依赖链中越深，修复步骤就越多。根据云安全专家评估，每秒有超过 1000 次利用 Log4j 漏洞的尝试 [2]。Log4j 漏洞不仅影响直接使用该库的基于 Java 的应用程序和服务，还影响许多其他流行的依赖它的 Java 组件和开发框架，包括但不限于 Apache Struts2、Apache Solr、Apache Druid、Apache Flink、ElasticSearch 和 ApacheKafka。随着危机的持续发酵，此次 Log4j 漏洞带来的损失目前尚无法准确评估。

图 1 LOG4J 事件

2022 年 6 月，美国 CISA 发布警告 [3] 强调 Log4Shell 漏洞目前已经影响了 1800 多种产品，产品安全团队需要额外注意识别任何包含风险 Log4j 包的软件。当前，黑客仍在利用 Log4Shell 漏洞，专门针对未打补丁的、面向 Internet 的 VMware Horizon 和 Unified Access Gateway 服务器。在《2021 年终漏洞快速查看》报告中，CISA 强调了 Log4j 漏洞具有广泛影响的潜力，自报告发布以来，受影响的产品总数增加了 11.6%。随着继续跟踪漏洞，受 Log4j 漏洞影响的产品总数可能会增加。自 2021 年年底 Log4j 漏洞爆发以来，Mirai、Muhstik 等多个僵尸网络家族利用此漏洞进行传播。同时，该漏洞利用正在发生快速变异，绕过现有缓解措施，并吸引了越来越多的黑客攻击者。Check Point[4] 的网络安全研究人员警告说，Log4j 漏洞正在快速变异，已经产生 60 多个更强大的变种，所有变种都在不到一天的时间内产生。下图2为按照地区分别统计的 LOG4J 事件对企业网的影响 [5]。

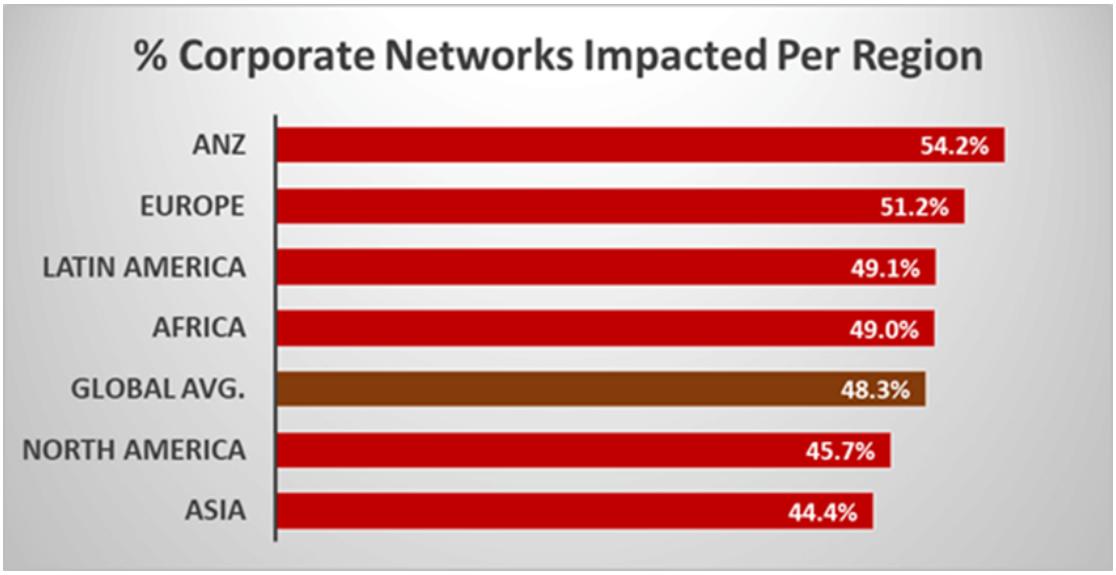

图 2 LOG4J 事件对洲际企业网的影响

2022 年 4 月以来，针对 VMware Horizon 服务器的 Log4j 攻击仍旧不断持续且有增无减。朝鲜黑客组织 Lazarus 一直通过 Log4j 远程代码执行漏洞，在未应用安全补丁的 VMware Horizon 虚拟桌面平台中大肆利用 Log4j 漏洞来部署勒索软件及其他恶意程序包 [6]。2021 年 12 月，比利时国防部网络最近受到不明攻击者的成功攻击，攻击者利用 Apache 日志库 log4j 的巨大漏洞实施攻击，国防部证实这次攻击是成功利用了 log4j 的漏洞。Log4j 漏洞在今后若干年长期存在，所有组织应具备发现和升级易受攻击软件的能力，以及长期维持这些漏洞管理的能力。所有组织都应继续主动监控和升级 Log4j 的易受攻击版本，优先应用软件升级，谨慎使用缓解措施，避免可能造成长期暴露的错误情况（例如，暴露易受攻击面的配置错误）。同时，运用成熟的业务流程来防止易受攻击版本的重新引入，采取基于风险的方法来补救 Log4j 漏洞，以便解决其他严重性漏洞。根据 Log4j 漏洞的严重性，所有企业都需要快速评估其业务运营的潜在风险，并制定和执行行动计划。当前对 Log4j 漏洞事件做出最有效响应的组织已经拥有技术资源和成熟的流程，可以识别易受攻击的产品资源、评估潜在风险。要降低 Log4j 和其他广泛使用的开源软件中的漏洞给生态系统带来风险的可能性，成熟可靠方法是确保代码的开发符合行业公认的安全编码实践，并由安全专家进行相应的审核。近年来，针对软件供应链的安全攻击事件一直呈快速增长态势 [7]，造成的危害也越来越严重，如下图所示。其中，开源软件的安全问题尤其值得关注。

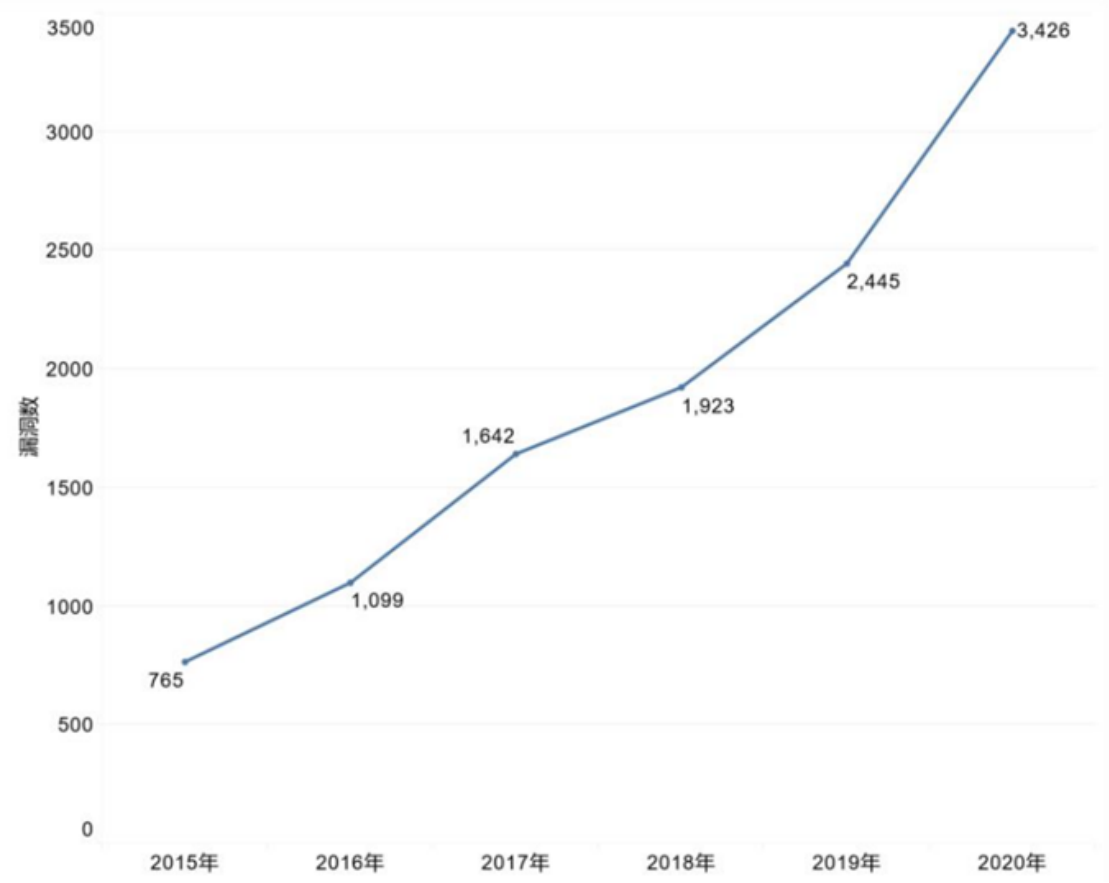

图 3 逐年上升的软件供应链安全事件

软件供应链安全事件愈演愈烈，下表归纳了最近年两年的典型事件 [8]，但这只是冰山一角。不难看出，供应链攻击可谓无处不在，在软件生命周期的各个环节中、软件产品的各种元素上都可能发生。

1.2 软件供应链安全

软件供应链 [9] 指的是软件从软件供应商到用户使用的整个过程中，从设计软件、编写代码到生成软件的开发环节，再到分发软件和用户下载的交付环节，最终到用户使用环节的三大环节组成的链状结构。软件供应链攻击指的是攻击者充分利用供应链的安全漏洞，在合法软件的开发、交付、使用中进行劫持或者篡改，并依赖供应链上的信任关系以逃避传统安全产品的检查，沿着供应链向后渗透，从而实施对目标的网络渗透及非法攻击。软件供应链安全有以下特点：

1) 攻击影响范围广

由于软件供应链是一个完整的流动过程，因此在软件供应链上发生的攻击具有扩散性。对于普通的安全漏洞的攻击，其一般发生在单点上，传播能力有

表 1 近两年典型的软件供应链安全事件

NO.	Name	Year	Type of compromise
1	PEAR PHP Package Manager compromise	2022	Dev Tooling
2	npm Library ”node-ipc” Sabotaged with npm Library ”peacenotwar” in Protest by their Maintainer	2022	Malicious Maintainer
3	npm Libraries ”colors” and ”faker” Sabotaged in Protest by their Maintainer	2022	Malicious Maintainer
4	GCP Golang Buildpacks Old Compiler Injection	2022	Source Code
5	WordPress theme publisher compromised	2022	Source Code
6	Remote code injection in Log4j	2021	Source code
7	Compromise of NP packages coa and rc	2021	Malicious Maintainer
8	Compromise of ua-parser-js	2021	Malicious Maintainer
9	The klow / klown / okhsa incident	2021	Negligence
10	PHP self-hosted git server	2021	Dev Tooling
11	Homebrew	2021	Dev Tooling
12	Codecov	2021	Source Code

限。相比之下，在软件供应链上游开发环节发生的攻击，一旦成功，便会波及整个软件供应链的中下游，对大量的软件供应商和最终用户造成影响。除此之外，随着软件的规模变得越来越大，软件的程序逻辑也变得越来越复杂，理解软件的完整语义和掌握操作逻辑会变得愈发困难。因此，在供应链开发环节，设计缺陷和深层漏洞更难被发现，源代码和开发工具更容易受到外部污染，厂商预留和第三方组建的后门将会变得更容易隐藏。

