本科毕业设计（论文）

设计（论文）题目

Linux 内核的漏洞态势感知与预测

设计（论文）英文题目

Perception and Prediction of Vulnerability Situation of Linux Kernel

学　　院：计算机与信息技术学院

专　　业：信息安全

学生姓名：陈力恒

学　　号：14281055

指导教师：何永忠

北京交通大学

2018 年 6 月

学士论文版权使用授权书

本学士论文作者完全了解北京交通大学有关保留、使用学士论文的规定。特授权北京交通大学可以将学士论文的全部或部分内容编入有关数据库进行检索，提供阅览服务，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：　　指导教师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

中文摘要

摘要：随着计算机发展，软件安全备受关注。本文刚实现或发布的软件从不稳定状态到指定稳定状态（简称稳定化过程）的时长研究是漏洞态势研究的一个方向，指定稳定状态指一款软件的关键漏洞类型高危害程度的漏洞平均已被修复的状态。

本文研究 Linux 内核，其被广泛用于定制操作系统。我们认为刚实现或发布的 Linux 内核版本不够稳定，存在未知漏洞。预测稳定化时长，便能选取稳定版本定制系统，且为 Linux 内核开发人员提供新版本关键测试阶段时长的参考。

本论文编写爬虫工具从 NVD、GitHub、Kernel 等网站上获取 CVE 漏洞信息、补丁代码、内核源代码等。编写漏洞检测工具纠正 NVD“受漏洞影响版本”的不正确，编写代码定位工具纠正 Diff 代码中定位信息的不正确。通过分析历史数据得到漏洞危害等级走势图和关键漏洞类型。利用 BP 神经网络工具对历史数据训练建模，并预测高危害程度的关键漏洞类型的漏洞的最早存在到被发现时间间隔，加权平均求得 Linux 内核的稳定化时长。本文详细阐述了相关背景知识，阐述了从数据获取、纠正到预测的设计实现和实验结果，在文末提出了相关的结论和指导意见。

关键词：漏洞态势研究；Linux 内核；稳定化时长；漏洞检测工具；软件版本选择问题

ABSTRACT

ABSTRACT: With development of computer technology, people pay more and more attention to software security. Perception and prediction of duration from unstable stage to specified stable stage (referred to stabilization process) of a software that has just been implemented or released is one of directions of vulnerability situation research. The specified stable stage in this article refers to a stage that mainly critical and high-risk vulnerability has been repaired.

This paper focused on Linux kernel, because it's widely used to customize various operating systems. A version of Linux kernel that have just been implemented or released are not stable enough, and it tends to have a lot of unknown but serious vulnerabilities. By predicting duration of stabilization process, we can choose a stable Linux kernel version as a custom operating system's kernel. On the other hand, this can also provide Linux kernel developers with duration of critical testing phase of a new version of Linux kernel.

This paper uses crawler tools to get CVEs' information, patches and source trees of Linux kernel from websites including NVD, GitHub, Kernel and so on. This paper uses vulnerability detection tool to correct the fault of "Vulenrable Software and Version" of NVD and uses code location tool to correct the fault of code location in Diff code of GitHub and so on. Then get perceptual result of vulnerability situation, which including trend of vulnerability's level and cirtical vulnerability type. The historical data is trained by using BP neural network tool for modeling, and forecasts interval between exists and be discovered of critical high-level vulnerabilities, finally calculates duration of software stabilization. This paper elaborates the related background knowledge, elaborates from the design and implementation of data acquisition, correct, forecast to experimental results, and in the end puts forward some conclusions and guidance.

KEYWORDS： Vulnerability situation; Linux Kernel; Duration of Stabilization Process; Vulnerability detection tool; Software's version selection problem

目　　录

1 引言

本章讲述本毕业设计的研究背景、研究目的、研究内容、相关领域的工作、研究贡献、假设约束等，使得读者能对漏洞态势研究和本毕业设计能有一个初步的了解。

1.1 研究背景

如今，由于各类计算机、互联网的系统变得日益复杂，这些系统存在着许许多多的各种各样的危害程度不尽相同的漏洞。其中被利用的漏洞对计算机、互联网造成巨大的经济损失，而那些尚未被利用的漏洞使系统存在巨大的安全隐患。因此，漏洞研究越来越备受瞩目，这些关注和支持对漏洞研究提供了很好的经济基础和研究平台。

漏洞态势的研究作为漏洞研究的一部分，可以有效指导相关从业人员了解漏洞的总体情况和发展情况，从而使相关从业人员在决策中获得一定的理论和技术支持，使得决策更加准确和有说服力。本论文针对稳定化时长这一漏洞态势进行感知与预测。

以 Linux 内核为例，它是一款当下非常流行的开源电脑操作系统内核，被全球无数的程序员免费使用同时无数的程序员为 Linux 内核的开发提供无偿援助。Linux 内核的开源、免费等特征使其广泛地被应用于定制各类操作系统，其中非常著名的有 Android 系统、Ubuntu、CentOS 等，值得一提的是 Android 被广泛应用于智能电视、智能手机等各种面向大众的产品中，因此，我们可以看到 Linux 内核的稳定性非常大程度地影响了各类定制操作系统和相关产品的稳定性和安全性。我们认为那些新发布的软件版本由于尚未被广泛使用，很多关键漏洞还没有显现出来，因此这些新版本的软件一定程度上是不安全的或者是不稳定的。所以为了解决这个冲突，我们需要研究软件从刚实现或发布的时候的不稳定状态进入指定稳定状态（简称为稳定化过程）所需要的时间，以确定新发布的软件版本如 Linux 内核新版本需要多长时间的测试使用修复等工作才能够达到指定稳定状态，另一方面，可以确定以前发布的 Linux 内核的哪些版本已经到达了指定稳定状态，以应用于定制操作系统。

针对这一背景，在本论文中，以 Linux 内核为案例，详细讲述了 Linux 内核的稳定化时长这一漏洞态势的感知与预测的设计实现和结果，最终对实验结果进行分析，得出结论。

1.2 研究目的

本论文旨在预测一款软件的一个版本需要多长时间的漏洞发现、修补等工作才能完

成稳定化过程，从而在软件刚实现或发布的时候便能够一定程度地预测软件需要多长时间来实现稳定化过程，而并不需要等到软件已经经历了漫长的漏洞发现、修补工作之后才能确定软件已达到指定稳定状态。

首先，这一研究在软件的测试过程中能起到预测测试过程时长的作用。该研究可以帮助测试人员和需要考虑测试过程时长的决策人员来预测测试过程中需要多久才能使软件到达指定稳定状态，从而帮助决策层安排软件开发进度。

此外，这一研究可以帮助软件使用者选择合适的软件版本。在软件使用过程中，我们往往将刚刚发布的软件视为不够稳定的产品，即软件处于不稳定状态，那么该如何选择软件版本就会成为一个需要考虑的问题。一般来说，我们需要足够稳定的且最新的版本，目的在于这个版本的软件具备足够的稳定性并且具备更佳的性能和更完善的功能。只有在选择了合适的软件版本之后，才能展开后续工作。

以 Linux 内核为例，有许许多多的工业系统、零售系统、面向大众消费者的系统等均是以 Linux 内核为基础，进而实现定制的操作系统（如 Android 系统）。那么在 Linux 内核的版本选择过程中，所选择的 Linux 内核首先要满足稳定性的需求，其次要满足功能更完善、性能更佳的需求。那么通过本研究，将提取 Linux 内核中最关键的漏洞并预测它们的“最早存在到被发现的时间间隔”，从而确定 Linux 内核需要多长时间用于稳定化过程，进而可以判断哪些版本的 Linux 内核在当前时间下已处于指定稳定状态，我们便可以选取其中最新版本的 Linux 内核用于定制操作系统。

1.3 研究内容

我通过对实际问题进行分析，确定了本毕业设计的研究内容。问题分析和研究内容如下：

1. 如何研究一款软件的某个版本的稳定化过程所需要的时间？一款新版本的软件的漏洞往往是未知的，但我们可以根据该软件的历史数据确定该软件的发展过程中哪些漏洞是最致命、最核心的（称为关键漏洞），确定了这些漏洞之后，我们可以预测这些漏洞的最早存在到被发现修复的时间间隔，以此便可代表稳定化过程所需要的时间。
2. 如何获取软件的关键漏洞？获取并分析软件历年漏洞数据，如从漏洞危害程度、漏洞数量角度进行分析。
3. 如何获得一款软件历史数据？利用 NVD 数据库获得漏洞的危害程度、类型；利用补丁网站获取漏洞的补丁代码（补丁代码中的删除代码视为漏洞代码），利用软件源代码下载网站获取软件源代码。进而判断一个漏洞是否存在于一个特定版本的软件中及确定漏洞最早存在的时间，进而确定漏洞最早存在到被发现的

时间间隔。

4. 如何预测一个漏洞的最早存在到被发现的时间间隔？使用神经网络对历史数据中的特征因子（本论文将漏洞类型、漏洞危害程度、漏洞存在到被发现时间间隔等称为特征因子）进行训练，最后用于预测漏洞的最早存在到被发现的时间间隔这一特征因子。

因此，本毕业设计的研究流程如下：

1. 获取与纠正历史漏洞数据。
2. 对漏洞数据进行分析得出漏洞态势感知结果，其中包括软件的关键漏洞类型。
3. 对历史漏洞数据运用神经网络进行训练并得出较为精准的模型。
4. 利用训练出来的模型对高危害程度的关键漏洞类型的漏洞进行预测，得出高危害程度的关键漏洞类型的漏洞的最早存在到被发现时间间隔。
5. 最终确定软件稳定化过程所需时长。

1.4 相关领域

- **前人工作**

一方面，以 NVD 为例的漏洞数据库已到达一定的成熟程度，给我们在研究中获取数据这一环节提供了极大的便利。另一方面，开始出现以这些数据库为基础研究漏洞态势的论文，只是有些论文的研究深度和指导意义可能不够理想。

通过阅读漏洞态势相关的论文[1]-[14]，可以看到以往研究漏洞态势的论文主要旨在反应漏洞的趋势如何，其中有的论文提出了漏洞态势预测的思路，如通过历史数据曲线拟合的方法来实现预测。这些论文在给我提供研究思路这方面起了非常重要的作用，让我能够抓住漏洞的关键特征，并构思如何实现本毕业设计的漏洞态势的预测，从而展开并实现本毕业设计。

- **知识空白**

目前尚未出现对漏洞的最早存在到被发现的时间间隔与漏洞其他特征因子之间的关系的研究，因此也无法利用漏洞最早存在到被发现的时间间隔去指导测试人员预测软件测试过程时长，以及指导软件使用者选择软件版本。本毕业设计针对这一尚未解决的问题获取、纠正历史数据并提出了相关的预测方法和模型。

1.5 研究贡献

本节，我将阐述本毕业设计对相关领域做出的贡献、参考价值以及本毕业设计的创新点。具体贡献和创新点如下：

- 本论文编写了 Linux 内核有关的数据获取工具，获取了大量的 Linux 内核的漏洞数据和补丁等，其数量级在 1000，保证了研究的有效性。这些工具和数据为相关领域的研究提供数据支持的作用，从而极大地节省了未来研究的数据获取的成本。漏洞感知方法和对历史数据的漏洞感知结果能够为相关领域的研究提供直观的有意义的参考。
- NVD 和 GitHub 部分信息存在的错误予以纠正。第一点，在本毕业设计实验过程中发现 NVD 的受漏洞影响版本数据存在错误，因此我编写了漏洞检测工具来纠正该错误，该工具中还必须实现代码预处理工具、代码匹配工具和代码定位工具。第二点，在如 GitHub 发布的具体的补丁代码中存在一行 Diff 段的头信息（如“@@ -712,6 +712,7 @@ struct sctp_chunk {”），该信息主要为了反映漏洞的一个 Diff 段存在于漏洞文件的哪些行和哪些函数体或结构体中，但是该信息存在一定程度的错误，因此我编写了代码定位工具来解决这个问题。
- 本毕业设计首次以漏洞最早存在到被发现时间间隔为关键因子展开研究。探究漏洞最早存在到被发现时间间隔与漏洞类型、漏洞危害程度之间的关系，这是以往的研究没有涉及到的。漏洞最早存在到被发现时间间隔的研究和预测为基于 Linux 内核开发定制系统的人员和 Linux 内核测试人员提供决策支持。
- 本论文利用神经网络技术来实现稳定化时长这一漏洞态势的预测，不再纠结于传统的数学公式来拟合预测，这为相关领域试图使用神经网络进行研究提供了参考经验。
- 本论文以 Linux 内核为案例来研究探讨，其中编写的大部分工具、经验可以很好地应用于其他软件中，如漏洞检测工具、代码定位工具、BP 神经网络工具等。

1.6 假设约束

本节主要讲述本毕业设计的假设和约束，这些假设和约束是在具体实验过程中得出的，为了说明本毕业设计的结论的适用范围，并保证结论的质量。具体如下：

- 在本毕业设计中，暂不考虑 Linux 内核各版本之间的在关键漏洞类型方面的差异。
- Linux 内核版本采用 80 个左右的部分正式版，暂不考虑测试版本和其他版本。
- 本次实验考虑的是 Linux 内核的漏洞最早存在到被发现时间间隔与漏洞类型、漏洞严重程度之间的关联性。
- BP 神经网络的结构为 125-25-300（即包括输入层在内共三层的神经网络，输入层结点 125 个，隐含层结点 25 个，输出层结点 300 个）。
- 本人实现的漏洞检测工具未考虑漏洞文件名被修改这一类问题。

- 本毕业设计将高危害的 CWE-119、CWE-189、CWE-20、CWE-200、CWE-264、CWE-362、CWE-399、CWE-416 类型的漏洞视为 Linux 内核的关键漏洞，这些关键漏洞的最早存在到被发现时间间隔来代表 Linux 内核稳定化过程所需要的时间，即我们认为 Linux 某个版本内核的稳定状态是指上述类型的高危漏洞平均而言被修复的阶段。对于稳定状态的要求需根据具体情况而定。

1.7 论文结构

第 1 章：引言。讲述本毕业设计的研究背景、研究目的、研究内容、相关领域的工作、研究贡献、假设约束等，使得读者能对漏洞态势研究和本毕业设计能有一个初步的了解。
第 2 章：背景知识。讲述阅读本毕业设计可能需要用到的背景知识，以便读者更好地理解本文叙述的内容。
第 3 章：设计与实现。首先提出本毕业设计的研究方案，然后讲述本毕业设计的设计和实现，让读者能够对本毕业设计有一个详细的了解，并能够更好地理解本毕业设计的代码实现、实验结果和结论。
第 4 章：实验结果。展示利用本毕业设计最终得到的结果，使得读者对本毕业设计的工作内容能有直观的了解，并为第五章结论的得出提供基础。
第 5 章：论文结论。展示本毕业设计最终得到的结论和意义，体现本毕业设计的价值。