2) 攻击检测难度大、持续性强

由于软件供应链攻击依赖供应链上的信任关系，以逃避传统安全产品的检查，因此软件供应链攻击具有高度隐藏性。一般来说，经过官方认证的软件添加后门，大部分供应链攻击会受到“合法软件”保护。在不被暴露的情况下，病毒可以不断地访问并攻击新的目标。同时，大多数企业和公司往往认为自己不会在软件供应链的开发、交付环节中被当作攻击的目标，会默认认为供应链上的软件是安全的，因此很难从单一角度检测到攻击行为。除此之外，不少软件供应链攻击者不直接攻击供应商，而是利用供应商来规避公司的网络安全机制检测风险。因此，想要从根源上就检测出攻击行为十分困难。

3) 攻击手法多种多样，数量不断上升

当前，全球互联网环境日益复杂，软件供应链的可攻击面也越来越多，从而衍生出多种多样的攻击手法，这也大大提升了供应链攻击事件的数量。前面已经提到，在软件供应链的开发、交付、使用环节都有各种各样的供应链攻击手法。在开发和交付阶段，开发人员与供应商会随时因为各种原因使软件供应链受到攻击，而站在的最终用户的视角，从供应链的下游向前看各环节，信息的能见度和安全的可控程度逐渐下降，对供应链上游和中游发生的问题，更是鞭长莫及。因此，攻击者只需要针对软件供应链各个环节的脆弱点实施攻击，就可以入侵并造成危害，即使企业拥有足够的安全意识，从发现安全隐患，到解决问题并发布新版本，仍然需要很长的周期，在这期间，攻击者根据漏洞生成的攻击速度会更快。

1.3 软件成分分析（SCA）

软件成分分析 Software Compostition Analysis（SCA）是一种用于管理开源组件应用安全的方法 [10]。通过 SCA，开发团队可以快速跟踪和分析引入项目

的开源组件。同时，SCA 工具可以发现所有相关组件、支持库以及它们之间直接和间接依赖关系。SCA 工具还可以检测软件许可证、已弃用的依赖项以及漏洞和潜在威胁。扫描过程会生成物料清单 Bill of Materials（BOM），从而提供项目软件资产的完整清单。

图 4 软件开发生命周期

SCA 的目标是第三方基础组件/可执行程序/源代码等类型的以二进制形式存储的文件，包括但不限于源代码片段或 Package，可执行的二进制组件/程序，基础 lib，tar/tgz 压缩文件，镜像/镜像层，广义的软件构建过程等等。因此，所有以二进制形式存储的文件皆可以是其分析目标。SCA 的方法不局限于具体的软件开发语言技术栈，即不关注是 Java、Python、C/C++、Golang 或者是 Node，还是 Docker（OCI）Image，或者是广义的构建过程，而是从文件层面关注目标的各组成文件本身，以及文件与文件之间的关联关系以及彼此组合成目标的过程细节。简而言之，SCA 是一种跨开发语言的二进制文件分析技术。软件成分分析分为两种模式，静态和动态。静态模式是使用工具对目标二进制文件进行解压，识别和分析各个部分的关系；动态模式则是依赖于活动过程，在活动执行的同时收集必要的活动元数据信息，通过数据的方式对目标组件各个部分之间的

关系进行标定。动态模式主要针对组件缺乏自描述信息的场景，如 tgz 文件，而静态模式比较适合有自描述信息的文件，如 docker image。

越来越多的应用程序由开源代码组成。据估计，开源代码占到了应用程序代码的 90%。实际上应用程序是由不同的组件组成，企业需要保证所有的组件成分都是安全的，这样才能够有效地管理和降低风险。这也正是企业在确保代码库安全时所面临的挑战。鉴于开源的日益普及，以及最近违规和网络攻击的宣传，大众对 SCA 的关注热度逐步上升。开源在推动数字化转型中所扮演着不可忽视的角色，而这个趋势在近期几乎不会发生改变。企业使用开源来帮助他们在各自的市场中拥有竞争优势。他们必须控制开源的使用来管理和减少其伴随的风险。当 SCA 工具能够满足我们所提到的这些关键需求，将能成功帮助企业在市场中更好地竞争。

1.4 复用关系推理

二进制软件与开源软件之间的关联关系和差异性关系能极大提升漏洞挖掘人员的效率。存在源代码的软件漏洞挖掘可以使用基于插桩的模糊测试，相比于二进制的模糊测试效率更高，同时人工审计成本也会极大降低。在发现二进制软件与源代码软件关联关系的基础上，寻找二进制软件与开源软件之间的差异性对于漏洞挖掘人员也能提供很大的帮助。但是现有的源代码和二进制映射关系发现使用极为简单的特征和规则，存在着适用性弱，效率低下等问题。为此，对嵌入文本和程序语义的二进制溯源推理就显得十分必要。

很多二进制软件在开发过程中会使用大量的第三方库软件来降低开发成本。例如腾讯玄武实验室在 2017 年 CanSecWest 上的报告显示 [11]，Windows 平台的软件除核心模块外，其余的模块都是使用了大量的第三方库，而且大多数动态链接库仍使用存在漏洞的老版本。趋势科技在对 Adobe Reader 软件分析发现 [12]，Adobe Reader 中也复用了 sablotron 和 libtiff 等开源第三方库，并通过对 sablotron 进行漏洞挖掘从而发现了 Adobe Reader 的近十个 0day 漏洞。

二进制与源代码软件间的关系也可分为不同的种类。根据二进制是否存在导出函数我们可以将二进制软件分为动态链接库和 EXE/ELF 软件。动态链接库软件有一部分是可以直接通过动态链接库的名称确定它的源代码项目名称，如 libpng.dll 对应的源代码项目为 libpng。但是仍存在大部分的动态链接库不能直

接通过软件名来判断其对应的源代码项目。这又分为三种情况：1）一个动态链接库是由多个开源项目编译而来，如 7z.dll 是由 7-Zip、bzip2 和 7zip32 编译而来；2）一个开源软件编译生成了多个动态链接库，如 QT 编译生成 Qt5core.dll、Qt5Gui.dll 等；3）一个动态链接库对应一个开源库，如 Adobe reader 的 AXSLE.dll 是由 Sablotron 编译而来。不存在导出函数的二进制 PE 文件也可能由部分源代码项目静态编译而成，如 FoxitReader.exe 中使用了 libpng 和 zlib 等开源软件。

在确认二进制软件是由某个或某些源代码项目编译而来后，我们还要明确开发者或厂商在此过程中是否对源代码进行修改后编译（差异性识别），因为有可能修改后的代码会引入新的漏洞。例如，CVE-2018-6830 是 Foscam 摄像头的一个漏洞 [13]，该漏洞是开发者在复用开源软件 lighttpd 的过程中对 lighttpd 增加的代码中引入的漏洞。

因此我们需要明确二进制程序和源代码程序之间的对应关系和差异性，以此指导漏洞挖掘人员更有目的地发现目标软件的漏洞。为此，需要建立源代码与二进制关联关系发现的推理规则，并以此为基础发现源代码项目与二进制项目之间的映射关系。

针对源代码和二进制项目的映射关系建立这一问题，我们需要解决三个问题。首先是，如何检测代码相似性的问题。二进制代码是机器语言，而源代码是高级语言，两者在代码表达和所含代码信息上存在巨大差异。为实现准确的代码相似性检测，应选择哪些代码特征，如何提取代码特征，使用什么特征匹配算法，怎样计算相似度，从而获得代码相似性检测结果？其次是，如何识别代码复用关系的问题。代码复用关系并不通常是简单的一对一关系，大概率情况下是复杂的多对多关系。一个二进制文件可能复用了一个或多个开源组件，也可能只复用了开源组件中的部分代码。一个开源组件可能编译成一个库文件，也可能被编译为多个库文件。一个源代码可能被编译进库文件中，也可能只是作为示例代码，不被编译到任何库文件中。如何识别复杂复用关系，避免受复杂复用关系影响得到偏差较大的相似性检测结果，从而得到错误的复用关系识别结果？最后是，如何进行规模化软件分析的问题。与源代码或 Android 代码分析中万级项目的规模相比，二进制软件的分析规模一直较为受限，现有研究大多以函数数量计数，通常为百万级函数规模，远远小于源代码或 Android 代码分析规模。究其原因，主要在于二进制文件难以批量获取，并且二进制代码特征粒度较细且结

构复杂，比对效率低。如何解决二进制文件的大规模获取问题，以及如何选取合理的代码特征并设计其表达形式与存储结构，来达到更高的分析效率?

为解决上述三个问题，实现准确高效的代码相似性检测和复用关系识别，并开展规模化软件分析，本工作将对以下三个内容进行深入研究：首先是抗编译优化的代码特征提取技术。抗编译优化的代码特征应很少或根本不受编译优化过程影响，从而在源代码及其同源二进制代码中共同存在，需要对代码特征进行评估来选取合适的代码特征，并且还需要采用合理的技术手段准确地提取二进制代码和源代码特征。其次是面向多复用类型的代码相似性分析技术。针对选取的代码特征，需要设计合理的特征匹配算法和权重算法，以准确计算代码相似度。另外，代码复用关系是复杂的多对多关系，需要对复杂复用关系进行准确识别，以减少由复杂复用关系引入的代码相似性检测中的误报和漏报，以便得到正确的分析结果。最后是大规模的自动化代码复用检测框架。为实现规模化软件分析，需要对大规模代码特征进行针对性的存储优化，以系统性的提升特征匹配效率。另外，还需实现二进制文件的自动化大批量获取，以便开展大规模实验。

1.5 漏洞

漏洞是软件系统或产品在设计、实现、配置、运行等过程中，由操作实体有意或无意产生的缺陷、瑕疵或错误，它们以不同形式存在于信息系统的各个层次和环节之中，且随着信息系统的变化而改变 [14]。漏洞是网络安全攻防对抗的焦点，软件与系统中的漏洞在网络空间安全博弈中往往具有决定性的作用，在复杂多样的系统中限时发现定向目标的高价值漏洞是网络攻防博弈中制胜的有力武器。

漏洞作为网络攻击的杀手锏，在众多网络攻击事件中发挥着核心作用。2010 年，Equation Group[15] 利用一系列 0day 漏洞攻击伊朗工业控制系统，导致其浓缩铀工厂约 1/5 的离心机报废，最后造成核电站推迟发电。2016 年，恶意软件 Mirai[16] 利用漏洞感染了大量的物联网设备，并对动态 DNS 提供商 DynDNS 进行攻击，导致了美国一半的互联网瘫痪 [17]。Shadow broker[18] 泄露的 NSA 方程式（Equation Group）的数据中包含微软、思科等公司的主流操作系统和网络设备的漏洞利用代码，例如“永恒之蓝 (EternalBlue)[19]”，“永恒浪漫 (EternalRomance)[20]”等漏洞利用工具，这两个利用工具基本覆盖了当时所有版本的

Windows 操作系统，攻击者通过这些漏洞可以直接远程攻击默认配置的主机，影响十分巨大。在其泄露约一个月后，“永恒之蓝”被黑产组织用来制造蠕虫式传播的勒索软件 WannaCry[21]，在短短一天内感染了全球 100 多个国家的数十万台设备。Hacking team[22] 泄露的 RCS（Remote Control System）系统中，包含主流操作系统和移动终端的多个 0day 漏洞，这些 0day 漏洞是 Hacking Team 的核心资产，它们可以帮助 RCS 系统轻松渗透并监视目标主机或移动终端。

当前软件与系统安全漏洞仍然频繁产生。2017 年仅 NVD（National Vulnerability Database，美国国家漏洞库）[23] 一家漏洞库就收录了 14714 条 CVE（Common Vulnerabilities & Exposures，公共漏洞披露）[24] 漏洞信息，2018 年截止至今即已收录 9800 多条 CVE 信息 [25]。在此之外还有大量未被发现或已被修复但未被披露的漏洞 [26]。

伴随着互联网、移动互联网、物联网和工业控制系统的快速发展，软件与系统正在发生着显著的变化，总体呈现出结构复杂化、种类多样化、数量规模化、形态动态化的特点。具有代表性的 Windows 操作系统源代码已从 Vista 版的 5000 万行增长到 Windows 10 的 1 亿多行，托管平台 Github 中的项目累计已超过 3800 万，BlackDuck 公司已经存储的源代码超过 5300 多亿行，而软件开发中越来越多使用类/模板等语言特性、COM/.NET 等组件或第三方库，这使得软件结构日益复杂，这些变化对现有的漏洞挖掘方法提出了新的挑战。另一方面，在“黑客地下产业”经济利益驱动下，从漏洞发现、利用到网络渗透攻击，已经形成分工明细、规模化的产业链，各种稀有漏洞资源待价而沽，从漏洞首次发现并被用于渗透攻击的周期被缩短到几周甚至几天。因此，快速、深入的漏洞分析与发现成为网络防御的关键。

1.6 研究意义

近年来，计算机软件在人们生活的方方面面快速普及并逐渐深入，软件数量呈现爆发式增长，功能日益丰富。随着软件复杂度的提升，人工代码审计越来越困难，漏洞挖掘工作越来越依赖于自动化的软件分析工具。开源软件因其源代码的可获取性，已经拥有了相对成熟的分析手段，在自动化软件分析和漏洞挖掘方面走在了无法获取源代码的闭源软件前面。

闭源软件的常用漏洞挖掘方法有模糊测试和代码审计。模糊测试通过向目

标系统提供非预期的输入并监视异常结果来发现软件漏洞。但在面对复杂软件时，通用模糊测试器难以应用于复杂软件，专用模糊测试器的开发成本高，两者都不能针对复杂软件开展快速有效的漏洞挖掘工作。代码审计通过直接审查软件代码是否存在设计和实现错误来发现软件漏洞。面对复杂软件时，代码审计则会消耗大量的时间成本和人力成本，同样存在明显的局限性。可以说，目前为止还没有能够很好地应对复杂闭源软件的漏洞检测技术。

从软件工程的角度来看，为了能够降低开发成本，缩短开发时间，复杂软件通常会通过引用现有的成熟组件来实现部分功能。也就是说，复杂软件通常由不同来源的组件构成。腾讯玄武实验室的阿图因系统分析全网软件后发现，存在至少 4000 个共享库，其中包括内部库也包括第三方库 [27]。而 OpenSSL 的 29 个版本共被不同软件复用了 2583 次，其中高达 291 次复用涉及受 Heartbleed 漏洞影响的版本 [28]。另外，通过手工分析知名的 PDF 阅读器 Foxit Reader，我们发现其中至少复用了 13 个库，包括 libcurl、OpenSSL、MFCDLL、VS C Runtime、Zlib、Qpid Proton、Jrsys、libpng、libtiff、libjpeg、leptonica、v8 和 sqlite 等。可见组件复用现象普遍存在于软件中。

如果组件含有漏洞，那么引用该组件的软件也有可能受漏洞影响。我们将因复用含有漏洞的组件而引入的软件漏洞称为组件复用型漏洞。组件复用型漏洞普遍存在。在 OWASP Top 10 Application Security Risk[29] 中，Using Components with Known Vulnerabilities 排名第 9，而且这一类安全风险的主要特征即为传播广泛。另外，组件复用型漏洞严重影响软件安全性。例如，2015 年曝光的 Java 反序列化漏洞存在于 Apache Commons Collections 这一基础类库中，利用该漏洞可实现远程命令执行，因这个库被广泛复用，这一漏洞几乎横扫各大 Java Web Server，影响巨大。

因为复杂软件通常由组件构成，组件复用型漏洞又普遍存在，并且有巨大的影响，我们可以考虑通过针对性的检测组件复用型漏洞，来应对复杂闭源软件的漏洞挖掘问题。

闭源软件的组件复用型漏洞检测与开源软件相比，相关研究尚不充分，仍有很大的研究空间。目前已有的组件复用关系分析系统——腾讯玄武实验室的阿图因系统，拥有全网软件分析能力，结合简单的软件分析后，得到了全网软件的组件复用情况概览。本工作将致力于拥有类似的全网软件分析能力，在此基础

上结合版本号粒度的复用关系检测，以及一个收录了已曝光漏洞的漏洞信息库，基于软件与组件间的复用关系，批量发现组件复用型漏洞。

不过，快速准确的大规模复用关系检测并不是一个简单的问题。目前需要分析的软件数量较大，全网至少有数万个软件，可以拆分出数十万个二进制文件，另外还有数千个组件，对检测方法的效率和通用性要求很高。版本号粒度的复用关系检测也尚无直接的分析方法，需要自行设计和实现。

综上所述，目前没有可应对复杂软件的闭源软件漏洞检测技术，为了能够快速的批量检测闭源软件中的漏洞，我们将采取针对性的检测组件复用型漏洞的思路，通过构建全网软件分析能力，研究版本号粒度的复用关系检测技术，积累漏洞信息库的方法，实现一个可靠的漏洞检测方案并部署运行。

二　国内外发展现状与趋势

2.1　二进制复用关系检测技术

复用关系检测的相关研究已经经过了若干年的积累，和代码克隆检测、代码相似性检测、代码搜索等基本属于同一研究方向，核心都是检测不同来源的代码是否相同或相似。目前这一领域已经有二进制代码间的复用检测，和源代码间的复用检测的相关研究，尚未有二进制与源代码间的复用检测的研究。

源代码间的复用检测目前已经有相对不错的研究成果，能够兼顾效率和准确率，实现规模化的自动分析。与源代码间的复用检测相比，二进制代码间的复用关系检测面临了更多挑战。首先，难以判断二进制代码间的差异来自于开发过程还是编译过程。不同的编译器和优化选项会导致二进制代码间存在巨大的差异，如何识别来自于开发过程的差异，忽略来自编译过程的差异，是二进制复用检测中的一大难点。其次，很多代码信息在编译过程中消失了，如函数名、变量名、宏等等，二进制复用检测所能使用的信息相比源代码要少很多。另外，反汇编是二进制分析的必要过程，这一过程耗时较长，而且并非完全正确，可能会引入新的错误。

目前复用关系检测主要有三种研究方法，分别基于句法分析、语义分析和机器学习。

1. 基于句法分析的复用关系检测技术

主要思路是比对指令或源代码文本的相似性，通常会对汇编指令或源代码进行正规化处理。基本假设是相同的代码会编译出相同或相似的机器码。

在源代码间的复用关系检测中，通常有以下几种方法：基于字符串的复用关系检测，通过对源代码中的硬编码字符串进行匹配来识别复用关系；基于 token 的复用关系检测，对源代码的 token（句法解析所获得的标识）进行匹配，如 [30]；基于抽象语法树（AST）的复用关系检测，对 AST 进行树同质性分析或子树匹配。

在二进制代码间的复用关系检测中，通常采用基于汇编指令的复用关系检测，对汇编指令进行正规化，然后利用文本分析技术进行匹配，如 [31]。

2. 基于语义分析的复用关系检测技术

主要思路是解析并提取二进制或源代码中的语义信息进行比对。基本假设

是相同的代码会编译出具有相同语义的代码。

源代码的复用关系检测通常无需使用语义分析来进行复用关系检测。

在二进制代码间的复用关系检测中，通常有以下几种方法：基于控制流的复用关系检测，对控制流图（CFG）做图同质性分析，或对 CFG 的质心（centroid）[32] 进行计算与比对，如 [32][33][34]；基于数据流的复用关系检测，利用符号执行技术分析函数的输入输出和数据传递路径，并进行匹配，如 [35]；基于中间表达（IR）的复用关系检测，将汇编指令逆向到 IR，结合定理证明技术对关键执行路径进行匹配，如 [36]。

3. 基于机器学习的复用关系检测技术

主要思路是将软件特征数值化，对软件特征向量进行学习训练。基本假设是少数软件的特征可代表全体软件。

源代码间的和二进制代码间的复用关系检测都可以基于机器学习进行分析。常用的方法有：基于 codebook[37] 的复用关系检测，通过聚类的方法分析目标软件集中的全部 CFG，获取其中最具代表性的几个作为基准 CFG，组成 codebook。其他 CFG 将分别与几个基准 CFG 进行相似性比对，由几个相似度数值组成 CFG 的特征向量；基于软件特征数值化的复用关系检测，提取软件中易于用数值表达的软件特征，如 centroid、某类指令或数据的数量等，生成软件或函数的特征向量，如 [38][39]。

三种复用关系检测研究方法的对比详见下表。经对比总结，各研究方法的优缺点见表 2-2。总的来说，现有方法普遍存在三个问题，分别是：

- 复用关系检测的组件识别粒度与漏洞分析的需求不完全相符
- 效率与准确率难以兼顾
- 规模化的分析较为困难

表 2 三种复用关系检测研究方法的对比

研究方法		比较对象	分析粒度	受编译影响	可拓展性	分析规模
句法分析	String Token AST	src vs. src	组件级	–	软件集可持续更新	万级开源项目
	汇编指令	bin vs. bin	多为函数级	影响严重	软件集可持续更新	百万级函数
语义分析	控制流 数据流 中间指令	bin vs. bin	多为函数级	几乎不受影响	软件集可持续更新	针对目标软件 大规模
机器学习	codebook 软件特征	src vs. src bin vs. bin	组件级或函数级	未知	更新软件集需重新训练调参	大规模
所需的方法		bin vs. bin bin vs. src	组件版本号级	几乎不受影响	软件集可持续更新	软件: 万级 函数: 亿级 组件: 千级

表 2-1 三种复用关系检测研究方法的对比

表 2-2 三种复用关系检测研究方法的优缺点

2.2 SCA 工具业界现状

2.2.1 学术界 SCA 工具发展情况

Armijn Hemel 等人 [40] 实现了首个二进制代码与源代码间的复用关系识别系统 Binary Analysis Tool(BAT)。该工具基于常量字符串和二进制数据进行匹

配，可用于识别固件中复用的开源库代码。不过，对于 Windows 上的可执行程序（Portable Executable,PE）而言，开源组件的字符串也不一定会被保留，因此 BAT 的方法不能广泛适用于普通桌面程序。实际上，即使在一些开源组件官方发布的二进制文件中，也几乎没有有效的字符串信息。对于二进制代码分析来说，单独使用字符串特征是不够可靠的。Ruian Duan 等人 [41] 实现了一个 Android C++/Java 代码的复用关系识别工具 OSSPolice，基于字符串和导出函数进行特征匹配，引入了层次索引（Hierarchic Indexing）方案，实现了大规模分析。但 Android C++ 代码一定会暴露出导出函数供程序调用，而 PE 文件可能将开源组件的导出函数封装为内部函数进行内部调用，不会再作为导出函数记录在导出表（Export Address Table,EAT）中。