1.8 本章小结

本章主要阐述了稳定化时长这一漏洞态势的相关背景，并根据此背景得出目前需要解决的问题和本毕业设计的目的，并对问题分析以确定本毕业设计的研究方向和思路，同时阐述了相关领域其他人的贡献，并提出本毕业设计的创新点和贡献，最后阐述了本毕业设计所得出的结论所适用的假设和约束。

2 背景知识

本章讲述阅读本毕业设计可能需要用到的背景知识，以便读者更好地理解本文叙述的内容。

2.1 词汇说明

AM: After Modification,文中主要标志漏洞修改后代码文件
BM: Before Modification,文中主要标志漏洞修改前代码文件
BPNN: Back Propagation Neural Network
CPE: Common Platform Enumeration
CVE: Common Vulnerabilities and Exposures
CWE: Common Weakness Enumeration
NVD: National Vulnerability Database
漏洞：本文中主要指 CVE
漏洞代码：补丁代码中的删除代码
漏洞态势：本文中主要指漏洞态势中的一个方向——稳定化时长
模块：C 语言中的函数体和结构体在本文称为模块
稳定化过程：软件从不稳定状态到指定稳定状态的过程
稳定状态：这个在本文中指关键漏洞类型的高危害的漏洞平均而言被修复的状态
关键漏洞类型：根据漏洞在 Linux 内核中的重要程度确定的漏洞类型
关键漏洞：本文指高危害程度的关键漏洞类型的漏洞
Diff 段：补丁代码（Diff 代码）中的其中一段，代表了漏洞文件中每一处所需要修改的那一段代码
Module-BM 文件:存储 BM 文件的模块的关键信息的文件
Module-Diff 文件:存储 Diff 段所在模块的关键信息的文件

2.2 NVD

NVD 数据库中包含了以“CVE-年份-数字”为标题记载的许多漏洞，在官网中可以通过输入关键词来搜寻所需要的 CVE 信息。NVD 中的每一个 CVE 都具有一个 CVE 详细信息的页面，其中包括了漏洞描述、漏洞分析、漏洞类型、参考链接等诸多信息。NVD 给漏洞研究者们提供了很好的数据获取平台，极大地降低了漏洞信息获取的成本。当然

NVD 的功能远非如此，具体内容读者可以进入 NVD 官网一探究竟。

CVE 中受漏洞影响的版本信息以 CPE 形式表示，如“cpe:2.3:o:linux:linux_kernel:1.2.0:*:*:*:*:*:*:*”，其中包含了具体的名称（linux_kernel）和版本号（1.2.0）。本论文后续会介绍该信息存在一定程度的不完善和错误。

CVE 中的漏洞类型以 CWE 形式表示，如“CWE-119”，表示第 119 类漏洞——Buffer Errors。这一信息方便了本毕业设计确定每一个漏洞的漏洞类型，提供了漏洞态势预测模型的训练中至关重要的因子之一。

2.3 Linux 内核补丁相关说明

软件与漏洞的关系：每一款软件都会包含若干漏洞（在本文中特指 CVE），有的 CVE 会有补丁代码可供参考，这些补丁代码是用于对存在漏洞的软件版本进行维护升级。在本毕业设计中，这些补丁代码还用于漏洞检测，以实现对获取的历史数据进行纠正这一关键点。

补丁与补丁代码文件的关系：对于一个漏洞而言，往往会涉及多个文件，即多个文件同时存在漏洞代码并以整体表示成这个漏洞，其中每一个文件都是漏洞文件，对应的补丁也会存在若干个相应的文件，我们将其称为补丁代码文件，也就是说一个补丁会存在若干个补丁代码文件。

补丁代码文件与 Diff 段的关系：每一个补丁代码文件中包含了多个 Diff 段，也就是说在一个漏洞文件中，有多处代码需要进行修改，而这每一处代码所需要执行的修改就称为 Diff 段。

Diff 段的结构：每一个 Diff 段中包含 Diff 头、补丁代码。Diff 头是指形如“@@ -712,6 +712,7 @@ struct sctp_chunk {”的关于 Diff 段的说明，旨在说明 Diff 段所在的行号和函数体或结构体的名称。本文将函数体、结构体简称为模块。

Diff 段的补丁代码的结构：删除代码、增加代码、非增加非删除代码，其中删除代码视为漏洞代码，代表软件原版本中存在的漏洞。

2.4 BP 神经网络

BP(back propagation)神经网络是 1986 年由 Rumelhart 和 McClelland 为首的科学家提出的概念，是一种按照误差逆向传播算法训练的多层前馈神经网络，是目前应用最广泛的神经网络。

BP 神经网络通过对数据进行训练之后用于预测，训练中最为关键的成果是神经网络中各层之间的权重值。BP 神经网络的训练步骤主要为前馈和反馈过程。在前馈中，

将输入层的数据通过逐层的权重计算，得到输出层的数据，当输出层的数据不够贴近输出参考值的时候（即损失函数不够小），那么需要对输出值和参考输出值进行求差值，并通过 S 型函数确定误差，并通过权重逐层地反馈，以确定权重层需要修改的量以修改权重，当损失函数足够小便停止训练，这时的权重便是最终所需要的权重，是这一个模型的核心。另一方面，BP 神经网络的测试和预测便是对测试、预测输入进行一次前馈过程。

2.5 本章小结

本章阐明了本文所使用的某些词汇的具体含义、NVD 的内容结构和使用方法、Linux 内核补丁的内容结构以及 BP 神经网络的基本知识，这些知识将在研究中和后续论文中被频繁涉及，因此理解这些背景知识是一个必不可少的环节。

3 设计与实现

本章首先提出本毕业设计的研究方案，然后讲述本毕业设计的设计和实现，让读者能够对本毕业设计有一个详细的了解，并能够更好地理解本毕业设计的代码实现、实验结果和结论。

3.1 研究方案

通过第一章对研究问题和研究内容的探讨，可以得到实现并解决研究问题的思路。具体的研究方案如下：

1. 确定软件名称为 Linux 内核；
2. 通过 NVD 获取软件相应的 CVE 信息，如漏洞类型、漏洞危害程度；
3. 通过 GitHub 等网站获取每一个 CVE 的漏洞发现及修复时间、补丁代码；
4. 在 Linux Kernel 官网中下载所有版本的 Linux 内核源代码；
5. 利用 CVE 对应的补丁代码和所有版本的 Linux 内核源代码，通过漏洞检测工具判断 CVE 所存在的那些 Linux 内核版本；
6. 利用 CVE 漏洞最早存在的版本计算漏洞最早存在的时间；
7. 计算得出漏洞最早存在到被发现的时间间隔；
8. 所需要的漏洞特征因子获取完毕；
9. 将漏洞特征因子用于神经网络的训练；
10. 根据漏洞危害程度、漏洞数量、关注程度等确定软件的关键漏洞类型；
11. 将关键漏洞的特征因子输入神经网络得到预测的最早存在到被发现的时间间隔
12. 对预测得到的关键漏洞的最早存在到被发现的时间间隔通过加权平均等方法得到软件稳定化过程所需要的时间

据此绘制的研究方案流程图见图 3-1。

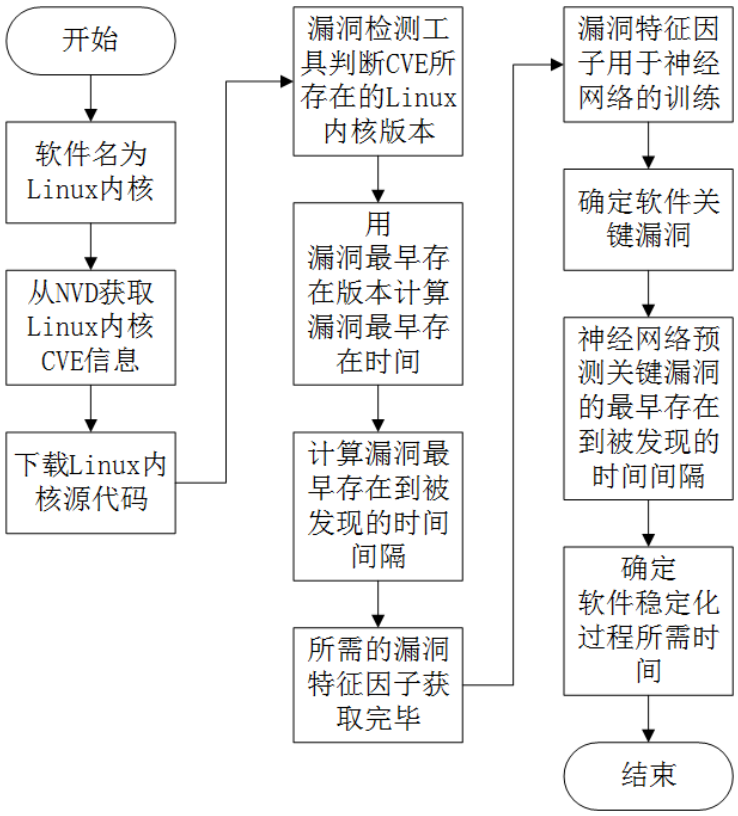

图 3-1 研究方案流程图

3.2 漏洞数据获取与筛选

本节主要讲述了稳定化时长这一漏洞态势的感知的设计实现过程中的漏洞数据获取、筛选工具的设计实现，为稳定化时长这一漏洞态势的感知结果的得出提供基础，并为稳定化时长预测模型提供可训练的样本。由于获取的部分数据不准确，因此需要下一节进行对漏洞数据的纠正。

1. **第一步。**我编写了“CVE 信息爬虫工具”用于获取 Linux 内核的所有 CVE 信息，将其保存到本地的 Excel 中，方便后续的研究分析，其中 Excel 的每一行代表一条 CVE 的所有信息（每一款软件往往有多个 CVE，即多个漏洞）。

   CVE 信息爬虫工具的设计实现：从指定 Excel 表格（如 OSSList.xls）中读取所需获取 CVE 信息的软件名称、关键词，利用 Requests 模块获取 NVD 查询结果网页（URL 的样式如 https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=keyword&search_type=all&startIndex=0）的源代码。在 NVD 查询结果网页的源代码中利用 CSS 选择器“strong[data-testid=vuln-matching-records-count]”定位并捕获该软件有关的 CVE 总数量，只有当 CVE 数量大于 0 时才继续后续的工作并确定 NVD 查

询结果网页 URL 中的“start index”值的范围。利用“div#row table[data-testid=vuln-results-table] tbody tr th strong a”定位到网页中所有的 CVE 编号，并为下一步确定每一个 CVE 的详细信息网页的 URL 打下基础。通过“https://nvd.nist.gov/vuln/detail/+CVE 编号”获得每一个 CVE 的详细信息网页的 URL，同样利用 CSS 选择器在 CVE 详细信息网页的源代码中获取所有的 CVE 信息。最后初始化一个 Excel 表格并往其中写入数据。

2. **第二步。**对获得的 CVE 信息中的参考链接根据其主网站站点 URL（如 git.kernel.org）进行计数，确定数量足够多且具有补丁代码的站点。

   CVE 参考链接统计工具的设计实现：在保存 CVE 信息的 Excel 中利用正则表达式“https?://[^/]*”查找所有以“http”开头的单元格并对该站点进行计数，最终保存在一个 Excel 表格中。手动确定各个站点的网页内容是否包含补丁代码，并最终确定数量足够多且具有补丁代码的站点。

3. **第三步。**从第二步的结果中确定数量足够多且具有补丁代码的站点之后（最终结果为“git.kernel.org”、“patchwork.kernel.org”、“github.com”这三个站点），将通过这些站点运用“补丁下载工具”将补丁保存到本地，用于之后的漏洞检测工具中。

   补丁下载工具：确定 NVD 查询结果网页并打开之，获取 NVD 查询结果网页的源代码，捕获所有的 CVE 编号并以此进入 CVE 详细信息网页。对每一条 CVE 的所有参考链接依序判断其是否为补丁代码链接（有补丁代码的参考链接），如果有则进入该网页将补丁代码保存至本地，之后对下一个 CVE 进行相同操作，如果没有则判断下一个参考链接（其中，每一个站点的网页样式、源代码各不相同，本毕业设计对每一个站点都需要具体处理）。最终保存的目录结构为每一个 CVE 编号均作为一个目录，其中包含补丁代码、补丁代码的来源链接、BM 文件（漏洞修改前该漏洞涉及到的文件的源代码文件）和 AM 文件（漏洞修改后漏洞涉及到的文件的源代码文件）。

4. **第四步。**这一步关于特征因子的选取。以 Linux 内核为案例，通过阅读相关论文[1]-[14]，我初步归纳了关于漏洞的那些比较重要的属性，具体如下：

   - 漏洞危害等级趋势：高、中、低
   - 漏洞影响程度：完全、部分、无
   - 漏洞类型趋势：Configuration……
   - 漏洞层次趋势：设计层、实现层、配置层
   - 漏洞来源：供应商、服务商
   - 漏洞数量：如漏洞数量在各年份的数量
   - 漏洞平均发现率：如平均一个月发现 2.9 个
   - 漏洞平均修补率

- 操作系统漏洞分布
- 漏洞危害类型：Disclosure、DOS、Theft of Service、Corruption、System Compromise
- System Compromise 原因：Run Arbitrary Code、Elevate Privileges、User Account Break-in、Root Break-in
- Corrective Actions Offered：Completely Vendor Solution、No Solution、Workaround、Average Time to Patch
- 漏洞的连接方式：网络、邻接网、本地

通过筛选，最终确定对本毕业设计而言最为重要的那些属性作为本毕业设计重点讨论的特征因子，具体如下：

- CVE Number：CVE 编号
- Linux Kernel Version Number：Linux 内核中最早存在该漏洞的版本号
- Vulnerability Exist Time：漏洞最早存在的时间
- Vulnerability Discover Time：漏洞被发现的时间
- Interval Between Exist And Discover：漏洞发现时间与最早存在时间的时间差
- Vulnerability Type：漏洞类型
- CVSS Severity V2：第二代 CVSS 评分体系
  - Base Score：基础评分
  - Attack Vector：攻击向量
  - Access Complexity：连接复杂性
  - Authentication：身份认证
  - Confidentiality Impact：机密性影响
  - Integrity Impact：完整性影响
  - Availability Impact：可用性影响

5. **第五步。**通过“特征因子爬虫工具”收集一款软件的所有漏洞的特征因子，为之后的漏洞态势预测工具提供可训练的样本。其中 NVD 发布的受漏洞影响的版本该信息被证实有错误（在第六步进行讨论）。

特征因子爬虫工具的设计实现：大致与 CVE 信息爬虫工具类似，但是捕获的信息为特征因子。特征因子中的漏洞被发现时间是优先从补丁网站搜集，如果不存在补丁网站，则以 CVE 的发布时间作为漏洞被发现时间，特征因子中的漏洞最早存在的版本是从 CVE 中的受影响版本中选取的最早发布的版本（该信息后被证实有错误），将受漏洞影响的版本通过查询“版本发布时间表”便可得到漏洞最早存在的时间。将漏洞被发现时间与漏洞最早存在的时间相减，便可得到漏洞最早存在到被发现的时间间隔。

6. **第六步。**这一步证明 NVD 中的受漏洞影响版本信息存在错误。以 CVE-2017-8069 为例，NVD 显示漏洞最早存在的版本为 Linux 内核的 4.9 版本中（见图 3-2），然而该漏洞在 Linux 内核 2.6.22 版本中已经存在，即图 3-3 中的漏洞代码存在于图 3-4 的 Linux 内核 2.6.22 版本的漏洞文件中。因此可以证明 NVD 中的受漏洞影响版本信息存在错误。

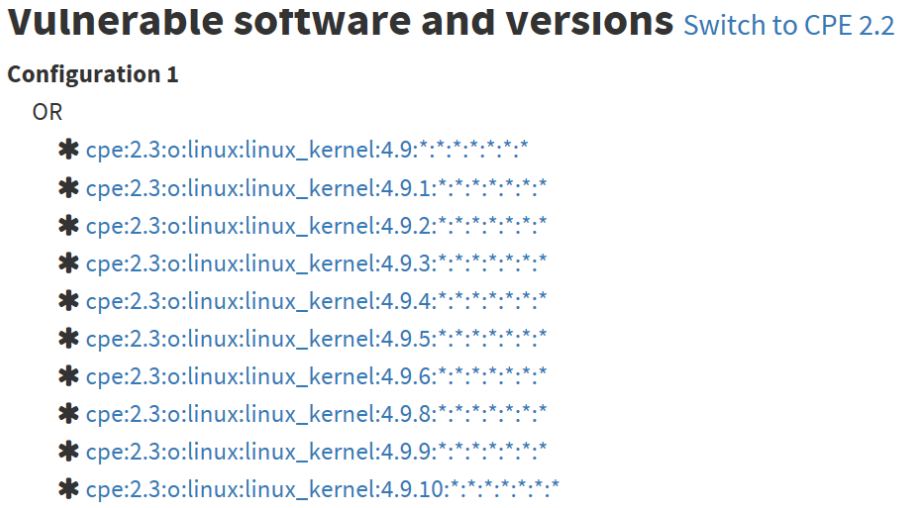

图 3-2 NVD 发布的受 CVE-2017-8069 影响的 Linux 内核版本号截图

```
diff --git a/drivers/net/usb/rtl8150.c b/drivers/net/usb/rtl8150.c
index 95b7bd0..c81c791 100644
--- a/drivers/net/usb/rtl8150.c
+++ b/drivers/net/usb/rtl8150.c
@@ -155,16 +155,36 @@ static const char driver_name [] = "rtl8150";
 */
 static int get_registers(rtl8150_t * dev, u16 indx, u16 size, void *data)
 {
-       return usb_control_msg(dev->udev, usb_rcvctrlpipe(dev->udev, 0),
-                              RTL8150_REQ_GET_REGS, RTL8150_REQT_READ,
-                              indx, 0, data, size, 500);
+       void *buf;
+       int ret;
+
+       buf = kmalloc(size, GFP_NOIO);
+       if (!buf)
+               return -ENOMEM;
+
+       ret = usb_control_msg(dev->udev, usb_rcvctrlpipe(dev->udev, 0),
+                             RTL8150_REQ_GET_REGS, RTL8150_REQT_READ,
+                             indx, 0, buf, size, 500);
+       if (ret > 0 && ret <= size)
+               memcpy(data, buf, ret);
+       kfree(buf);
+       return ret;
 }

-static int set_registers(rtl8150_t * dev, u16 indx, u16 size, void *data)
+static int set_registers(rtl8150_t * dev, u16 indx, u16 size, const void *da
 {
-       return usb_control_msg(dev->udev, usb_sndctrlpipe(dev->udev, 0),
-                              RTL8150_REQ_SET_REGS, RTL8150_REQT_WRITE,
-                              indx, 0, data, size, 500);
+       void *buf;
+       int ret;
+
```

图 3-3 CVE-2017-8069 对应的补丁代码部分截图

```
static int get_registers(rtl8150_t * dev, u16 indx, u16 size, void *data)
{
	return usb_control_msg(dev->udev, usb_rcvctrlpipe(dev->udev, 0),
			       RTL8150_REQ_GET_REGS, RTL8150_REQT_READ,
			       indx, 0, data, size, 500);
}

static int set_registers(rtl8150_t * dev, u16 indx, u16 size, void *data)
{
	return usb_control_msg(dev->udev, usb_sndctrlpipe(dev->udev, 0),
			       RTL8150_REQ_SET_REGS, RTL8150_REQT_WRITE,
			       indx, 0, data, size, 500);
}

static void ctrl_callback(struct urb *urb)
{
	rtl8150_t *dev;

	switch (urb->status) {
	case 0:
		break;
	case -EINPROGRESS:
		break;
	case -ENOENT:
		break;
```

图 3-4 Linux 内核 2.6.22 版本的 rtl8150.c 代码的部分截图

3.3 漏洞数据纠正

本小节旨在对上一节中获取数据进行纠正，以使漏洞态势感知与预测的结果更加准确和有说服力。

7. **第七步。**由于 NVD 中的受漏洞影响版本信息存在错误，为了纠正该错误，使所有特征因子都尽可能准确，必须编写一个“漏洞检测工具”。利用漏洞检测工具对一个 CVE 在所有 Linux 内核版本中判断，并选取最早的 Linux 内核版本，这就是漏洞最早存在的版本了，再查询 Linux 内核发布时间表便可得到漏洞最早存在的时间。将漏洞最早存在的时间与漏洞被发现的时间相减便可得到漏洞最早存在到被发现时间间隔。值得一提的是漏洞检测工具的实现需要依靠代码预处理工具、代码匹配工具、代码定位工具的实现。

   漏洞检测工具的设计实现：再次说明，一款软件往往存在若干个 CVE 条目（即多个漏洞）。每一个 CVE 条目往往包含若干个补丁代码文件（即一个漏洞涉及多个文件，这些文件均需要修改）。每一个补丁代码文件对应 Linux 内核的一个漏洞文件。每一个补丁代码文件往往具有多个 Diff 段（即一个漏洞文件中有多处地方需要修改）。本论文将补丁代码中的删除代码被视为漏洞代码，将 C 语言的函数、结构体都

视为模块。

以一项 CVE 的所有补丁代码文件（在代码实现过程中，一个补丁代码文件的文件名是该补丁代码对应的文件的路径名，其中路径分隔符替换成“~!@#”以防止操作系统不要将文件名错误地理解为路径名）和一个版本的 Linux 内核源代码树作为输入，以存在与否作为输出。

一个漏洞往往涉及多个文件，这些文件中都存在漏洞代码，因此这个漏洞对应的补丁表现为多个补丁代码文件，因此需要一个一个文件地判断漏洞代码是否存在于该版本内核的对应文件中。对于其中一个补丁代码文件，通过补丁代码的文件名确定该补丁代码相关的文件路径，据此找到该补丁相关的漏洞文件。

一个补丁代码文件中往往存在若干个 Diff 段，因此需要一一判断一个 Diff 段是否存在于漏洞文件中，由于如果只将 Diff 段中的删除代码（即漏洞代码）对漏洞文件全文匹配，那么可能在不相关的函数中查找到漏洞代码，这样就明显存在问题，因此通过代码定位工具确定漏洞代码所在模块名，然后再将漏洞代码在疑似漏洞文件的对应模块中判断是否存在，如果定位不到模块名，则在疑似漏洞文件中全文匹配。

代码预处理工具的设计实现：将代码进行预处理，以方便代码匹配等更为复杂的工作。比如将“static int crypto_ccm_auth(struct aead_request *req, struct scatterlist *plain,”转化为“static int crypto_ccm_auth (struct aead_request * req , struct scatterlist * plain ,”，也就是说将所有非字符与字符用空格进行隔开，同时去掉首尾的空格。只有这样，才能避免因为代码书写格式不一样的问题导致代码被判定为不同。

代码匹配工具的设计实现：我思考过两种方案：

- 方案一：判断一段代码是否按顺序出现在另一段代码中
- 方案二：一段代码与另一段代码完全相同

在实践过程中，由于一个 Diff 段中的漏洞代码在某些版本的内核的漏洞文件的漏洞代码处，其中间可能被穿插其他的代码，方案二不能在此条件下起作用。因此采用方案一。

判断代码段 A 是否依序出现于代码段 B。以代码段 A、代码段 B 作为输入，以是否存在作为输出。

每一次选取代码段 A 中的一行代码，进行代码预处理，然后与代码段 B 中的第一行开始逐行匹配（代码段 B 中的每一行代码也需要通过代码预处理）。如果代码段 A 中的每一行代码都依序出现于代码段 B 中，那么就输出存在。

代码定位工具的设计实现：该工具主要是针对 C 语言而设计的，由于 Linux 内核中 C 语言文件占了 77%（以 Linux 内核 4.0.1 版本为例，表 4-2），因此这个工具是有很大作用的。同时该工具还需要用到漏洞修改前代码文件，根据补丁代码来源

统计，存在漏洞修改前代码文件的补丁网站占了 99%（表 4-1）。

通过该工具，可以判断一段代码是否存在于一个目标文件中的结构体、函数体里面，如果是的话，返回结构体名或函数名，如果不是的话，则返回空，说明代码在后续漏洞检测工具中必须在可能的漏洞文件中进行全文匹配。

该工具可以首先将第三步中下载下来的代码中的“漏洞修改前代码”进行提炼，即将该文件的所有函数体、结构体都提炼为模块名称、模块信息及在文件中的首尾行号。该工具可以对 Linux 内核中可能存在漏洞的文件也进行相同的提炼。该工具还可以将补丁代码文件中的每一个 Diff 段根据其所在的模块，另外保存一份 Diff 段所在模块的模块名、首尾行号的文件。这样便可以很好的帮助漏洞检测工具的实现，并提高其效率。具体设计实现如下：

1. 首先，在一个代码文件中，根据正则表达式“\bstruct (\w+) {”提取所有的结构体名，根据正则表达式“(?!if|while)(\w+) (?:*)*(?!if|while)(\w+) \([^)]*\) {”提取所有的函数体名，这样就得到了所有的模块名。
2. 之后，重新从头遍历这个文件，找到第一个模块名，然后根据“{”“}”配对的原则，确定模块名后紧接着的“{”，并确定与该“{”匹配的“}”，然后将模块名的行号和“}”的行号同模块名一起保存在本地（为了节省代码定位工具所占用的时间，即避免重复对一个文件进行提炼）。后续所有的模块名均按此进行操作。
3. 上述步骤完成了代码模块提炼的工作。最初先完成漏洞修改前代码文件的代码模块提炼工作，之后利用代码匹配工具判断 Diff 段中的漏洞代码是否存在于一个模块，然后将这个模块的模块信息和首尾行号保存起来，说明该 Diff 段存在于这个模块中。
4. 代码定位工具的工作完成后，在漏洞检测工具中，Diff 段根据其所在的模块名在 Linux 内核的疑似漏洞文件中确定模块，然后将 Diff 段中的漏洞代码在 Linux 内核的该模块中利用代码匹配工具进行匹配，最终判断该 Diff 段是否存在。

3.4 漏洞态势感知

本小节具体阐述了漏洞态势感知模型，讲述了如何获得基本的漏洞趋势以及确定本毕业设计中非常重要的关键漏洞类型。

8. **第八步。**通过对纠正后的数据进行分析，我们首先根据每年发现的不同漏洞危害等级的漏洞数量绘制漏洞危害等级走势图，以便获得对漏洞趋势的直观的了解。
9. **第九步。**由于本毕业设计的关键问题是如何预测一款软件如 Linux 内核的稳定化过

程所需时长，因此我们需要确定 Linux 内核的关键漏洞类型并预测其高危害等级（漏洞危害等级的选取以实际需求为参考，本毕业设计中认为稳定的软件指的是一款软件的重要的高危的漏洞平均而言被修复，从而极大程度地避免了严重问题的出现）漏洞最早存在到被发现时间间隔以代表漏洞稳定化过程所需时长，那么如何获取 Linux 内核的关键漏洞类型呢？本毕业设计提出了四种确定关键漏洞类型的标准，具体如下：

- 根据各种漏洞类型的漏洞数量确定关键漏洞类型。
- 根据高危害漏洞中各种漏洞类型的漏洞数量确定关键漏洞类型。
- 首先利用公式“漏洞类型影响程度=1*低危害漏洞数量+2*中危害漏洞数量+3*高危害漏洞数量”确定各种漏洞类型的影响程度，然后选取漏洞影响程度最大的漏洞类型作为关键漏洞类型。
- 首先利用公式“漏洞类型影响程度=Σ该漏洞类型的每个漏洞的 CVSSv2 评分”确定各种漏洞类型的影响程度，然后选取漏洞影响程度最大的漏洞类型作为关键漏洞类型。

至此，我们便可获得关键漏洞类型，便能用于漏洞态势的预测了。在本毕业设计中，是将以上四个标准下的关键漏洞类型都作为需要考虑的关键漏洞类型。

3.5 漏洞态势预测

本小节阐述漏洞态势预测的模型设计实现和最终稳定化过程所需时长结果的得出方法。

10. **第十步。**将之前得到的准确的所有特征因子数据进行分组——训练组和测试组，用训练组在 BP 神经网络中进行建模，最终可以得到 BP 神经网路的各层之间的权重。将训练阶段得到的权重对测试组数据进行测试，确定模型的准确度。

BP 神经网路预测工具的设计实现：我同时采用了 Pybrain、我自己写的 BP 神经网络来建立神经网络。由于 Pybrain 会将样本分成训练组、验证组、测试组，当训练组和验证组的误差相近时，便会停止迭代，为了让迭代次数足够多，因此，我自己写了一个 BP 神经网路。