比较有代表性的学术界工具见下表。

表 2-3 四个比较有代表性的学术界 SCA 工具

2.2.2 工业界 SCA 软件发展情况

工业界目前比较有名的两款软件分别是 Cybellum 和 BinaryAI。

Cybellum[42] 安全套件是用于汽车组件的风险评估方案套件，它专门针对汽车市场提供完整组件可见性和风险评估。Cybellum 安全套件既可满足汽车管理人员对已部署和即将推出的车辆型号进行持续风险分析的要求，也可满足组件开发团队的需要。它使得安全汽车软件的开发变得自动化，轻松地与标准开发以及 CI/CD 环境集成。V-Ray 是用于汽车组件的基于云或内部部署的风险评估方案。它基于自动漏洞检测，提供完整的组件可见性和风险评估。对固件自动地逆向并扫描其中的安全漏洞和威胁，模仿攻击者的操作手法，并提供对所扫描组件的完整可见性。V-Ray 在集成阶段运行，不需要任何源代码，从而使厂商能够评估风险，进行补救并安全地发布车辆。

BinaryAI[43] 基于软件安全和人工智能领域的多年研究经验，科恩实验室积极布局软件成分分析方向，已落地并助力厂商修复软件安全问题，现在首次将 SCA 能力以平台型产品 BinaryAI 免费开放给用户，旨在推动软件成分分析在 DevSecOps、安全研究等场景应用发展。BinaryAI 是二进制文件 SCA 的智能分析平台，自动化完成文件解析到输出分析结果全部使用环节，帮助研究人员高效实现 SCA 线上检查的需求。

下表对比了上述两款软件的基础信息。

表 2-4 两个比较有代表性的工业界 SCA 工具

2.3 软件复用关系推理技术

2.3.1 开源软件关联关系建立

开源软件之间存在多种关联关系，学术界将目光主要聚焦在代码克隆领域[44][45][46]。源代码软件的克隆检测主要分为基于词法的克隆检测 [47][48]、基于抽象语法树的克隆检测 [49][50]、基于控制流图的克隆检测 [39][34]，这些检测方法已经取得很好的研究成果。

开源软件之间存在着 OSS（Open Source Software）[51] 复用的情况，以 Chrome 源码为例，Chrome 源代码中使用了 expat、ffmpeg、sqlite、freetype、libxlst、zlib、unrar、libjpeg 等第三方库。检测开源软件之间的 OSS 复用对于漏洞传播推理、二进制软件的源代码项目识别等具有重要的意义。Lee[41] 等创新性提出了基于层次索引的方法尝试检测开源软件之间 OSS 复用情况。具体来说，Lee 等对开源软件构建了函数名称和字符串的倒排索引，其上一层为函数或字符串所在的文件名，然后是上一层目录，以此类推最后结点为源代码项目根目录。在构建完索引树后，从叶子节点出发，如果软件 A 的索引树是软件 B 索引树的子树，则软件 B 复用了软件 A。该方法能够完美识别 A 的索引树是 B 的索引树子树情况的软件项目复用检测，但是经过我们对近 20 余个存在复用的开源软件项目分析发现，并不存在某软件的索引树是另外一个软件索引树的情况。

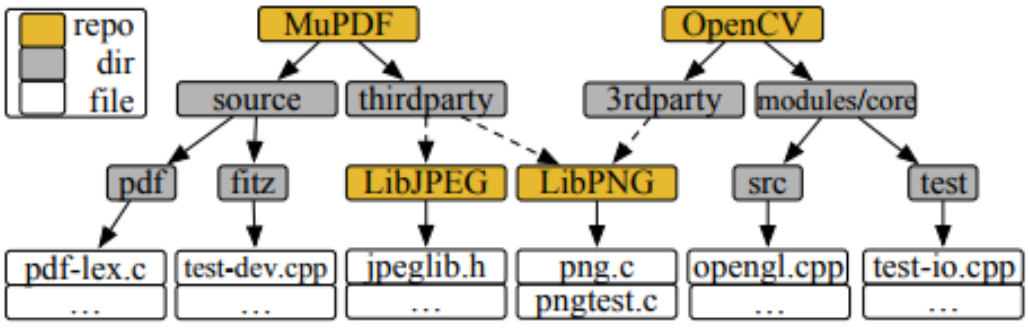

图 5 开源软件 OSS 复用示例

2.3.2 开源软件与二进制关联关系建立

商业软件在开发过程中也会大量使用开源第三方库，BlackDuck 公司对超过 1000 个商业软件代码审计发现，超过 96% 的商业软件含有开源组件，且每个软件平均含有 257 个开源组件。腾讯玄武实验室在 17 年 CanSecWest 会议上也提到了 Windows 平台上的软件除了核心功能是自己开发外，其他的很多模块也都是使用了互联网上开源的第三方库。对于给定的二进制软件，如果能够辨别其是否由公开的源代码项目编译而来且能够获得项目的名称对于挖掘二进制软件漏洞具有重要意义 [27]。

Lee 等人通过导出函数和字符串信息来判断源代码与二进制程序之间的关联关系，但该方法只适用于动态链接库且动态链接库中未丢弃字符串信息，适用性较差。Dhaval[52] 等人提出的 BinPro 使用字符串，整数常量，函数调用等信息作为二进制和源代码比对的特征，同时引入机器学习的方法对源代码编译过程中的编译优化导致的函数内联进行预测，从而发现源代码函数与二进制函数间的一一对应关系的方法。作者创造性提出了基于机器学习方法的函数内联预测。该方法是二进制和源代码映射关系发现的一大步尝试，但该方法还存在诸多不足之处，例如，该方法是已知二进制和源代码之间存在关系的情况下进行的函数映射发现，从而确认二进制是否由源代码进行编译，故该方法不能快速有效解决给定二进制情况下的二进制溯源问题。

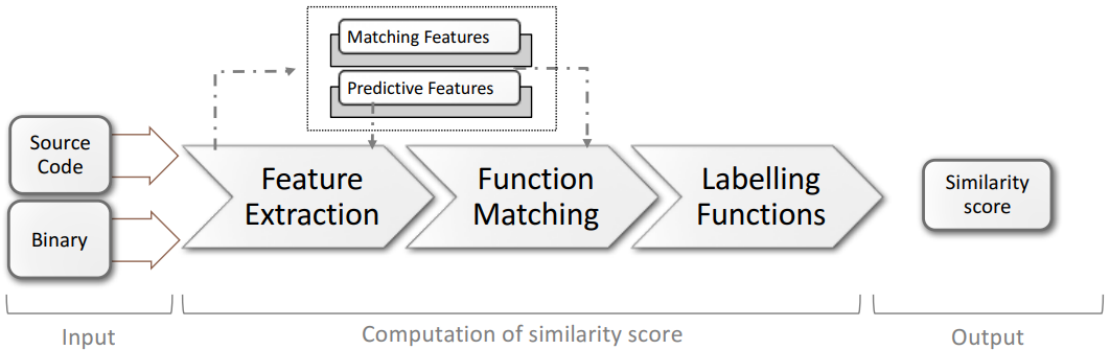

图 6 BinPro 流程图

2.4 基于已知漏洞的漏洞发现

信息安全界通过借鉴软件分析领域的代码克隆检测技术对基于已知漏洞的漏洞发现提出了一些行之有效的方法。较具代表性的有：

- 德国哥廷根大学 Yamaguchi[53][54] 等人提出了一种将已知漏洞的函数嵌入到一个向量中，在同样被向量化的目标系统代码中通过计算向量间的距离来发现与漏洞函数类似的函数，从开源媒体处理库 FFmpeg 等系统中发现了数个 0day 漏洞；

- 美国雪城大学的 Feng[55] 等人针对物联网设备中的固件，设计实现了一个基于图匹配的相似缺陷检测系统 Genius。在 Genius 中，代码中的函数被建模为一个属性控制流图 ACFG(Attributed Control Flow Graph)，并通过聚类大量的固件函数的 ACFG 形成一个包含 16 个码字的码书。含有漏洞的函数和待检测的函数与各个码字所对应的质心函数在 ACFG 层面计算匹配度并编码为一个十六维的向量，用其进行函数件的相似度计算来查找潜在的漏洞函数；

- 韩国高丽大学的 Kim 等人提出了一个高效的漏洞函数克隆检测系统 VUDDY。与同类系统类似，VUDDY 特别强调系统的可扩展性，即检测克隆漏洞的效率[30]。为此，VUDDY 采用了较为极端的方式来处理代码。程序函数被以一种激进的方式进行了抽象，函数中的局部变量名、形参名、数据类型和调用函数名被转换为四个固定的字符串然后使用 MD5 算法对每个抽象后的函数进行摘要生成，并在此过程中记录抽象后函数的长度。同时基于相同的处理方法对漏洞函数进行处理，最后漏洞函数克隆检测实质上转化为各函数间摘要比对的问题。

- 华中科技大学的邹德清教授团队针对漏洞检测中依赖人类专家定义漏洞特征和漏报较高的问题，首次将深度学习技术引入到漏洞检测领域，提出了基于深度学习的漏洞检测系统 VulDeePecker[56]。以代码片段（code gadget）为粒度，基于双向长短期记忆网络（Bi-LSTM,bidirectional Long Short-Term Memory）模型自动学习生成漏洞模式，在不需要人类专家定义特征的前提下，自动检测目标程序是否含有漏洞，并给出漏洞代码的位置。

以上工作在实际应用中都取得了明显的效果，从实际系统中也成功发现了相当数量的 0day 漏洞，较好地证明了相关技术路线的合理性。但是，通过漏洞相关知识的提取来进行漏洞检测仍面临着一些亟待解决的突出问题，如噪声代码对漏洞相关知识提取的干扰、程序的规范化表示等。

三　课题主要研究内容、预期目标

3.1　总体内容

图 7 研究内容概况

本文围绕面向二进制软件的成分分析及复用漏洞检测技术展开研究，首先对二进制软件 SCA 工具进行多维度测评，基于测评结果优化自研二进制 SCA 工具，进行精准高效的二进制多粒度成分分析，最终服务于研究室内的科研项目，为大规模特定领域组件复用型漏洞检测与评估提供支撑。

3.2　二进制软件 SCA 工具多维度测评技术

二进制软件 SCA 工具多维度测评技术用于对当前有代表性的 SCA 学术界工具和工业界工具进行横向测评。目前没有对二进制复用检测工具的大规模测评类参考文献；通过此工作，可以了解 B2Sfinder 在同类工具中的优势和劣势，找到功性能提升点，进而进行自研工具的功性能提升。构建数据集的过程中需要对大规模软件进行自动爬取和自动编译，以提取精准的复用关系，其中大规模软件自动编译是一个难点；对二进制 SCA 工具进行测评需要预先设计测评体系，其中测评指标是非常重要的部分，对多维度测评指标的设计也是其中的一个难点。

针对这两个技术难题，本节将围绕 SCA 工具测评工作展开，重点研究大规模软件自动编译技术和二进制 SCA 多维度测评指标设计。

3.2.1 大规模软件自动编译技术

对于二进制软件 SCA 工具测评而言，无论是从 Ground Truth 的构建角度，还是从自研工具基础特征库的丰富角度，大规模软件的自动编译都是一个不可绕开的问题。要想构建一个内容丰富合理的 Ground Truth 或者是提取到更多的特征，都需要对软件的自动编译技术，人工手动编译可能编译的成功率会更好，但是其时效性无法满足客观实际的需求。

以构建 Ground Truth 为例，首先要构建爬虫系统对特定源码托管平台进行项目数据的爬取，这一步需要处理爬虫可能遇到的一些问题，比如更换代理、注册临时账号等；接下来就是对项目在版本级别上进行自动编译，这一步可能涉及到的问题是编译环境的搭建与选择、编译选项的处理等问题；然后就是对编译信息进行充分分析，识别出参加编译的源文件进而对其进行特征提取工作。