关于 Pybrain 实现 BP 神经网络比较简单，个人在此不再熬述。下面是关于我个人编写的 BP 神经网络。

根据输入的神经网络结构（如“[125, 25, 300]”），初始化各层的结点、各层之间的权重。由于在输出层可能是为了输出多个分类，如漏洞最早存在到被发现的时间间隔是在 0 个月到 300 个月之间，每一个月数都视为一个分类（因为我在反馈的过程中采用高确定度减小预测误差的方法，即以 Sigmoid 函数的导数去乘以输出差值，

因此该值总是往 0，1 这两个方向靠拢。），因此我才用独热的方式表示分类输出，如 300 个结点的独热输出。

在前馈过程中，利用 Numpy 模块实现矩阵运算加快前馈运算。在反馈过程中，利用 Sigmoid 函数的导数去乘以该层输出差值得到该层的输出误差，将前一层的转置乘以该层的输出误差作为两层之间的权重的误差，然后再将该层的输出误差通过权重反馈给前一层作为前一层的输出差值。其中在权重学习过程中，学习率这一参数利用学习退火算法使其在学习后期避免出现学习震荡现象（本人采用的方法是每 N 步，学习率减半）。

测试过程便是根据最新的权重进行一次前馈。

11. **第十一步。**确定 BP 神经网络预测工具满足要求的精度后，将高危害等级的关键漏洞类型的漏洞进行预测，获得关键漏洞的漏洞最早存在到被发现时间间隔，并以每个漏洞类型的漏洞数量为权重加权平均确定 Linux 内核的稳定化过程所需时长。

3.6 本章小结

本章首先确定了研究思路和流程，然后以分步的方式顺序地讲解了本毕业设计的研究过程，并在每一步中讲解具体的设计与实现，尤其是给出了漏洞态势感知方法和最终讲解了如何得到本毕业设计的关键问题的解答：Linux 内核稳定化过程所需时长。

4 实验结果

本章展示利用本毕业设计最终得到的结果，使得读者对本毕业设计的工作内容能有直观的了解，并为第五章的结论得出提供基础。

4.1 漏洞数据获取和筛选

本小节主要阐述了漏洞数据获取和筛选的结果，这些数据是后续结果得出的基础。

通过 CVE 信息爬虫工具，将所有 CVE 的各项信息均保存至 Linux 内核 CVE 信息表中（图 4-1，图 4-2），各项信息包括 CVE 编号、漏洞描述、分析描述、漏洞类型、CVSS 评分、参考链接等等。

				CVSS Severity V3			
CVE Number	Current	Analysis	Vulnerability Type	Base	Vector	Impact	Exploitability
CVE-2018-7492	A NULL pointer de						
CVE-2018-7480	The blkcg_init_que						
CVE-2018-7273	In the Linux kernel						
CVE-2018-6927	The futex_requeue						
CVE-2018-6412	In the fur	In the fun	Information Leak / D	7.5	CVSS:3.	3.6	3.9
CVE-2018-5750	The acpi	The acpi_	Information Leak / D	5.5	CVSS:3.	3.6	1.8
CVE-2018-5703	The tcp_	The tcp_\	Out-of-bounds Write	9.8	CVSS:3.	5.9	3.9
CVE-2018-5344	In the Lir	In the Lin	Use After Free (CW	7.8	CVSS:3.	5.9	1.8

图 4-1 Linux 内核 CVE 信息表部分截图（左半部分）

CVSS Severity V2				Reference 1	
Base	Vector	Impact	Exploitability	HyperlinkHyperlink	Resource
				http://git.kernel.org/c	
				http://git.kernel.org/c	
				http://www.securityfc	
				http://git.kernel.org/c	
5.0	(AV:N/A	2.9	10.0	https://marc.info/?l=li	Issue Tracking; Patch
2.1	(AV:L/A	2.9	3.9	https://patchwork.kei	Issue Tracking; Patch;
10.0	(AV:N/A	10.0	10.0	https://groups.google	Issue Tracking; Mailing
4.6	(AV:L/A	6.4	3.9	http://git.kernel.org/c	Patch; Third Party Advi

图 4-2 Linux 内核 CVE 信息表部分截图（右半部分）

为了获取 CVE 对应的补丁代码，需要确定存在补丁代码的网站。通过 CVE 参考链接统计工具，将参考链接的网站主站点 URL 进行计数（图 4-3），并逐一验证是否可以查阅到补丁代码。最后发现数量足够多且具有补丁代码的网站为 git.kernel.org（1158 次）、github.com（783 次）、patchwork.kernel.org（31 次）。

Hyperlink	Count
http://www.ubuntu.co	2166
http://lists.opensuse.c	1785
http://www.securityfo	1455
http://git.kernel.org	1158
http://www.openwall.(	1019
https://bugzilla.redhat	882
http://www.debian.or(	836
https://github.com	783
http://www.kernel.org	770
http://www.redhat.co(	732
http://rhn.redhat.com	700
http://www.mandriva.	510

图 4-3 CVE 参考链接的网站主站点 URL 的计数的部分截图

在确定了补丁代码网站之后，通过补丁下载工具，将每一个 CVE 的补丁信息各自保存在以 CVE 编号为名的文件夹中（图 4-4），总数共计 1131 个。

每一个 CVE 中的补丁等文件的结构如图 4-5 所示，Source.txt 保存补丁来源，以漏洞文件路径命名的文件是补丁代码文件（其中路径分隔符被替换成“~!@#”），“(AM)”开头的文件是漏洞修改后代码文件，“(BM)”开头的文件是漏洞修改前代码文件。

每一个补丁代码文件中往往由若干个 Diff 段，每一个 Diff 段如图 4-6 所示（图 4-6 所示的补丁代码文件恰巧只有一个 Diff 段），其中包括 Diff 头和补丁代码。

根据补丁来源进行统计得到表 4-1，可以看到具有 BM 文件（漏洞修改前代码文件）的 CVE 占到了 99%，具有 AM 文件（漏洞修改后代码文件）的 CVE 占到了 95%。

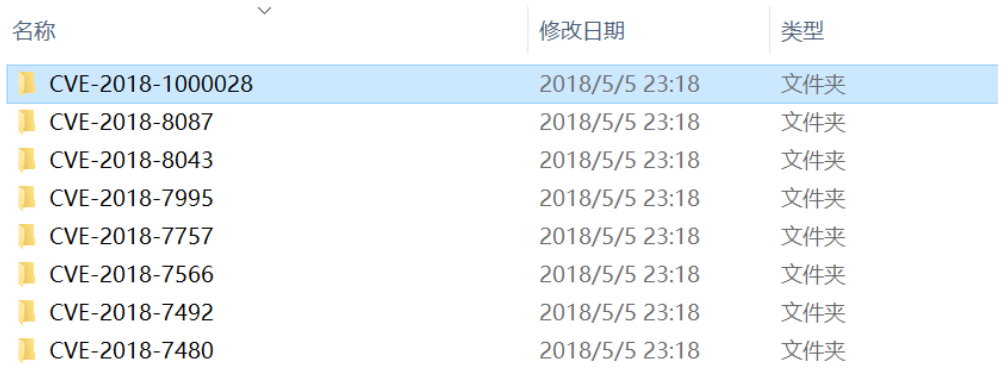

图 4-4 Linux 内核补丁目录部分截图

名称	修改日期	类型
(AM)net~!@#llc~!@#llc_conn.c	2018/4/3 17:44	C 文件
(AM)net~!@#llc~!@#llc_sap.c	2018/4/3 17:44	C 文件
(BM)net~!@#llc~!@#llc_conn.c	2018/4/3 17:43	C 文件
(BM)net~!@#llc~!@#llc_sap.c	2018/4/3 17:44	C 文件
net~!@#llc~!@#llc_conn.c	2018/4/3 17:44	C 文件
net~!@#llc~!@#llc_sap.c	2018/4/3 17:44	C 文件
Source.txt	2018/4/3 17:44	文本文档

图 4-5 CVE-2017-6345 目录结构截图

```
diff --git a/net/llc/llc_sap.c b/net/llc/llc_sap.c
index d0e1e80..5404d0d 100644
--- a/net/llc/llc_sap.c
+++ b/net/llc/llc_sap.c
@@ -290,7 +290,10 @@ static void llc_sap_rcv(struct llc_sap *sap, struct sk_buff *skb,

         ev->type   = LLC_SAP_EV_TYPE_PDU;
         ev->reason = 0;
+        skb_orphan(skb);
+        sock_hold(sk);
         skb->sk = sk;
+        skb->destructor = sock_efree;
         llc_sap_state_process(sap, skb);
 }
```

图 4-6 补丁代码文件 net~!@#llc~!@#llc_sap.c 代码截图

表 4-1 补丁来源表

网站	数量	百分比
git.kernel.org	1073	95%
github.com	43	4%
patchwork.kernel.org	15	1%

在第三章中我们确定了本文重点考虑的那些特征因子，通过特征因子爬虫工具，得到 Linux 所有 CVE 的特征因子（图 4-7，图 4-8），其中 NVD 发布的受漏洞影响的版本存在错误，因此需要有漏洞检测工具进一步纠正。

CVE Number	Version Number	Exist Time	Discover Time	Interval(Month)	Vulnerability Type
CVE-2018-7995	1.2.0	1995年03月	2018年03月	276	Race Conditions (CV
CVE-2018-7757	1.2.0	1995年03月	2018年01月	274	Resource Managem
CVE-2018-7755	1.2.0	1995年03月	2018年03月	276	Information Leak / D
CVE-2018-7740	1.2.0	1995年03月	2018年03月	276	Buffer Errors (CWE-

图 4-7 Linux 内核 CVE 特征因子部分截图（左半部分）

CVSS Severity V2

Base Score	Attack Vector	Access Complexity	Authentication	Confidentiality	Integrity	Availability
4.7;MEDIUM	Local	Medium	None	None	None	Complete
2.1;LOW	Local	Low	None	None	None	Partial
5.0;MEDIUM	Network	Low	None	Partial	None	None
4.9;MEDIUM	Local	Low	None	None	None	Complete

图 4-8 Linux 内核 CVE 特征因子部分截图（右半部分）

4.2 漏洞数据纠正

本小节是针对前一节获取数据的纠正，并展示纠正后的数据结果。

在展示纠正后的特征因子数据之前，我们有必要展示一下代码定位工具的成果，让读者能更好地理解漏洞检测工具的成果。

通过代码定位工具，可以将 BM 文件（漏洞修改前文件，图 4-9）中的模块信息提炼出来得到 Module-BM 文件（存储 BM 文件的模块的关键信息的文件，图 4-10），其中包扩了模块是结构体还是函数、返回值（函数具有）、模块名、起始行号、结束行号。

对于补丁（图 4-11）的定位如图 4-12，在 Module-Diff 文件中（存储 Diff 段所在模块的关键信息的文件）可以看到每一个 Diff 段所在的模块关键信息，每一行代表一个 Diff 段所在的模块信息，若该行为空，则说明这段代码处于全局中。通过 Moddle-Diff 文件来解决补丁代码中 Diff 头存在错误的问题。

表 4-2 显示了 C 语言文件占据了 Linux 内核所有文件的 77%，因此该工具是有意义的。

```
#include <crypto/internal/aead.h>
#include <crypto/internal/hash.h>
#include <crypto/internal/skcipher.h>
#include <crypto/scatterwalk.h>
#include <linux/err.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>

#include "internal.h"

struct ccm_instance_ctx {
        struct crypto_skcipher_spawn ctr;
        struct crypto_ahash_spawn mac;
};

struct crypto_ccm_ctx {
        struct crypto_ahash *mac;
        struct crypto_skcipher *ctr;
};
```

图 4-9 漏洞修改前代码文件的代码截图

```
struct  ccm_instance_ctx        24      27
struct  crypto_ccm_ctx  29      32
struct  crypto_rfc4309_ctx      34      37
struct  crypto_rfc4309_req_ctx  39      43
struct  crypto_ccm_req_priv_ctx 45      52
struct  cbcmac_tfm_ctx  54      56
struct  cbcmac_desc_ctx 58      60
function        crypto_ccm_req_priv_ctx crypto_ccm_reqctx       62      68
function        int     set_msg_len     70      86
function        int     crypto_ccm_setkey       88      114
function        int     crypto_ccm_setauthsize  116     133
function        int     format_input    135     155
function        int     format_adata    157     174
```

图 4-10 Module-BM 文件内容截图

```
diff --git a/crypto/ccm.c b/crypto/ccm.c
index 4428488..1ce37ae 100644
--- a/crypto/ccm.c
+++ b/crypto/ccm.c
@@ -45,6 +45,7 @@ struct crypto_rfc4309_req_ctx {

 struct crypto_ccm_req_priv_ctx {
        u8 odata[16];
+       u8 idata[16];
        u8 auth_tag[16];
        u32 flags;
        struct scatterlist src[3];
@@ -183,8 +184,8 @@ static int crypto_ccm_auth(struct aead_request *req, struc
        AHASH_REQUEST_ON_STACK(ahreq, ctx->mac);
        unsigned int assoclen = req->assoclen;
        struct scatterlist sg[3];
-       u8 odata[16];
-       u8 idata[16];
+       u8 *odata = pctx->odata;
+       u8 *idata = pctx->idata;
        int ilen, err;

        /* format control data for input */
```

图 4-11 补丁文件中的 Diff 段截图

```
struct  crypto_ccm_req_priv_ctx 45      52
function        int     crypto_ccm_auth 176     233
```

图 4-12 Module-Diff 文件内容截图

表 4-2 Linux 内核文件类型分布表（Linux 内核 4.0.1 版本）

文件类型	数量	百分比
无后缀	4799	10%
.c	20981	43%
.S	1386	3%
.h	16555	34%
.dtsi	621	1%
.dts	791	2%
.txt	2698	5%
其他	1126	2%
总计	48957	100%

现在便可展示纠正后的特征因子数据（即漏洞检测工具的成果，图 4-13），由于部分 CVE 没有补丁代码，因此无法使用漏洞检测工具，由于本次实验采用的 Linux 内核为 80 个左右的 Linux 内核正式版本，有些漏洞存在于 Linux 内核的某些测试版本，因此这一部分在本次试验中检测不到 Linux 版本（对于这一点，只需要解压更多的 Linux 内核用于漏洞检测工具即可，只是漏洞检测工具所花费的时间将大幅增加，本人下载下来的 Linux 内核版本数量超过 2000）。最终得到 800 条包括漏洞最早存在版本在内所有信息均较为准确的 CVE 信息（不包含完整数据的 CVE 项需予以删除）。

CVE Number	Linux Kernel Version Number	Vulnerability Exist Time
CVE-2018-7995	3.3	2012年03月
CVE-2018-7757	2.6.19	2006年11月
CVE-2018-7755		
CVE-2018-7740		
CVE-2018-7492	2.6.30	2009年06月
CVE-2018-7480	4.3	2015年11月
CVE-2018-7273		
CVE-2018-6927	2.5.70	2003年05月
CVE-2018-6412	3.15	2014年06月
CVE-2018-5750	2.6.24	2008年01月

图 4-13 修正后的受漏洞影响的最早版本及其发布时间部分截图

4.3 漏洞态势感知

本小节主要讲述根据第三章中漏洞态势感知模型得出的结果。

在获取了 Linux 内核漏洞的基本历史数据后，对其进行分析统计，便可以得到 Linux 内核漏洞态势的感知结果。

首先，我们可以根据不同等级的漏洞数量走势绘制漏洞危害等级走势图（图 4-14），可以发现最下面的高危害漏洞的发现数量随着年份增加而波动上升，而中危害漏洞的发

现数量先增加后趋于稳定，在 37%左右，低危害漏洞的发现数量比例较低且数量非常稳定，在 10%左右。

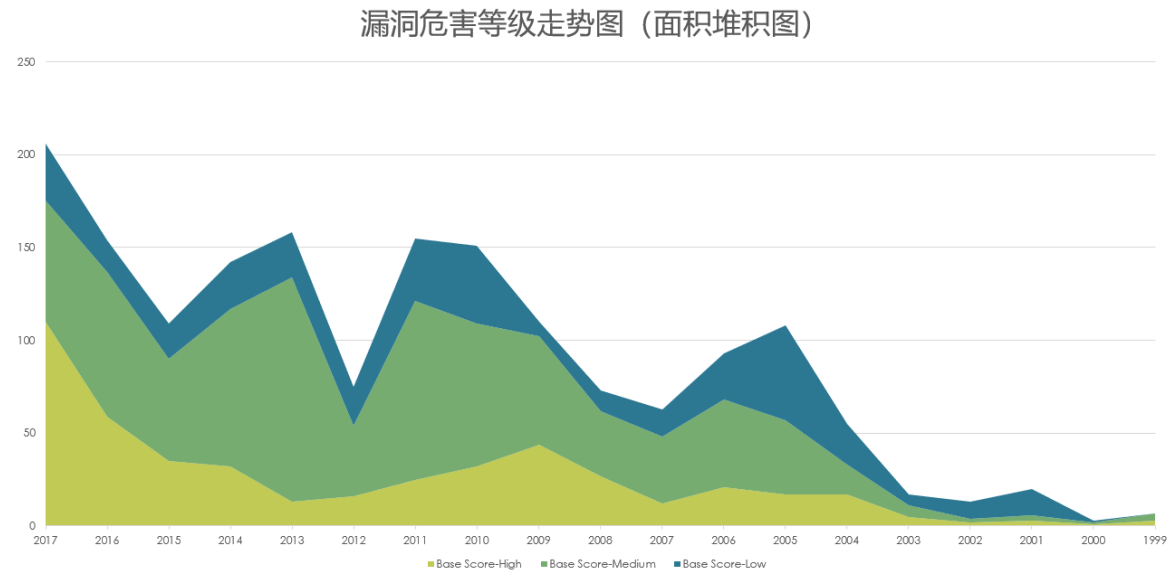

图 4-14 漏洞危害等级走势图

其次，我们可以得出不同标准下漏洞类型的分布情况，如图 4-15、图 4-16、图 4-17、图 4-18。我们可以发现并得出 CWE-264、CWE-362、CWE-189、CWE-200、CWE-20、CWE-119、CWE-399、CWE-416 为 Linux 内核历史中最为关键的漏洞类型（本毕业设计中，是将四个标准下的关键漏洞类型都作为需要考虑的关键漏洞类型。），当然随着时间的流逝，不同的版本的最关键漏洞类型会存在一些差别（最新发布的版本的 Linux 内核并无任何数据可以查询），在对本毕业设计具体使用过程中，可以结合实际经验在此基础上确定当前需要考虑的 Linux 内核关键漏洞类型，这将会起到更佳的作用。在本次实验中，以高危害程度的 CWE-264、CWE-362、CWE-189、CWE-200、CWE-20、CWE-119、CWE-399、CWE-416 作为关键漏洞，上述类型的漏洞数量分别为 203，107，164，161，170，85，204，26 个（共 1120 个），对应权重比例为 18%，10%，15%，14%，15%，8%，18%，2%。

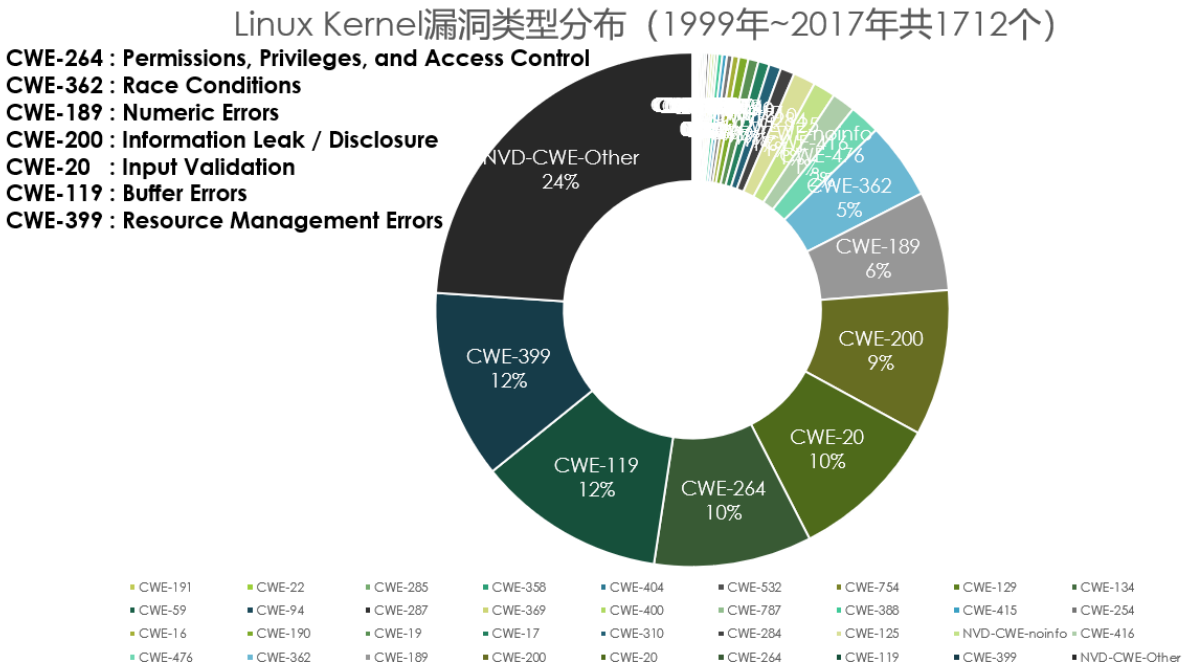

图 4-15 Linux 内核漏洞类型分布图

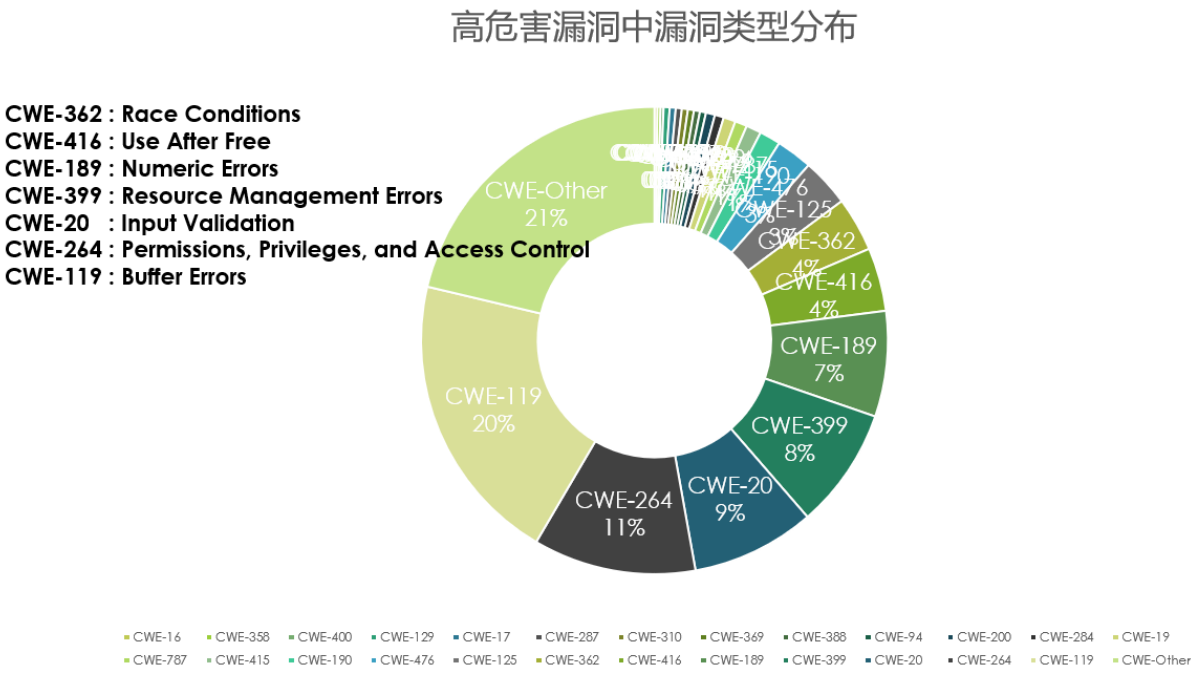

图 4-16 Linux 内核的高危害漏洞中的漏洞类型数量分布图

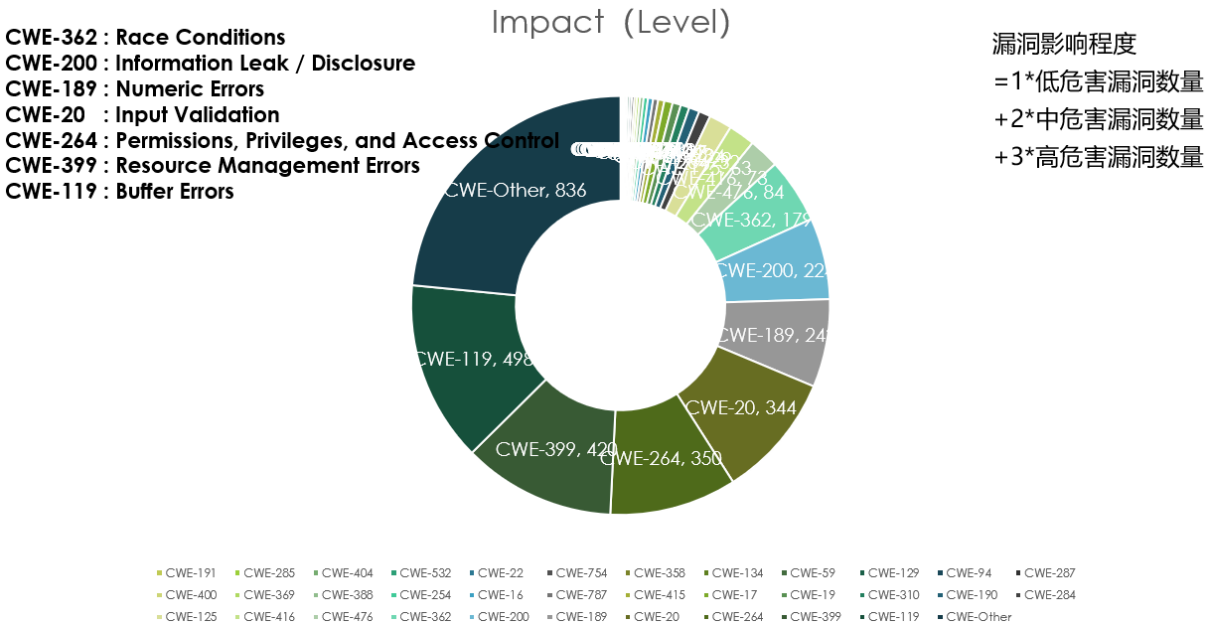

图 4-17 Linux 内核的漏洞影响程度分布图（标准一）

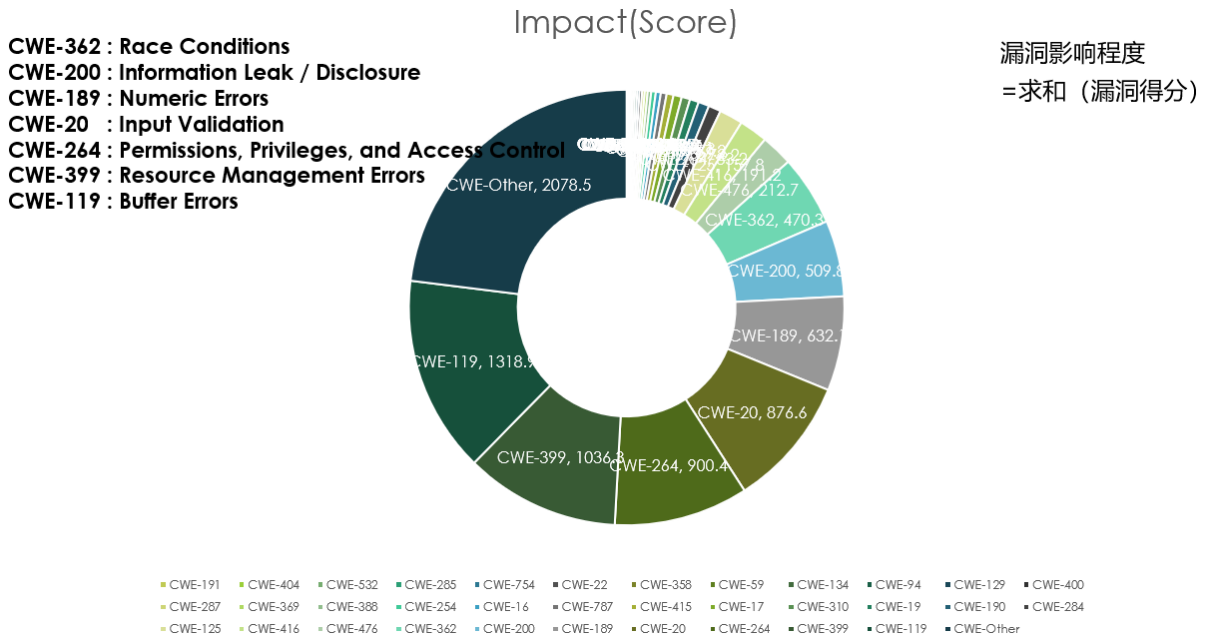

图 4-18 Linux 内核的漏洞影响程度分布图（标准二）

4.4 漏洞态势预测

我们已经获得了 800 条左右的较为准确的可训练数据，因此可以进行模型的训练。在此，本人使用了 Pybrain 工具和自己编写的 BP 神经网络工具。值得一提的是，Linux 内核的最早存在到被发现的时间间隔取值范围为 0 年-25 年（在实验中，为了表达为独热输出，月数的范围表示为 0 月-300 月）。