3.2.2 二进制 SCA 多维度测评指标设计

测评是对事物进行客观的测量与科学的评价。漏洞发现工具测评首先需要根据测评目的来构建或挑选一套目标程序集合作为测试集，然后制定被测工具的效能评判指标。

其中，效能评判指标是用来量化评估被测工具漏洞发现效能的数据计算与统计方式，根据测评目的、被测工具类型等来制定，例如静态分析工具的效能评判指标通常为漏洞检测的漏报率、误报率等，模糊测试工具的效能评判指标通常是漏洞触发数量、代码覆盖率等。为了能够按照评判指标来计算被测工具的效能，测试集通常会包含标注信息来辅助效能指标的计算，例如目标程序漏洞类型、漏洞位置的标注信息能够被用于评判工具的漏报从而计算漏报率。执行测评时，被测工具以测试集中的目标程序为输入，输出的漏洞检测结果依照测试集提供的标注信息进行评判，然后以事先制定的评判指标为指导进行计算与统计，得到测评结果数据。测评结果数据可以被进一步分析，依照测评目的进行应用，例如对比不同工具的效能优劣、验证方法技术的有效性等。

上述测评过程是否能得到相应的结果，离不开测评指标的设计，合理的测评指标会更加容易发现科学问题。

3.3 精准高效二进制多粒度成分分析技术

针对自研工具在功性能方面存在诸多可提升空间，如版本识别率较低、对ELF 文件支持不友好以及基础数据库规模较小等问题，通过多粒度编译不变量特征提取技术进行多特征提取，优化基础数据集；通过渐进式特征敏感的成分分析检测技术对二进制文件进行库级别、版本级别、函数级别进行复用关系确定，进而达成软件成分分析的目的。

3.3.1 多粒度编译不变量特征提取技术

软件指纹，也称作软件摘要，指的是从软件中提取出的、可以用来进行复用关系检测的有效特征。这一部分将从闭源软件中提取出版本敏感的软件摘要，满足尽可能的不受编译过程影响，在同一软件的不同版本间存在差异，提取和匹配的时间复杂度较低（如达到线性复杂度），并且易于存储（如可以数值形式存储）等要求。多粒度编译不变量特征提取技术包含以下两个子技术：

1. 版本敏感的数据特征生成技术

将以软件中可能存在的常量为特征，体现软件的算法、特有数据和标识、所涉功能等基本特征。

2. 版本敏感的控制流特征生成技术

将以软件中存在的特殊控制流或复杂控制流路径为特征，体现软件特有的执行逻辑和功能特征。

3.3.2 渐进式特征敏感的成分分析检测技术

渐进式特征敏感的成分分析检测将同时实现闭源软件中开源组件和闭源组件的复用关系检测。通过匹配闭源软件摘要和候选组件摘要中的各类特征，生成复用关系图，结合漏洞信息库生成可疑漏洞列表。渐进式特征敏感的成分分析检测技术包含以下两个子技术：

1. 基于完全匹配的数据特征匹配技术

数据特征相对稳定，大多不会随版本更迭发生重大变化，也不会因编译过程的不同而有巨大差异。因此将基于完全匹配设计数据特征的匹配规则。

2. 基于相似度计算的控制流特征匹配技术

控制流特征相对易变，即使特征的真实含义没有变化，其具体的表达形式仍

有可能存在差异。因此将基于相似度计算，设置阈值来设计控制流特征的匹配规则。

3.4 大规模特定领域组件复用型漏洞检测与评估

软件知识图谱旨在为限时定向漏洞挖掘人员提供帮助，提高漏洞挖掘的效能，将易于发现的漏洞（浅层漏洞）交由软件知识图谱来寻找，而将具有复杂逻辑较难发现的漏洞（深层漏洞）交由漏洞挖掘人员来发现。经分析发现，软件知识图谱适用于解决由于各种复用问题导致的漏洞，但是现有的基于复用的漏洞挖掘方法仍存在效能或准确率低的问题。

因此，我们将围绕复用知识支持的漏洞发现与确认展开研究，研究基于代码特征的软件 1day 漏洞发现、基于漏洞特征的软件 0day 漏洞发现和差异代码执行信息指导的漏洞确认技术。

3.4.1 大规模开源软件指纹生成技术

用于成分分析及复用漏洞检测的开源软件指纹生成需满足的必要条件之一是在静态分析的过程中消除代码间漏洞无关符号间的差异，如统一重命名不同的参数名/变量名/寄存器名等，否则将很难使生成的指纹对应于漏洞模式，用于不同开源项目代码间、不同可执行文件间的检测。同时，缺少对语句间及变量间控制与数据依赖的描述将不能很好地体现漏洞成因，造成所生成的漏洞模式受限，导致检测中漏报或误报的现象发生。因此，用于规模化漏洞检测的开源软件指纹生成技术需从特征提取、漏洞指纹生成到复用型漏洞定位的各个环节考虑消除无关符号差异，设计并体现关键语句和变量的控制与数据依赖关系。

1. 支持指纹生成的代码特征提取

在消除代码间漏洞无关符号的基础上，提取与所标记漏洞代码相关联的语句和变量间的控制依赖和数据依赖关系作为特征。具体来说，以对无关符号统一重命名的方式（如将所有函数传入参数重命名为 PARAM）消除代码间的差异，并在此基础上生成抽象语法树和程序依赖图，提取选择后的控制边和数据边依赖关系作为生成漏洞模式的基础特征。

2. 基于特征提取的漏洞指纹生成

以补丁修复前所标记的漏洞相关代码行作为输入，生成基于漏洞关联语句

及变量间控制和数据依赖关系的漏洞模式，并应用所生成的模式进行漏洞检测。具体来说，从漏洞相关代码行出发，获取其消除无关符号的代码表示形式，根据代码行语句类型设计选择切片方法（如针对赋值语句进行数据流前向切片；针对条件语句中的变量进行数据流后向切片，并在特定条件下开展基于控制流的前向切片），综合切片结果中的控制依赖和数据依赖关系以刻画漏洞成因，并形成漏洞模式开展跨“代码-代码”间的漏洞标记。

3. 用于复用性漏洞规模化定位的可检索指纹标记

以全局定位和局部定位相结合的方式，找到给定二进制可执行文件中所包含的开源库漏洞代码。具体来说，通过抗编译优化全局特征获取源代码与二进制代码的总体映射关系（如通过匹配字符串和函数调用关系进行映射），结合漏洞代码和全局特征位于源代码文件中的位置，以定位二进制可执行文件中漏洞函数的位置，其后进一步使用函数内的漏洞指纹标记确认漏洞代码在可执行文件中的具体位置，验证软件指纹生成技术的可行性。

3.4.2 固件快速解压缩及固件包直接分析技术

对于固件快速解压缩及固件包直接分析，拟开展整体系统框架的优化设计与研究实践。

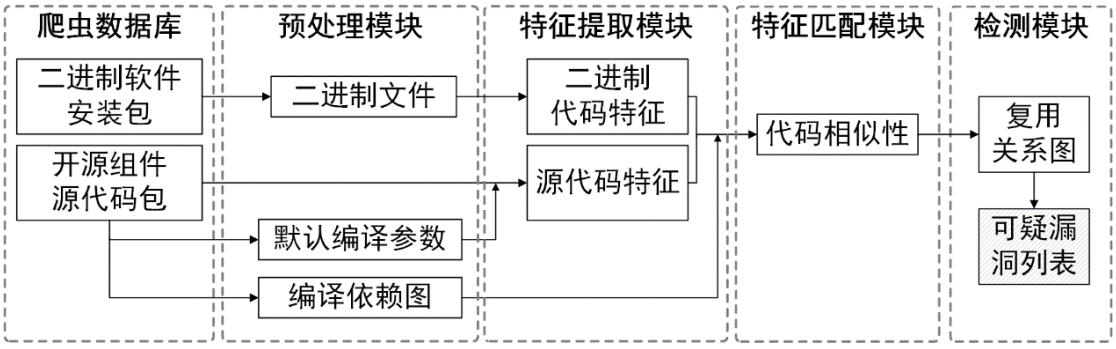

图 8 用于固件快速解压缩及固件包直接分析的规模化检测框架示意图

- 预处理模块：实现固件安装包自动拆解，提取默认编译参数提取，并构建编译依赖图；
- 特征提取模块：实现二进制代码特征和源代码特征的规模化提取和相关特征存储；
- 指纹匹配模块：实现代码指纹匹配，计算相应匹配特征权重，并开展基于指纹的相似度计算；

- 检测模块：识别固件包中复用关系，构建复用关系图，输出可疑漏洞列表。

3.5 预期目标

以大规模特定领域组件复用型漏洞检测与评估为牵引，首先对二进制软件 SCA 工具进行多维度测评，根据评测结果信息，优化自研工具以期进行精准高效的二进制多粒度成分分析，在此基础上，进行大规模固件基础特征库的扩充，完成大规模特定领域组件复用型漏洞检测与评估的目的。

四　拟采用的研究方法、技术路线、实验方案及其可行性分析

4.1　二进制软件 SCA 工具多维度测评技术

图 9 二进制软件 SCA 工具多维度测评技术

4.1.1　大规模软件自动编译技术

开源组件通常利用 makefile 来指定编译规则，然后利用 make 指令对 makefile 进行解析并对项目进行自动编译。Makefile 中包含了项目编译所需的全部信息，包括宏文件和头文件位置等编译参数。只要能够自动化的解析 makefile，提取开源组件所需的编译参数信息，就可以解决源代码静态编译中宏定义和头文件缺失的问题。

Makefile 有三种常见的生成方式，分别是利用 cmake 配置 makefile，用 autogen 配置 makefile，以及由开发人员直接编写 makefile。本文对这三种编译模式进行自动化识别，并自动执行相应的 makefile 生成命令，以生成正确的 makefile。生

成 makefile 后，可利用 make –n 命令获取从 makefile 中解析出的编译指令，如 gcc 命令或 libtool 命令，并从中获取所需的编译参数，包括-D、-L、-I、-std 等。下表列举了各编译参数的功能。提取这些编译参数，可以获取静态分析所需的必要信息，从而显著提升源代码静态分析的成功率。

表 4-1 编译参数的含义

4.1.2 二进制 SCA 多维度测评指标设计

对于学术界工具的横向测评而言，大多指标都集中在正确率、漏报率和误报率，本文的评价指标也不例外，但是会分两个层面来体现，分别是正确率【库级别、版本级别】、漏报率【库级别、版本级别】、误报率【库级别、版本级别】，如下表所示。

表 4-2 基础评价指标

除了基础评价指标外，本文根据实际情况新增了四个评价指标，简述如下：

1. 解压缩成功率

二进制 SCA 的第一步就是解压缩，解压缩是否成功决定了后面的复用分析是否能够往下继续进行。

2. 单文件执行时长

对于最终用户来讲，整个复用检测过程的时间长度也是一个重要的指标，本文取平均单文件执行时长作为评价指标。

3. 支持的文件类型

检测系统支持的文件类型也是一个重要指标。支持越多的文件类型，说明检测工具的可处理范围就越广，检测能力就越强。

4. 支持的处理器架构类型

随着国产化 CPU 等产品的不断发展，目前越来越多的送测产品对国产化产品会进行定向适配，于是就出现了更多架构的送测产品。支持更多架构的产品评测意味着工具的检测能力更强。