关于 BP 神经网络的输入，需要利用“特征因子标准化工具”对本次实验所用到的特征因子进行提取和标准化，其中 CWE 编号表示为独热模式，因为 CWE 编号为离散非连续数据。值得注意的是本次实验中 Interval 采用的月份数，使用的漏洞危害程度是 0 对应低危害，0.5 对应中危害，1 对应高危害（关于 CVSS 评分目前暂未采用）。将 CWE 编号的独热表示和漏洞危害程度作为 BP 神经网络的输入，Interval 作为神经网络的参考输出。

Pybrain 工具（BP 神经网络结构为 125-25-300，即包括输入层在内共三层的神经网络，输入层结点 125 个，隐含层结点 25 个，输出层结点 300 个）效果如图 4-19、图 4-20。测试样本测试的误差一年内的准确率为 43%，误差三年内的准确率为 64%，误差五年内的准确率 76%；回归测试的误差一年内的准确率为 32%，误差三年内的准确率为 58%，误差五年内的准确率 72%。

本人编写的工具（BP 神经网络结构为 125-25-300，即包括输入层在内共三层的神经网络，输入层结点 125 个，隐含层结点 25 个，输出层结点 300 个）效果如图 4-21、图 4-22。测试样本测试的误差一年内的准确率为 42%，误差三年内的准确率为 69%，误差五年内的准确率 77%；回归测试的误差一年内的准确率为 37%，误差三年内的准确率为 63%，误差五年内的准确率 75%。

值得注意的是，由于可能存在个例数据的异常情况，会对神经网络的效果起到一定程度的影响。此外，由于之前提到的 Linux 内核版本采用的数量不到 100，很多测试版本的 Linux 内核在本次实验中没有被考虑在内，因此可能出现漏洞最早存在版本不够精确的可能。再者，神经网络的结构层次和节点数需要通过实验来摸索出最佳的结构，125-25-300 可能不是本实验最佳的神经网络结构。综上原因都会对实验效果产生影响，这些可以在未来的研究中进行探讨。

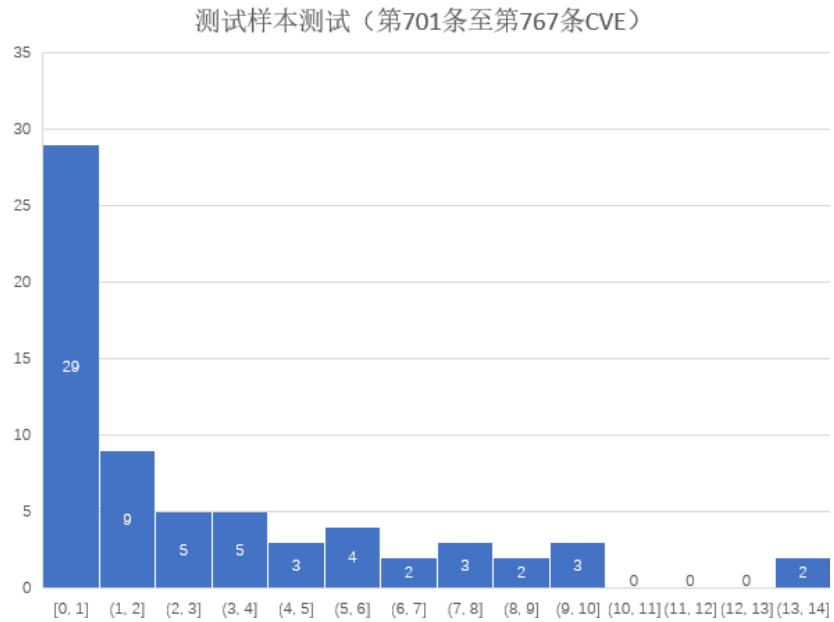

图 4-19 测试样本的测试结果误差分布图（Pybrain 库）

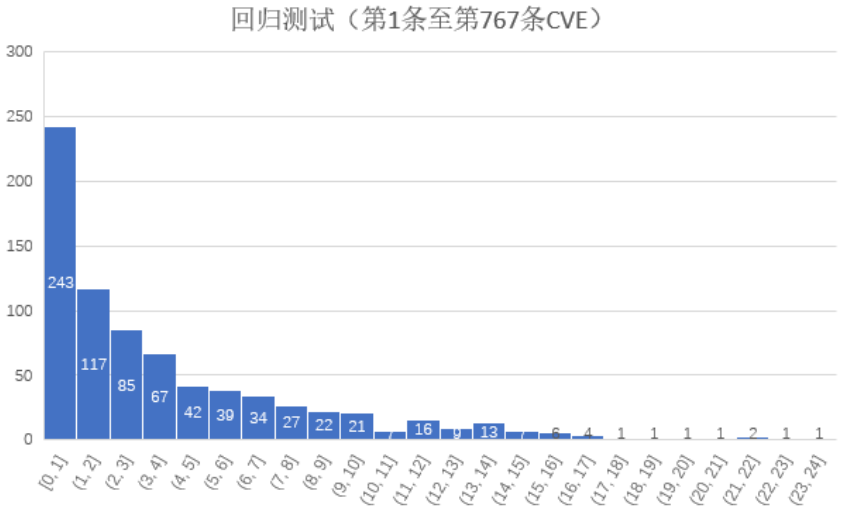

图 4-20 回归测试的测试结果误差分布图（Pybrian 库）

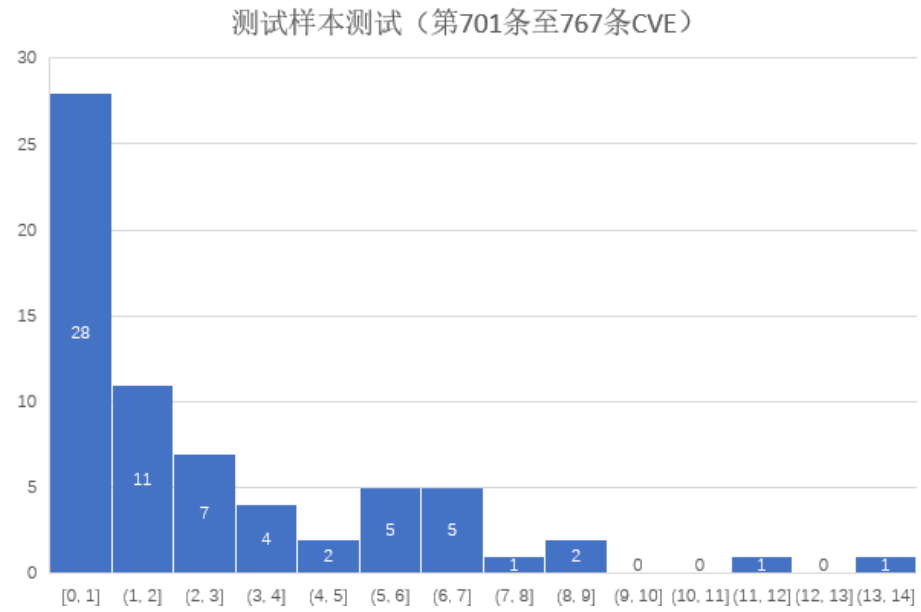

图 4-21 测试样本测试结果误差分布图（自己的而非 Pybrain 库的）

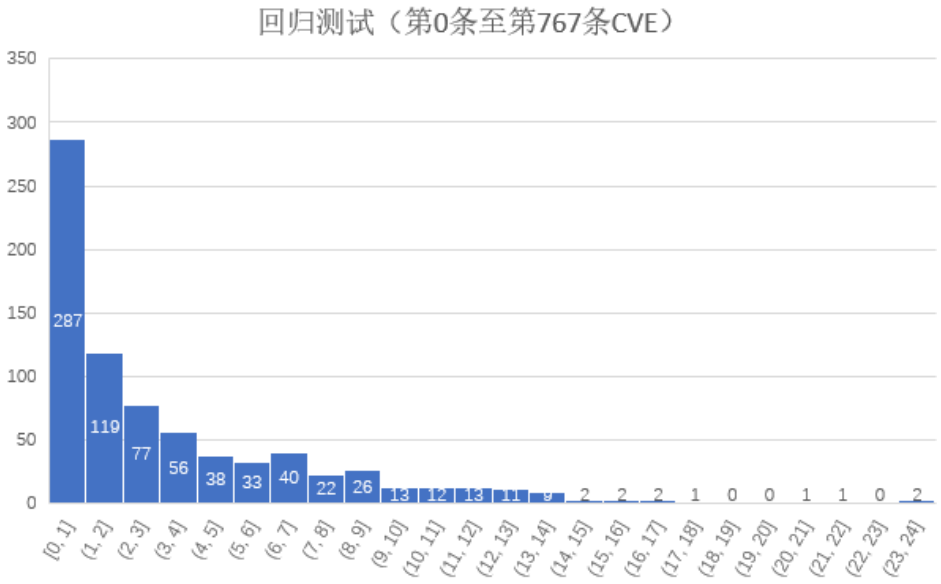

图 4-22 回归测试的测试结果误差分布图（自己的而非 Pybrain 库的）

根据神经网络训练和测试的结果，本人认为准确度达到了基本的期望，可以用于预测输出。根据 Linux 内核漏洞的感知结果，我们将预测的输入设置为 CWE-119、CWE-189、CWE-20、CWE-200、CWE-264、CWE-362、CWE-399、CWE-416 的独热表示和高危害程度（值表示为 1，表示最高的危害程度），得到预测的结果分别为 1 个月、26 个月、6 个月、266 个月、4 个月、57 个月、20 个月、18 个月。如果采用平均的方法则结果为 49.75 个月，即 4.14 年。如果采用加权平均的方法（上述类型漏洞总数量的比例分别为 18%，10%，15%，14%，15%，8%，18%，2%，以此为权重），则结果为 50.04 个月，即 4.17 年

当本人把高危害程度定义为值等于 0.7、0.8、0.9、1（其中 1 表示最高的危害程度）时，得到的预测结果如表 4-3。预测结果的平均值为 4.1875 年（对“4 个高危害程度值*8 个漏洞类型=32 个预测值”求平均）。如果先对每一个高危害程度值的对应结果（8 个漏洞类型的结果）求加权平均，那么不同高危害程度（值为 0.7、0.8、0.9、1.0）的预测结果分别为 4.83 年、4.25 年、4.16 年、4.17 年，求平均为 4.35 年。如果根据不同高危害程度以及漏洞类型进行加权平均（计数见表 4-4），则结果为 22.4 个月，即 1.87 年（相对于 4 年有明显差距，原因是时间间隔长的漏洞数量较少，时间间隔短的漏洞数量较多）。

在输入设计的过程中，可以根据实际情况来优化，那么得出的结果将更可靠。在输出的选择、分析过程中，可以根据实际情况采用更佳的权重来进行加权平均。

在本毕业设计中，在抛开 Linux 内核各版本之间的差异的情况下，Linux 内核需要 2 到 4 年左右的时间来实现稳定化过程，即 2 到 4 年为 Linux 稳定化过程的所需时长的参考。

表 4-3 Linux 内核关键漏洞类型的不同高危害程度的漏洞的最早存在到被发现时间间隔预测表

Score(Normalization)	CWE-119	CWE-189	CWE-20	CWE-200	CWE-264	CWE-362	CWE-399	CWE-416
1	1	26	6	266	4	57	20	18
0.9	1	26	6	266	23	25	20	0
0.8	1	36	6	266	23	25	20	0
0.7	1	36	36	266	1	25	46	55

表 4-4 Linux 内核关键漏洞类型的不同高危害程度数量分布表

Score(Normalization)	CWE-119	CWE-189	CWE-20	CWE-200	CWE-264	CWE-362	CWE-399	CWE-416
1	9	2	3	0	1	0	0	3
0.9	8	1	0	0	8	1	0	2
0.8	3	1	1	0	0	0	0	0
0.7	77	31	37	3	45	17	40	16

4.5 本章小结

本章按毕业设计实现过程中的先后顺序对各项成果进行展示，期望能使读者有一个直观的了解。本章得出了漏洞态势感知的结果并最终得到了 Linux 内核稳定化过程所需时长的基本参考值，可以用于结论的得出。

5 论文结论

本章展示了本毕业设计最终得到的结论和意义，说明本毕业设计的价值。

5.1 结论

本毕业设计实现了爬虫工具、漏洞检测工具、神经网络预测工具、其他辅助工具，以此来得出最终的实验结果，这些工具和结果对于相关研究的人员可以起到很好的参考作用，同时这些工具和内容通过细节部分的修改能够很好的应用于其他软件，并对这些软件的漏洞态势进行感知和对这些软件的稳定化过程时长进行预测。以下是满足第一章提出的假设约束下的漏洞态势感知和预测的结论：

在漏洞态势感知方面。首先，根据图 4-14，我们可以发现高危害漏洞的发现数量随着年份增加而波动上升，而中危害漏洞的发现数量先增加后趋于稳定，在 37%左右，低危害漏洞的发现数量比例较低且数量非常稳定，在 10%左右。其次，根据第四章的结果，我们可以发现 CWE-119、CWE-189、CWE-20、CWE-200、CWE-264、CWE-362、CWE-399、CWE-416 类型的漏洞是 Linux 内核的关键的危害影响程度最大的漏洞。

在漏洞态势预测方面，对于我采用的关键漏洞类型的不同高危害程度的漏洞最早存在到被发现时间间隔的预测如表 4-3。本论文的关键结论为：在满足第一章的假设和约束的条件下，本文认为 Linux 内核的稳定化过程所需要的时长为 2 到 4 年左右。该结论起到的参考意义如下：

- 在基于 Linux 内核定制操作系统时，开发人员可以选取 2 到 4 年前发布的 Linux 内核作为参考。
- 在 Linux 内核测试过程中，2 到 4 年可以作为 Linux 内核关键的测试阶段时长的参考。

5.2 建议

由于本人的时间和水平的约束，本论文还存在许多问题值得进一步探讨。未来的实践过程中可以对本论文中尤其是假设和约束部分进行改进。此外，在第四章实验结果中，对于数据的细节处理可以根据实际情况和反复实验来得出更佳的结论。

5.3 本章小结

本章主要阐述了在特定的前提假设下所得出的 Linux 内核漏洞态势感知结果和稳定化时间的参考值，以及对实际问题的解答和贡献，最后提出了本研究可以改进的方向。

参考文献

[1] Kuhn D R, Raunak M S, Kacker R. An Analysis of Vulnerability Trends, 2008-2016[C]. IEEE International Conference on Software Quality, Reliability and Security Companion. IEEE, 2017:587-588.

[2] Slaby J, Strejček J, Trtík M. ClabureDB: Classified Bug-Reports Database[M]. Verification, Model Checking, and Abstract Interpretation. Springer Berlin Heidelberg, 2013:268-274.

[3] Lee S C, Davis L B. Learning from experience: operating system vulnerability trends[J]. It Professional, 2003, 5(1):17-24.

[4] Grieco G, Grinblat G L, Uzal L, et al. Toward Large-Scale Vulnerability Discovery using Machine Learning[C]. ACM Conference on Data and Application Security and Privacy. ACM, 2016:85-96.

[5] Abal I, Brabrand C, Wasowski A. 42 variability bugs in the linux kernel: a qualitative analysis[C]. Acm/ieee International Conference on Automated Software Engineering. 2014:421-432.

[6] Woo M, Sang K C, Gottlieb S, et al. Scheduling black-box mutational fuzzing[C]. ACM Sigsac Conference on Computer & Communications Security. ACM, 2013:511-522.

[7] Homaei H, Shahriari H R. Seven Years of Software Vulnerabilities: The Ebb and Flow[J]. IEEE Security & Privacy, 2017, 15(1):58-65.

[8] Chang Y Y, Zavarsky P, Ruhl R, et al. Trend Analysis of the CVE for Software Vulnerability Management[C]. IEEE Third International Conference on Privacy, Security, Risk and Trust. IEEE, 2012:1290-1293.

[9] Rick Kuhn, Chris Johnson. Vulnerability Trends: Measuring Progress[J]. It Professional, 2010, 12(4):51-53.

[10] 何晶. 基于 WooYun 的视听新媒体网站漏洞统计分析[J]. 电视技术, 2014, 38(16):65-69.

[11] 佚名. 赛门铁克互联网安全威胁报告[J]. 软件和集成电路, 2003(6):64-64.

[12] Ullah N, Morisio M, Vetrò A. Selecting the Best Reliability Model to Predict Residual Defects in Open Source Software[J]. Computer, 2015, 48(6):50-58.

[13] Okamura H, Dohi T. Towards comprehensive software reliability evaluation in open source software[C]. IEEE, International Symposium on Software Reliability Engineering. IEEE Computer Society, 2015:121-129.

[14] Rahimi S, Zargham M. Vulnerability Scrying Method for Software Vulnerability Discovery Prediction Without a Vulnerability Database[J]. IEEE Transactions on Reliability, 2013, 62(2):395-407.

致　　谢

本论文的工作是在我的指导老师何永忠教授的悉心指导下完成，何永忠老师严谨的治学态度和科学的工作方法给了我极大的帮助和影响，在此由衷地感谢何永忠老师给予的关心和指导。

刘嘉乐学姐和梁迪学长在我的工作过程中给予了热情的帮助，在此向他们表达我的感激之情。

附　　录

附录 A 程序代码

详情请见见毕业设计的附件，此处贴上 22 个 python 文件中的 3 个。

BPNNByEpoch.py

说明：该 Python 代码文件内容为所编写的 BP 神经网络。

代码如下：

```
# Author: 14281055 Liheng Chen CIT BJTU
# File Name: BPNNByEpoch.py

import numpy
import os
import Repository


class BPNeuralNetwork(object):
    def sigmoid(self, x, derivation=False):
        if derivation:
            return self.sigmoid(x) * (1 - self.sigmoid(x))
        else:
            return 1 / (1 + numpy.exp(-x))

    input_data = None
    target_data = None

    layer_input_list = None
    layer_output_list = None

    layer_bias = None

    layer_number = None
    layer_node_number_list = None

    weight_list = []
    correction_matrix_list = []

    layer_output_delta_list = None
    layer_error_list = None

    activation_func = sigmoid

    epochs = 100000000
```

```
goal = 0.00001
learn_rate = 0.08    # usually in (0,0.1); annealing
correct_rate = 0.008    # annealing

loss = None

weight_file_path = None

def __init__(self, layer_node_number_list, input_data, target_data=None, weight_file_path=None):
    self.layer_node_number_list = layer_node_number_list

    self.input_data = input_data
    self.target_data = target_data

    self.weight_file_path = weight_file_path

    self.layer_number = len(self.layer_node_number_list)

    self.layer_bias = [0] * (self.layer_number - 1)

    self.layer_input_list = [numpy.array(None)] * self.layer_number
    self.layer_output_list = [numpy.array(None)] * self.layer_number

    self.layer_output_delta_list = [numpy.array(None)] * self.layer_number
    self.layer_error_list = [numpy.array(None)] * self.layer_number

    for layer_index in range(self.layer_number - 1):
        self.weight_list.append(numpy.random.randn(
            layer_node_number_list[layer_index],
            layer_node_number_list[layer_index + 1]
        ) / numpy.sqrt(self.layer_node_number_list[layer_index]))
        self.correction_matrix_list.append(numpy.zeros(self.weight_list[layer_index].shape))
    if self.weight_file_path is not None and os.path.exists(self.weight_file_path):
        self.weight_list = numpy.load(self.weight_file_path)

def train(self):
    for epoch in range(self.epochs):
        self.loss = 0
```

```
            for index in range(self.input_data.shape[0]):
                self.forward_propagation(
                    numpy.atleast_2d(self.input_data[index])
                )
                self.back_propagation(numpy.atleast_2d(self.target_data[index]))
                self.loss += numpy.square(
                    numpy.atleast_2d(self.target_data[index]) - self.layer_output_list[-1]).sum() / 2
            if self.weight_file_path is not None and epoch % 100 == 0:
                numpy.save(os.path.splitext(self.weight_file_path)[0], self.weight_list)
            self.annealing(epoch)
            print('\rEpoch:%d Loss:%f' % (epoch, self.loss), end='')
            if self.loss < self.goal:
                break

    def test(self):
        self.forward_propagation(self.input_data)
        return self.layer_output_list[-1]

    def forward_propagation(self, input_data):
        self.layer_input_list[0] = input_data
        self.layer_output_list[0] = input_data
        for layer_index in range(1, self.layer_number):
            self.layer_input_list[layer_index] = \
                self.layer_output_list[layer_index - 1].dot(self.weight_list[layer_index - 1]) + \
                self.layer_bias[layer_index - 1]
            self.layer_output_list[layer_index] = self.activation_func(self.layer_input_list[layer_index])

    def back_propagation(self, target_data):
        self.layer_output_delta_list[-1] = target_data - self.layer_output_list[-1]
        self.layer_error_list[-1] = \
            self.layer_output_delta_list[-1] * \
            self.activation_func(
                self.layer_input_list[-1],
                derivation=True
            )
        self.weight_list[-1] += self.learn_rate * self.layer_output_list[-2].T.dot(
            self.layer_error_list[-1]) + self.correct_rate * self.correction_matrix_list[-1]
        self.correction_matrix_list[-1] = self.layer_output_list[-2].T.dot(self.layer_error_list[-1])
```

```
        for layer_index in range(-2, -self.layer_number, -1):
            self.layer_output_delta_list[layer_index] = \
                self.layer_error_list[layer_index + 1].dot(self.weight_list[layer_index + 1].T)
            self.layer_error_list[layer_index] = \
                self.layer_output_delta_list[layer_index] * \
                self.activation_func(
                    self.layer_input_list[layer_index],
                    derivation=True
                )
            self.weight_list[layer_index] += \
                self.learn_rate * self.layer_output_list[layer_index - 1].T \
                    .dot(self.layer_error_list[layer_index]) \
                + self.correct_rate * self.correction_matrix_list[layer_index]
            self.correction_matrix_list[layer_index] = self.layer_output_list[layer_index - 1].T \
                .dot(self.layer_error_list[layer_index])

    def annealing(self, epoch, step=100000):
        if epoch % step == 0:
            self.learn_rate /= 2    # learn rate annealing
            self.correct_rate /= 2


def train(layer_node_number_list, train_input_file_path, train_target_file_path, weight_file_path=None,
          target_one_hot_flag=False):
    if target_one_hot_flag:
        train_target_one_hot_file_path = Repository.add_suffix_to_file_name(train_target_file_path,
                                                                            '_one_hot')
        if not os.path.exists(train_target_one_hot_file_path):
            Repository.save_one_hot(
                train_target_file_path,
                layer_node_number_list[-1],
                train_target_one_hot_file_path
            )
        train_target_file_path = train_target_one_hot_file_path
    bp_neural_network = BPNeuralNetwork(
        layer_node_number_list,
        Repository.excel_to_np_array(train_input_file_path),
        Repository.excel_to_np_array(train_target_file_path),
```

```
        weight_file_path
    )
    bp_neural_network.train()


def test(layer_node_number_list, test_input_file_path, test_predict_file_path, weight_file_path,
         target_one_hot_flag=False):
    bp_neural_network = BPNeuralNetwork(
        layer_node_number_list,
        Repository.excel_to_np_array(test_input_file_path),
        weight_file_path=weight_file_path
    )
    if target_one_hot_flag:
        test_predict_one_hot_file_path = Repository.add_suffix_to_file_name(test_predict_file_path,
                                                                            '_one_hot')
        Repository.np_array_to_excel(bp_neural_network.test(), test_predict_one_hot_file_path)
        Repository.save_one_hot_like_reversal(test_predict_one_hot_file_path, test_predict_file_path)
    else:
        Repository.np_array_to_excel(bp_neural_network.test(), test_predict_file_path)


if __name__ == '__main__':
    train(
        [125, 25, 300],
        r'C:\Users\陈力恒\Downloads\VulnerabilitySituation\Backup\Data\train_input_normal_l.xlsx',
        r'C:\Users\陈力恒\Downloads\VulnerabilitySituation\Backup\Data\train_target.xlsx',
        r'C:\Users\陈力恒\Downloads\VulnerabilitySituation\Backup\Backup3(My)\weight125_25_300.npy',
        target_one_hot_flag=True
    )
    test(
        [125, 25, 300],
        # r'C:\Users\陈力恒\Downloads\VulnerabilitySituation\Backup\Data\test_input_normal_l.xlsx',
        r'C:\Users\陈力恒\Downloads\VulnerabilitySituation\Backup\Data\predict_input_normal.xlsx',
        # r'C:\Users\陈力恒\Downloads\VulnerabilitySituation\Backup\Backup3(My)\test_predict.xlsx',
        r'C:\Users\陈力恒\Downloads\VulnerabilitySituation\Backup\Backup3(My)\predict_output.xlsx',
        r'C:\Users\陈力恒\Downloads\VulnerabilitySituation\Backup\Backup3(My)\weight125_25_300.npy',
```

```
        target_one_hot_flag=True
    )
```