4.2 精准高效二进制多粒度成分分析技术

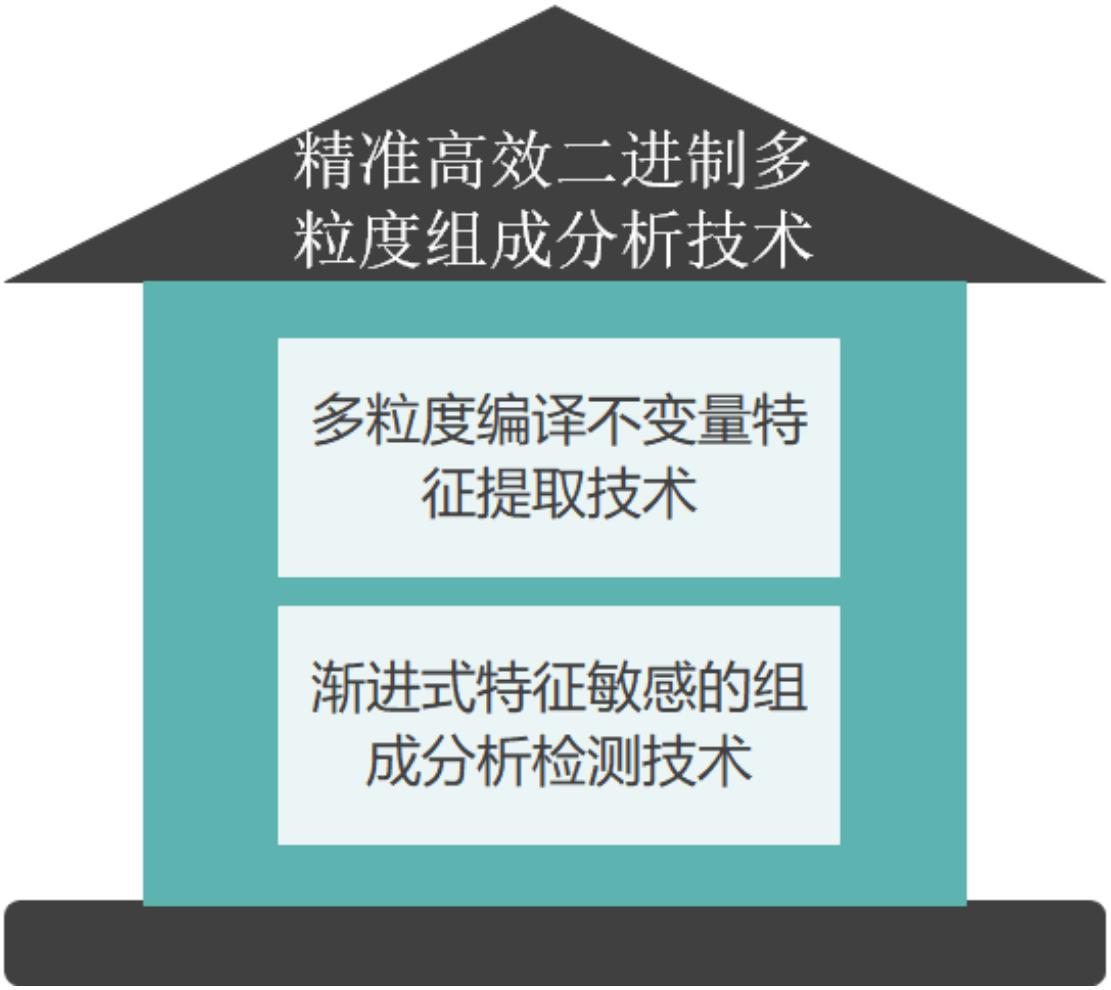

图 10 精准高效二进制多粒度成分分析技术

4.2.1 多粒度编译不变量特征提取技术

多粒度编译不变量特征提取技术包含以下三项工作：

1. 提取版本敏感的数据特征和控制流特征；
2. 提取二进制文件的文件信息；
3. 整合有效数据生成版本敏感的软件摘要。

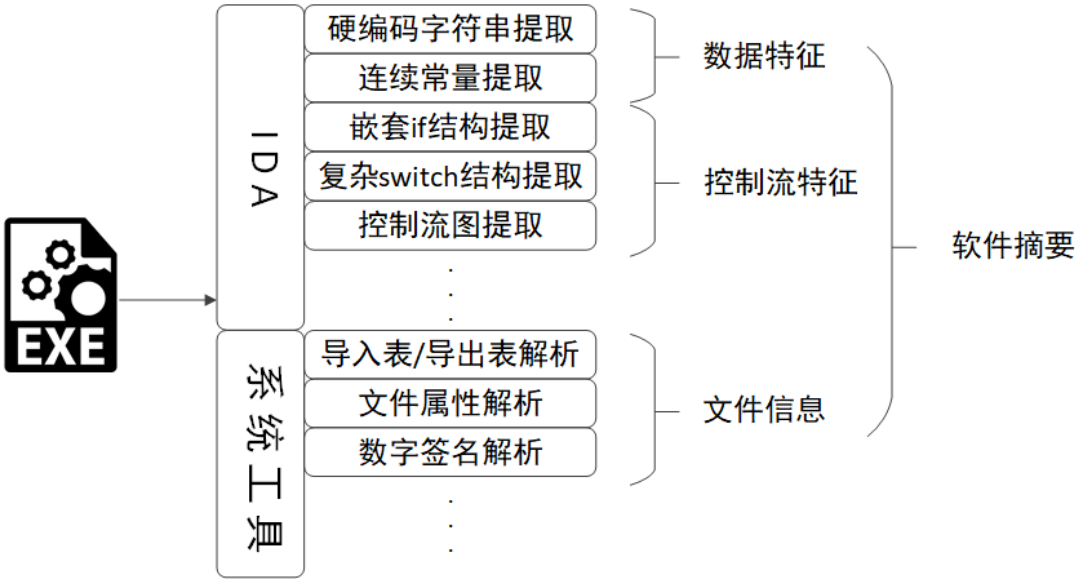

图 11 多粒度编译不变量特征提取方法

版本敏感的数据特征和控制流特征的提取采用版本敏感的数据特征生成技术，主要依赖定制的 idapython 脚本。目标二进制文件经 IDA 反汇编后，利用 idapython 脚本分别提取二进制文件中的硬编码字符串、连续常数等作为数据特征，提取嵌套 if 结构、复杂 switch 结构、控制流图等作为控制流特征。

二进制文件的文件信息的提取采用版本敏感的控制流特征生成技术，主要依赖系统工具或其他现有工具。系统工具或其他现有工具通过对文件头和资源节的解析，可以提取出二进制文件的导入导出表、文件详细属性和数字签名等文件信息。

在分析得到版本敏感的数据特征和控制流特征以及二进制文件的文件信息的基础上，对其进行整合，即可生成版本敏感的软件摘要。

4.2.2 渐进式特征敏感的成分分析检测技术

渐进式特征敏感的成分分析检测技术包含三项工作：

1. 获取闭源软件摘要和候选组件摘要；
2. 匹配摘要间特征并生成复用关系图；
3. 生成可疑漏洞列表。

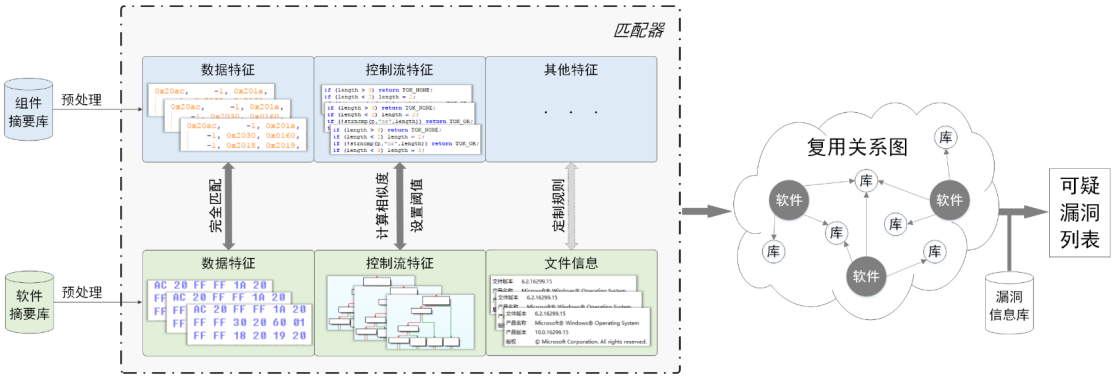

图 12 基于复用关系的漏洞检测流程图

闭源软件摘要采用版本敏感的软件摘要生成技术生成，候选组件的摘要通过改造现有工具并基于已有研究工作生成。在本课题中，候选组件可以为开源组件，也可以为闭源组件。目前开源组件的源代码分析工具相对成熟，可以通过改造现有工具实现开源组件摘要的生成。闭源组件与闭源软件本质上并没有区别，因此可以基于已有的研究工作生成闭源组件摘要。

摘要间的特征匹配将分为数据特征匹配、控制流特征匹配和其他特征匹配三部分进行。数据特征匹配将采用基于完全匹配的数据特征匹配技术，控制流特征将采用基于相似度计算的控制流特征匹配技术，其他特征将定制匹配规则来实现匹配。摘要中所有有效特征的匹配结果经综合分析生成复用关系检测结果。具有复用关系的软件和组件将生成复用关系图，节点为软件和组件，边表示复用关系，由引用者指向被引用者。复用关系图具有较强的可拓展性，随着研究的深入，复用关系可能会超出目前简单的软件与组件间的引用关系，而包含更丰富的信息和更高的阶数，复用关系图可经拓展用于更复杂情况下的攻击路径推理和攻击方式探索。

可疑漏洞列表是由复用关系图结合漏洞信息库生成的。漏洞信息库中会将漏洞与受影响的版本号粒度的组件关联起来，而复用关系图连接了闭源软件和复用的版本号粒度的组件。结合这两种信息，就可以将闭源软件与具体的漏洞关联起来，生成可疑漏洞列表。

4.3 大规模特定领域组件复用型漏洞检测与评估

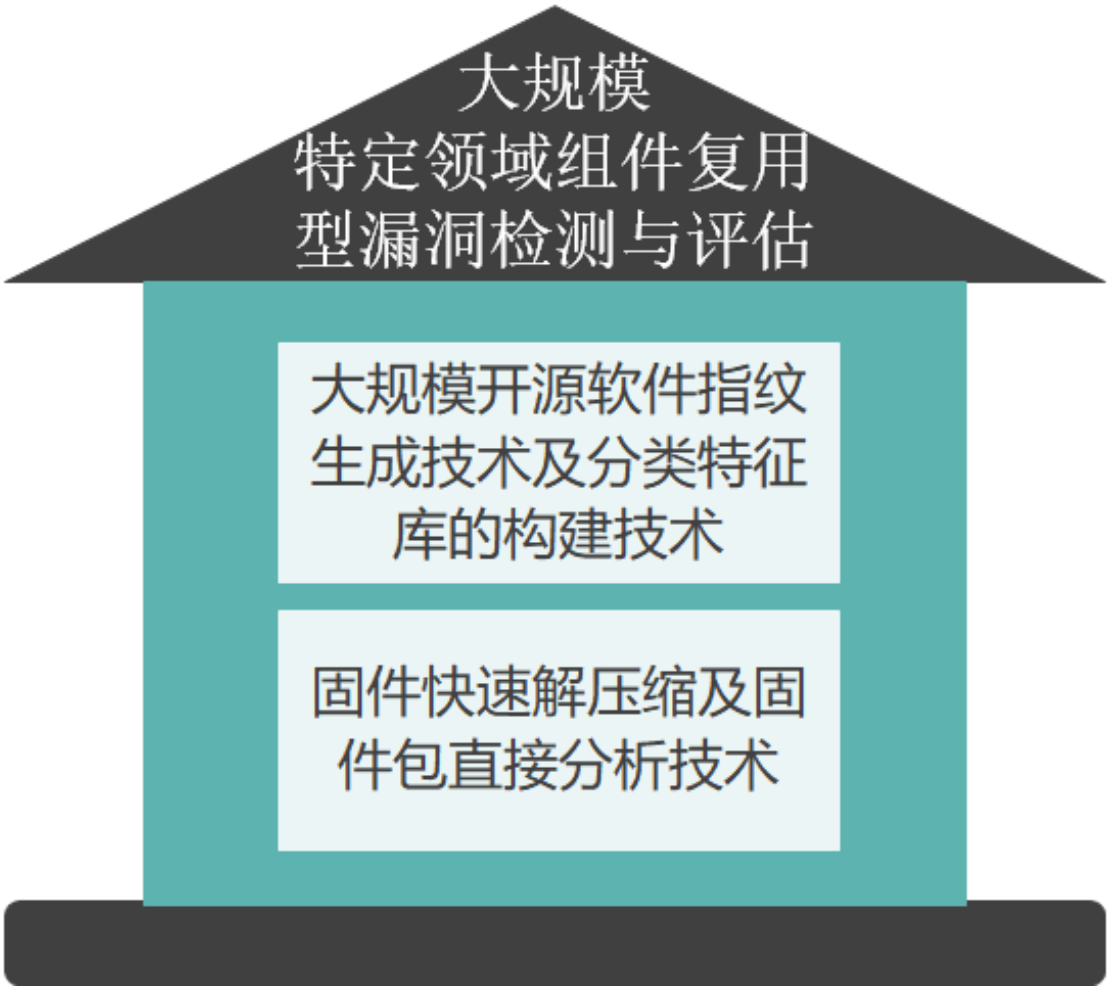

图 13 大规模特定领域组件复用型漏洞检测与评估

4.3.1 大规模开源软件指纹生成技术

1. 支持指纹生成的代码特征提取

正规化函数特征指在消除不同函数代码间漏洞无关符号后所提取的特征。将正规化函数特征的提取过程分为抽象化、格式化、生成代码行特征等三个步骤。以源代码为例，在对函数进行抽象化时，可采用将不同的传入参数、局部变量和字符串分别统一重命名为 PARAM、VARIABLE、STRING 的方式，避免无关符号对生成漏洞/修补模式的影响；在对函数进行格式化时，通过移除函数体内的注释、括号、tab 和空格符等，使得代码易于被处理；在抽象化和格式化处理之后，可从语法分析层面针对函数体提取程序依赖图，获取函数语句间的数据或控制依赖关系，生成语句间依赖关系集合作为正规化的函数特征。

2. 基于特征提取的漏洞指纹生成

通过对函数体进行抽象化、格式化、生成代码行特征等处理，可以生成函数体语句间依赖关系集合。在此基础上，可通过对漏洞/修补语句切片的方式，获取代码语句间特定的控制依赖及数据依赖关系，以此表征漏洞/修补模式。根据对已知漏洞的分析总结，拟根据语句类型采取不同的切片方式生成漏洞/修补模式。

3. 用于复用性漏洞规模化定位的可检索指纹标记

在生成漏洞指纹之后，需通过比对源代码和二进制代码中的相应类型特征集合，验证软件指纹生成技术的可行性。

4.3.2 固件快速解压缩及固件包直接分析技术

针对固件的快速解压缩及直接分析，在系统实现中需聚焦于固件安装包的快速提取以及固件指纹的即时可用技术。为此，在研究方法主要集中在固件安装包自动拆解技术和大规模代码特征存储设计及比对。

固件安装包自动拆解技术采用流水线并行模式，将待分析的固件安装包通过分发管理流水至处理主机，并在记录其拆解状态。

在静态拆解方面，处理主机利用 InnoUnpacker、7z 等专门打包工具对固件进行识别，解析其格式并进行提取，将可执行文件输出至分析目录。对所提取出的可执行文件进行验证，确认其为可读取、可分析。

在无法静态拆解时，采取搭建安装运行环境的方式，调试运行固件的安装过程，并进行安装状态监控，获取动态运行时生成的可执行文件。

4.4 可行性分析

本课题的研究内容和研究技术路线具有可行性，原因如下：

1. 已获取足够多的构建软件知识图谱所需要的数据；

2. 对研究内容的每个研究点都做了详尽的调研和分析，手动验证了各技术的可行性；

3. 二进制 SCA 技术已经在多个项目中发挥了重要作用，我们可以直接借鉴其成功的经验。

五　已有科研基础与所需的研究条件

5.1　团队基础与原型研发

本人所在的小组为软件知识图谱研发小组，本小组于 2017 年 4 月份成立，现组内有助理研究员两名，博士研究生一名，硕士研究生四名。小组围绕供应链安全进行了长达四年多的调研和探索，小组搭建了软件复用检测平台，验证了基于软件复用分析途径发现目标二进制漏洞的可行性。原型系统包含基于漏洞指纹的同源漏洞发现，源代码、二进制软件的代码知识图谱构建，源代码关联关系建立（OSS 复用关系发现和软件编译依赖等），源代码、二进制软件关联关系建立，软件漏洞关联关系建立等模块。

图 14 软件复用分析检测页面

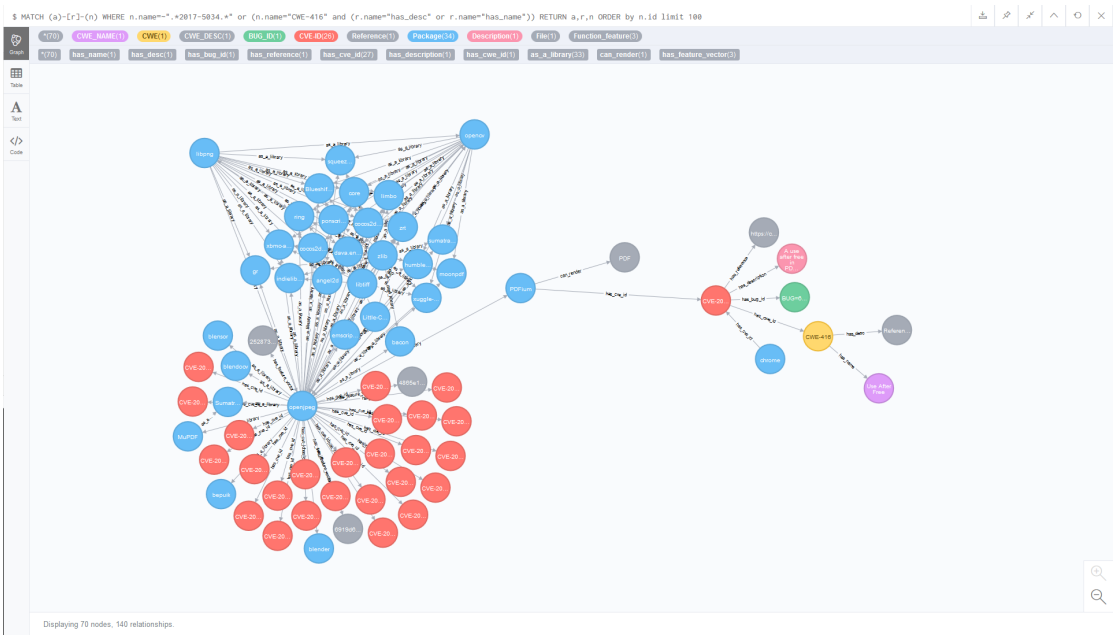

图 15 软件知识图谱示例

5.2 数据基础

软件知识图谱的构建需要大量的软件、文本原数据，这些原数据都积累在 VARAS 数据层。围绕数据积累，我们已经实现了大量的爬虫对原数据进行爬取。目前系统中已经积累的数据主要分为五个部分。

- 源代码程序：现已经实现了 Github、Gitlab、Sourceforge、Codeplex、Ubuntu 软件市场爬虫，目前已积累源代码项目 100 余万（searchcode 共收集 900 万）。
- 二进制软件：目前已经积累的 Windows 平台安装程序 2 万余个（对百度应用市场所有二进制软件进行爬取），自动安装后已生成 PE 文件 30 余万；Linux 平台下的 PE 文件 6 万余个（Ubuntu 和 Debian 的 PE 文件为 6 万 +）；同时收集了近 5000 个物联网固件和 300 多万的 Android 应用程序。
- 种子文件：系统目前已积累的 HTML、JS、PDF、图片、音视频文件等上千种种子文件数十亿；VARAS 平台产出模糊测试种子近 10T。
- 漏洞：围绕软件漏洞，我们系统目前已经收集漏洞信息 10 万余条（NVD 所有漏洞信息），收集漏洞对应的补丁近 5000 条（覆盖主流开源软件），并积累漏洞对应的 PoC 或 Exploit 两万余个。
- 文本信息：我们通过实现爬虫对 Twitter 和 vulners 等文本信息进行持续爬取，目前已经积累 Twitter 数据 900 余万条，积累 vulners 信息 50 余万条。上述

数据爬取系统目前仍在不间断爬取，并在图谱构建过程中发现更多的数据源进行爬取。

这些数据是构建软件知识图谱所必须的，同时这些目前积累的数据已经可以支持软件评测基础知识库的构建。

5.3 项目基础

本人自 2018 年开始接触软件复用检测知识以来，参加了相关项目 8 项，以核心骨干参与的项目有 3 项。下面简要介绍其中的两个比较重要的项目。

某二进制软件成分与漏洞分析项目。通过此项目的实施，积累了如下数据：

- 积累 70 万级软件版本的代码信息和漏洞信息
- 支持对 20 种代码特征的自动提取
- 自动化爬取、编译 github 高星项目
- 积累 3000 万量级代码特征

使复用检测系统拥有下述能力：

- 支持对目标二进制软件中软件成分的自动识别和漏洞的自动检测
- 支持 PE/ELF 文件分析
- 支持 7 种特征的提取与比对
- 支持多任务并行分析

某部门数据库和中间件评测项目，以及某部队操作系统评估项目，内容如下：

- 分析了两款数据库产品，分别是：人大金仓、优炫
- 分析了两款中间件产品，分别是：金蝶软件、东方通中间件
- 分析了四款操作系统：中科方德、武汉深之度、湖南麒麟、银河麒麟
- 涉及到四种处理器架构：mips64、loongarch64、aarch64、sw_64
- 累计分析 800 万级二进制软件的代码信息和漏洞信息
- 累计对 105 万检测结果进行初步分析
- 累计出具 8 份二进制检测报告
- 对操作系统评估进行对比实验，工具为：自研工具、cybellum、BinaryAI

六　研究工作计划与进度安排

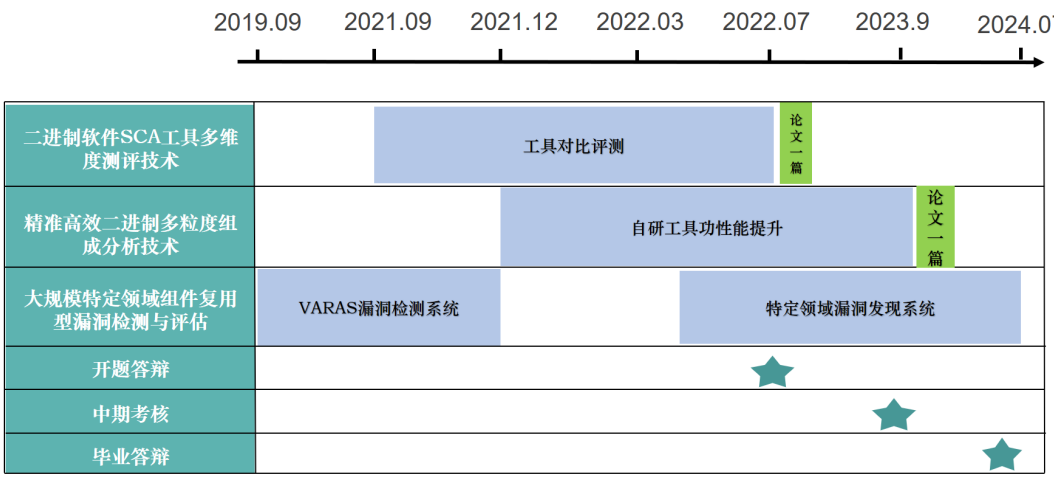

图 16 研究计划安排

如16所示，研究计划初步安排如下：

1. 2019 年 9 月到 2021 年 12 月，在建立涵盖软件、漏洞、漏洞类型、漏洞补丁等实体类别的基础上，完成漏洞检测系统的开发与完善，为工具对比评测和自研工具功性能提升提供基础保障；

2. 2021 年 9 月到 2022 年 7 月，通过数据集的扩充和 Ground Truth 的构建，完成四款学术界 SCA 工具和两款工业界 SCA 软件的功性能横向评测，找到自研工具的确切提升空间和提升方法，并投稿一篇论文；

3. 2020 年 12 月到 2023 年 9 月，设计推理规则发现开源软件与二进制软件之间的关联关系，并标记开源软件与二进制软件关联关系的种类，并完成漏洞特征（开源）和二进制软件特征（闭源）的增量提取，在此基础上完成在研工具在功性能方面的提升，并投稿一篇论文；

4. 2022 年 3 月到 2024 年 7 月，持续对软件知识图谱构建的数据进行收集，并调研相关研究工作，扩充固件将优秀的方法集成到知识图谱系统中；

5. 2023 年 9 月，完成中期考核，2024 年 7 月，完成毕业答辩。

参考文献

[1] BOARD C S R. Csrb review: December 2021 log4j event: Key findings and recommendations [EB/OL]. 2022. https://www.cisa.gov/sites/default/files/publications/CSRB-Log4J-Key-Findings-and-Recommendations-Summary-508c.pdf.

[2] 网安论坛. Log4j 漏洞：互联网历史上破坏力最惊人漏洞之一 [EB/OL]. 2022. https://www.wangan.com/p/7fy7f8f9879dbb94.

[3] CISRC. CISA 警告：黑客仍在利用 Log4Shell 漏洞 [EB/OL]. 2022. https://www.ics-cert.org.cn/portal/page/111/1726b3f9eaa849cf986cdd9e907a2951.html.

[4] checkpoint. Leader in cyber security solutions, check point software [EB/OL]. 2022. https://www.checkpoint.com/.

[5] ACSM. The numbers behind a cyber pandemic [EB/OL]. 2021. https://australiancybersecuritymagazine.com.au/the-numbers-behind-a-cyber-pandemic/.

[6] MissD. The nukesped backdoor is back! north korean hacker group lazarus exploits log4j flaw to attack vmware horizon servers [EB/OL]. 2022. https://segmentfault.com/a/1190000041893723/en.

[7] 美国国家标准技术研究院. Defending Against Software Supply Chain Attacks [EB/OL]. 2021. https://www.cisa.gov/sites/default/files/publications/defending_against_software_supply_chain_attacks_508_1.pdf.

[8] github. 近两年典型的软件供应链安全事件 [EB/OL]. 2021. https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/README.md.

[9] wikipedia. Software supply chain [EB/OL]. 2022. https://en.wikipedia.org/wiki/Software_supply_chain.

[10] SEAL. 软件成分分析 Software Compostition Analysis（SCA）[EB/OL]. 2022. https://www.seal.io/_detail/25.html.

[11] 腾讯安全玄武实验室. 腾讯玄武实验室 [EB/OL]. 2021. https://xlab.tencent.com/cn/.

[12] Adobe. 可用于 Adobe Acrobat 和 Reader 的安全更新 [EB/OL]. 2016. https://helpx.adobe.com/cn/security/products/reader/apsb16-26.html.

[13] CVE. Cve-2018-6830 [EB/OL]. 2018. https://cve.mitre.org/cgi-bin/cvename.cgi?name=2018-6830.

[14] 吴世忠. 软件漏洞分析技术 [M]. 科学出版社, 2014.

[15] wikipedia. Equation group [EB/OL]. 2016. https://en.wikipedia.org/wiki/Equation_Group.

[16] Kolias C, Kambourakis G, Stavrou A, et al. Ddos in the iot: Mirai and other botnets [J]. Computer, 2017, 50(7): 80-84.

[17] wikipedia. 2016 dyn cyberattack [EB/OL]. 2016. https://en.wikipedia.org/wiki/2016_Dyn_cyberattack.

[18] wikipedia. The shadow brokers [EB/OL]. 2016. https://en.wikipedia.org/wiki/The_Shadow_Brokers.

[19] Brokers S. Eternalblue [EB/OL]. 2017. https://en.wikipedia.org/wiki/EternalBlue.

[20] microsoft. eternalromance [EB/OL]. 2018. https://cloudblogs.microsoft.com/microsoftsecure/tag/eternalromance/.

[21] Ehrenfeld, Jesse M. Wannacry, cybersecurity and health information technology: A time to act [J]. Journal of Medical Systems, 2017, 41(7).

[22] wikipedia. Hacking team [EB/OL]. 2017. https://en.wikipedia.org/wiki/Hacking_Team.

[23] nist. National vulnerability database [EB/OL]. 2019. https://nvd.nist.gov/.

[24] cve. Cve [EB/OL]. 2019. https://cve.mitre.org/.

[25] cvedetails. cvedetails [EB/OL]. 2010. https://www.cvedetails.com/browse-by-date.php.

[26] Zhen L, Zou D, Xu S, et al. Vulpecker: an automated vulnerability detection system based on code similarity analysis [C]//Conference. 2016.

[27] Ding C. State of windows application security: Shared libraries [EB/OL]. 2017. https://www.asiasecwest.com/csw-presentation-db/2017/3/15/state-of-windows-application-security-shared-libraries.

[28] 砍柴网. 腾讯安全玄武实验室“阿图因”系统入选世界互联网大会领先科技成果 [EB/OL]. 2017. http://tech.ifeng.com/a/20171206/44793650_0.shtml.

[29] owasp. Owasp top 10 [EB/OL]. 2017. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

[30] Kim S, Woo S, Lee H, et al. Vuddy: A scalable approach for vulnerable code clone discovery [C]//2017 IEEE Symposium on Security and Privacy (SP). 2017.

[31] S?Bj?Rnsen A, Willcock J, Panas T, et al. Detecting code clones in binary executables [C]// Eighteenth International Symposium on Software Testing Analysis. 2009: 117.

[32] Kai C, Peng L, Zhang Y. Achieving accuracy and scalability simultaneously in detecting application clones on android markets [C]//International Conference on Software Engineering. 2014.

[33] ewny J, Schuster F, Bernhard L, et al. Leveraging semantic signatures for bug search in binary programs [C]//conference on computer security applications. 2014.

[34] Eschweiler S, Yakdan K, Gerhards-Padilla E. discovre: Efficient cross-architecture identification of bugs in binary code [C]//The Network and Distributed System Security Symposium (NDSS 2016). 2016.

[35] Luo L, Jiang M, Wu D, et al. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection [C]//Acm Sigsoft International Symposium on Foundations of Software Engineering. 2014.

[36] David Y, Partush N, Yahav E. Similarity of binaries through re-optimization [C]//Acm Sigplan Conference. 2017: 79-94.

[37] Ding S, Fung B, Charland P. Kam1n0: Mapreduce-based assembly clone search for reverse engineering [C]//the 22nd ACM SIGKDD International Conference. 2016.

[38] Chen K, Wang P, Lee Y, et al. Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale [C]//USENIX Security Symposium. 2015.

[39] Xu X, Chang L, Qian F, et al. Neural network-based graph embedding for cross-platform binary code similarity detection [J]. 2017.

[40] Hemel A, Kalleberg K T, Vermaas R, et al. Finding software license violations through binary code clone detection [C]//Working Conference on Mining Software Repositories. 2011.

[41] Duan R, Bijlani A, Meng X, et al. Identifying open-source license violation and 1-day security risk at large scale [C]//the 2017 ACM SIGSAC Conference. 2017.

[42] cybellum. cybellum [EB/OL]. 2022. https://cybellum.com/.

[43] binaryai. 二进制安全智能分析平台 [EB/OL]. 2021. https://www.binaryai.cn/.

[44] Bellon S, Koschke R, Antoniol G, et al. Comparison and evaluation of clone detection tools [J]. IEEE Transactions on Software Engineering, 2007.

[45] Kamiya T, Kusumoto S, Inoue K. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code [J]. IEEE Transactions on Software Engineering, 2002, 28 (7): 654-670.

[46] Svajlenko J, Roy C K. Cloneworks: A fast and flexible large-scale near-miss clone detection tool [J]. ACM, 2017.

[47] Li Z, Zhou Y. Pr-miner: automatically extracting implicit programming rules and detecting violations in large software code [J]. Acm Sigsoft Software Engineering Notes, 2005.

[48] Li Z, Shan L, Myagmar S, et al. Cp-miner: Finding copy-paste and related bugs in large-scale software code [J]. IEEE Transactions on Software Engineering, 2006, 32(3): 176-192.

[49] Jiang, LX, Misherghi, et al. Deckard: Scalable and accurate tree-based detection of code clones [J]. PROC INT CONF SOFTW ENG, 2007.

[50] Mou L, Li G, Zhang L, et al. Convolutional neural networks over tree structures for programming language processing [C]//National Conference on Artificial Intelligence. 2014.

[51] Feller J, Fitzgerald B. Understanding open source software development [J]. Itnow, 2001, 7 (1): 31.

[52] Miyani D, Huang Z, Lie D. Binpro: A tool for binary source code provenance [J]. 2017.

[53] Yamaguchi F, Lindner F, Rieck K. Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning [C]//5th USENIX Workshop on Offensive Technologies (WOOT). 2011.

[54] Yamaguchi F, Lottmann M, Rieck K. Generalized vulnerability extrapolation using abstract syntax trees [C]//Proc. of 28th Annual Computer Security Applications Conference (ACSAC). 2012.

[55] Qian F, Zhou R, Xu C, et al. Scalable graph-based bug search for firmware images [C]//Acm Sigsac Conference on Computer Communications Security. 2016.

[56] Li Z, Zou D, Xu S, et al. Vuldeepecker: A deep learning-based system for vulnerability detection [C]//Network and Distributed System Security Symposium. 2018.

[57] NNNN. Nnnnnnn 51 [J]. 2099.

[58] nnnn. nnnn 52 [C]//nnnn. 2016.

[59] N. Nnnnn 14 [C]//NN. 2017.

[60] N. Nnnnn 27 [C]//NN. 2017.

[61] n. nnnn30 [C]//n. 2017.