GetModuleDiffIn.py

说明：该 Python 代码文件内容主要为代码定位工具。

代码如下：

```
# Author: 14281055 Liheng Chen CIT BJTU
# File Name: GetModuleDiffIn.py
import Repository
import re
import os


def match_line(module_info_list, line):
    line = Repository.separate_word_and_nonword(line)
    if module_info_list[0] == 'struct':
        return True if re.search(r'\bstruct ' + re.escape(module_info_list[1]) + ' {', line) else False
    elif module_info_list[0] == 'function':
        return True if re.search(
            r'\b' + re.escape(module_info_list[1]) + ' ' + re.escape(module_info_list[2]) + r' \(',
            line) else False
    return None


def match_lines(module_info_list, line_list, default_line_index_range=30):
    if match_line(module_info_list, line_list[0]):
        return True
    line_index_range = default_line_index_range if len(line_list) > default_line_index_range else len(
        line_list)
    lines = Repository.line_list_to_lines(line_list[:line_index_range])
    first_line_end_index = Repository.get_first_line_end_index_in_lines(line_list[0], lines)
    if module_info_list[0] == 'struct':
        match = re.search(r'\bstruct ' + re.escape(module_info_list[1]) + ' {', lines)
        if match:
            tag_end_index = match.start() + 5    # length of 'struct' is 6
            if tag_end_index <= first_line_end_index:
                return True
        return False
    elif module_info_list[0] == 'function':
        match = re.search(
            r'\b' + re.escape(module_info_list[1]) + ' (?:\* )*' + re.escape(
```

```
                module_info_list[2]) + r' \([^)]*\) {',
            lines)
        if match:
            tag_end_index = match.start() + len(module_info_list[1]) - 1
            if tag_end_index <= first_line_end_index:
                return True
        return False
    return None


def get_module_info_list(line_list):
    lines = Repository.line_list_to_lines(line_list)
    first_line_end_index = Repository.get_first_line_end_index_in_lines(line_list[0], lines)
    match = re.search(r'\bstruct (\w+) {', lines)
    if match:
        tag_end_index = match.start() + 5    # length of 'struct' is 6
        if tag_end_index <= first_line_end_index:
            return ['struct', match.group(1)]

    match = re.search(r'(?!if|while)(\w+) (?:\* )*(?!if|while)(\w+) \([^)]*\) {', lines)
    if match:
        tag_end_index = match.start() + len(match.group(1)) - 1
        if tag_end_index <= first_line_end_index:
            return ['function', match.group(1), match.group(2)]
    return None


def read_module_and_scope(module_file_path, line_index):
    line = Repository.get_file_line_list(module_file_path)[line_index]
    if line.strip() == '':
        return []
    else:
        module_and_scope = line.strip().split('\t')
        module_and_scope[-2] = int(module_and_scope[-2])
        module_and_scope[-1] = int(module_and_scope[-1])
        return module_and_scope
```

```
def read_module_and_scope_list(module_file_path):
    module_file_line_list = Repository.get_file_line_list(module_file_path)
    module_and_scope_list = []
    if module_file_line_list:
        for line in module_file_line_list:
            module_and_scope = line.strip().split('\t')
            module_and_scope[-2] = int(module_and_scope[-2])
            module_and_scope[-1] = int(module_and_scope[-1])
            module_and_scope_list.append(module_and_scope)
    return module_and_scope_list


# def get_module_info_list_list(dst_file_path, default_line_list_range=30):
#     dst_file_line_list = Repository.get_file_line_list(dst_file_path)
#     module_info_list_list = []
#     for line_index, line in enumerate(dst_file_line_list):
#         line_list_range = default_line_list_range if len(
#             dst_file_line_list) - line_index > default_line_list_range else len(
#             dst_file_line_list) - line_index
#         module = get_module_info_list(dst_file_line_list[line_index:line_index + line_list_range])
#         if module:
#             module_info_list_list.append(module)
#     return module_info_list_list


def get_module_and_scope_list(dst_file_path, default_line_list_range=30, write_flag=False):
    module_file_path = Repository.add_prefix_to_file_name(dst_file_path, '(Module)')
    is_module_file_exist = True if os.path.exists(module_file_path) else False
    dst_file_line_list = Repository.get_file_line_list(dst_file_path)
    module_and_scope_list = []
    for line_index, line in enumerate(dst_file_line_list):
        line_list_range = default_line_list_range if len(
            dst_file_line_list) - line_index > default_line_list_range else len(
            dst_file_line_list) - line_index
        module = get_module_info_list(dst_file_line_list[line_index:line_index + line_list_range])
        if module:
            scope = get_module_scope(module, dst_file_path, line_index)
            if not scope:
```

```
                scope = [None, None]
            module_and_scope = module + scope
            module_and_scope_list.append(module_and_scope)
            if write_flag and not is_module_file_exist:
                line = '\t'.join(module_and_scope[:-2]) + '\t' + str(module_and_scope[-2]) + '\t' + str(
                    module_and_scope[-1])
                Repository.append_file_with_eol(module_file_path, line)
    return module_and_scope_list


def print_module_and_scope(file_path):
    for module_and_scope in get_module_and_scope_list(file_path):
        print('\t'.join(module_and_scope[:-2]) + '\t' + str(module_and_scope[-2]) + '\t' + str(
            module_and_scope[-1]))


def write_module_and_scope(module_file_path, module_and_scope=None, flag=False):
    if flag:
        line = '\t'.join(module_and_scope[:-2]) + '\t' + str(module_and_scope[-2]) + '\t' + str(
            module_and_scope[-1])
        Repository.append_file_with_eol(module_file_path, line)
    else:
        Repository.append_file_with_eol(module_file_path, '')


def write_module_file(dst_file_path):
    module_file_path = Repository.add_prefix_to_file_name(dst_file_path, '(Module)')
    if not os.path.exists(module_file_path):
        get_module_and_scope_list(dst_file_path, write_flag=True)


def write_module_diff_file(diff_file_path):
    module_diff_file_path = Repository.add_prefix_to_file_name(diff_file_path, '(Module)')
    if not os.path.exists(module_diff_file_path):
        bm_file_path = Repository.add_prefix_to_file_name(diff_file_path, '(BM)')
        write_module_file(bm_file_path)
        for diff_segment in get_diff_segment_list(diff_file_path):
            for module_and_scope in read_module_and_scope_list(
```

```
                    Repository.add_prefix_to_file_name(bm_file_path, '(Module)')):
                if module_and_scope[-2] != 'None' and module_and_scope[-1] != 'None' \
                        and Repository.is_lines_in_lines(diff_segment[4],

Repository.get_file_line_list(bm_file_path)
                                                                [module_and_scope[-2]:
module_and_scope[-1] + 1]):
                    write_module_and_scope(module_diff_file_path, module_and_scope, True)
                    break
            else:
                write_module_and_scope(module_diff_file_path)


def is_diff_segment_in_lines(diff_segment, dst_line_list, has_module_info=True):
    if not has_module_info and diff_segment[1] == []:
        return Repository.unknown
    else:
        return Repository.true if Repository.is_lines_in_lines(diff_segment[1], dst_line_list) \
                                  and not Repository.is_lines_in_lines(diff_segment[2],
                                                                       dst_line_list)  else
Repository.false


def get_module_scope(module, dst_file_path, line_offset=0):
    dst_file_line_list = Repository.get_file_line_list(dst_file_path)
    for line_index in range(len(dst_file_line_list[line_offset:])):
        if match_lines(module, dst_file_line_list[line_offset + line_index:]):
            start_line_index = line_offset + line_index
            break
    else:
        return []
    remain_lines = Repository.line_list_to_lines(dst_file_line_list[start_line_index:])
    remain_line_end_char_index_dict = Repository.get_line_end_char_index_dict_with_lines(
        dst_file_line_list[start_line_index:],
        remain_lines)
    opening_brace_number = 0
    closing_brace_number = 0
    match = re.search(re.escape(module[-1]) + '.*?{', remain_lines)
```

```
    if match:
        # opening_brace_line_index = Repository.get_line_index(remain_line_end_char_index_dict,
        #                                                       match.end() - 1) +
start_line_index
        opening_brace_number += 1
    else:
        return []
    closing_brace_line_index = len(dst_file_line_list)
    for char_index, character in enumerate(remain_lines[match.end():]):
        if character == '{':
            opening_brace_number += 1
        elif character == '}':
            closing_brace_number += 1
            if opening_brace_number == closing_brace_number:
                closing_brace_line_index = Repository.get_line_index(
                    remain_line_end_char_index_dict,
                    char_index + match.end()
                ) + start_line_index
                break
    return [start_line_index, closing_brace_line_index]


def get_diff_segment_list(diff_file_path):
    diff_file_line_list = Repository.get_file_line_list(diff_file_path)
    diff_segment_list = []
    diff_segment = []
    diff_segment_delete_list = []
    diff_segment_add_list = []
    diff_segment_other_list = []
    diff_segment_non_add_list = []
    has_diff_segment_head = False
    for diff_file_line in diff_file_line_list:
        match = re.search(r'@@.*?@@(.*)', diff_file_line)
        if match:
            if diff_segment:
                diff_segment.append(diff_segment_delete_list)
                diff_segment.append(diff_segment_add_list)
                diff_segment.append(diff_segment_other_list)
```

```
                    diff_segment.append(diff_segment_non_add_list)
                    diff_segment_list.append(diff_segment)
                    diff_segment_delete_list = []
                    diff_segment_add_list = []
                    diff_segment_other_list = []
                    diff_segment_non_add_list = []
                    diff_segment = []
                diff_segment_head = match.group(1).strip()
                diff_segment.append(diff_segment_head)
                has_diff_segment_head = True
            else:
                if has_diff_segment_head and diff_file_line.strip() != '':
                    match = re.match(r'\+(.*)', diff_file_line)
                    if match:
                        diff_segment_add_list.append(match.group(1).strip())
                    elif re.match(r'[^-+]', diff_file_line):
                        diff_segment_other_list.append(diff_file_line.strip())
                        diff_segment_non_add_list.append(diff_file_line.strip())
                    else:
                        match = re.match(r'-(.*)', diff_file_line)
                        if match:
                            diff_segment_delete_list.append(match.group(1).strip())
                            diff_segment_non_add_list.append(match.group(1).strip())
        if diff_segment:
            diff_segment.append(diff_segment_delete_list)
            diff_segment.append(diff_segment_add_list)
            diff_segment.append(diff_segment_other_list)
            diff_segment.append(diff_segment_non_add_list)
            diff_segment_list.append(diff_segment)
        return diff_segment_list

# def write_diff_module_and_scope_tuple(diff_file_path):
#                     bm_file_path = os.path.join(os.path.dirname(diff_file_path), '(BM)' +
os.path.basename(diff_file_path))
#                     save_path = os.path.join(os.path.dirname(diff_file_path), '(Module)' +
os.path.basename(diff_file_path))
#         if os.path.exists(save_path):
#             os.remove(save_path)
```

```
#         bm_file_line_list = Repository.get_file_line_list(bm_file_path)
#         for diff in get_diff_list(diff_file_path):
#             diff_module_and_scope_tuple = get_diff_module_and_scope_tuple(diff[1:], bm_file_line_list)
#             string = '\t'.join(diff_module_and_scope_tuple[0])
#             string += '\t'
#             string += '\t'.join(
#                 [str(diff_module_and_scope_tuple[1]), str(diff_module_and_scope_tuple[2])]
#             )
#             Repository.append_file_with_eol(save_path, string)
#
#
# def write_all_diff_module_and_scope_tuple(patch_root):
#         for patch_dir_name in os.listdir(patch_root):
#             patch_dir_path = os.path.join(patch_root, patch_dir_name)
#             if os.path.isdir(patch_dir_path):
#                 for file_name in os.listdir(patch_dir_path):
#                     if not re.match(r'\(', file_name) and not re.match(r'Source\.txt', file_name):
#                         write_diff_module_and_scope_tuple(os.path.join(patch_dir_path, file_name))
```

VersionNumberAndExistTime.py

说明：该 Python 代码文件内容主要为漏洞检测工具。

代码如下：

```
# Author: 14281055 Liheng Chen CIT BJTU
# File Name: VersionNumberAndExistTime.py


import xlrd
import os
import re
import datetime
import Repository
import CharacterFactor
import GetModuleDiffIn


def is_diff_segment_in_file(diff_segment, diff_segment_index, dst_file_path, diff_file_path):
    bm_file_path = os.path.join(os.path.dirname(diff_file_path),
                                '(BM)' + os.path.basename(diff_file_path))
    if os.path.splitext(diff_file_path)[1] in ('.c', '.h') and os.path.exists(bm_file_path):
        GetModuleDiffIn.write_module_diff_file(diff_file_path)
        module_diff_file_path = os.path.join(os.path.dirname(diff_file_path),
                                             '(Module)' + os.path.basename(diff_file_path))
        diff_module_and_scope = GetModuleDiffIn.read_module_and_scope(module_diff_file_path,

diff_segment_index)
        if diff_module_and_scope:
            scope_list = GetModuleDiffIn.get_module_scope(diff_module_and_scope[:-2],
dst_file_path)
            if scope_list:
                return GetModuleDiffIn.is_diff_segment_in_lines(
                    diff_segment,
                    Repository.get_file_line_list(dst_file_path)[scope_list[0]:scope_list[1] + 1]
                )
            else:
                return False
    return GetModuleDiffIn.is_diff_segment_in_lines(diff_segment,
```

```
Repository.get_file_line_list(dst_file_path), False)


def is_one_vuln_in_linux_kernel_accurate(diff_file_name, patch_dir, kernel_dir):
    vuln_relative_path = diff_file_name.replace('~!@#', '\\')
    vuln_dir_path = os.path.join(kernel_dir, os.path.dirname(vuln_relative_path))
    if os.path.exists(vuln_dir_path):
        for file_name in os.listdir(vuln_dir_path):
            if os.path.isdir(os.path.join(vuln_dir_path, file_name)):
                continue
            elif re.match(
                    re.escape(os.path.splitext(os.path.basename(vuln_relative_path))[0])
                    + r'(_[1-5])?'
                    + re.escape(os.path.splitext(os.path.basename(vuln_relative_path))[1]),
                    file_name, re.I):
                for diff_segment_index, diff_segment in enumerate(
                        GetModuleDiffIn.get_diff_segment_list(os.path.join(patch_dir,
diff_file_name))):
                    if not is_diff_segment_in_file(diff_segment, diff_segment_index,
                                                   os.path.join(vuln_dir_path, file_name),
                                                   os.path.join(patch_dir, diff_file_name)):
                        break
                else:
                    return True
    return False


# def is_one_vuln_in_linux_kernel(patch_name, patch_dir, kernel_dir):
#     patch_relative_path = patch_name.replace('~！@#', '\\')
#     patch_relative_path_dir_name = os.path.dirname(patch_relative_path)
#     patch_relative_path_base_name = os.path.basename(patch_relative_path)
#     vuln_path_dir_name = os.path.join(kernel_dir, patch_relative_path_dir_name)
#     if not os.path.exists(vuln_path_dir_name):
#         return False
#     for file_name in os.listdir(vuln_path_dir_name):
#         if os.path.isdir(os.path.join(vuln_path_dir_name, file_name)):
#             continue
#         if os.path.splitext(patch_relative_path_base_name)[0] in file_name:
```

```
#                     with open(os.path.join(patch_dir, patch_name), 'r') as patch_file:
#                         for line in patch_file.readlines():
#                             match = re.match(r'-([^-].*)', line)
#                             if match and not Repository.is_string_in_file(
#                                     match.group(1).strip(),
#                                     os.path.join(vuln_path_dir_name, file_name)
#                             ):
#                                 break
#                         else:
#                             return True
#         return False

def is_one_vuln_in_linux_kernel_path(diff_file_name, kernel_dir):
    vuln_relative_path = diff_file_name.replace('~!@#', '\\')
    vuln_dir_path = os.path.join(kernel_dir, os.path.dirname(vuln_relative_path))

    if os.path.exists(vuln_dir_path):
        for file_name in os.listdir(vuln_dir_path):
            if os.path.isdir(os.path.join(vuln_dir_path, file_name)):
                continue
            elif re.match(
                    re.escape(os.path.splitext(os.path.basename(vuln_relative_path))[0])
                    + r'(_[1-5])?'
                    + re.escape(os.path.splitext(os.path.basename(vuln_relative_path))[1]),
                    file_name, re.I):
                return True
    return False


def is_vuln_in_linux_kernel_path(patch_dir, kernel_dir):
    for file_name in os.listdir(patch_dir):
        if file_name != 'Source.txt' and not re.match(r'\(', file_name):
            if not is_one_vuln_in_linux_kernel_path(file_name, kernel_dir):
                return False
    return True


def is_vuln_in_linux_kernel(patch_dir, kernel_dir):
```

```
    if Repository.is_dir_empty(patch_dir) or not is_vuln_in_linux_kernel_path(patch_dir, kernel_dir):
        return False
    for file_name in os.listdir(patch_dir):
        if file_name != 'Source.txt' and not re.match(r'\(', file_name):
            if not is_one_vuln_in_linux_kernel_accurate(file_name, patch_dir, kernel_dir):
                return False
    return True


def print_current_search(kernel_name, rate_str):
    print(rate_str + '\tSearch:Linux-' + get_version_number(kernel_name), end='')


def get_version_number(kernel_name):
    match = re.match(r'linux-(\d+(\.\d+)*)', kernel_name, re.I)
    if match:
        return match.group(1)
    else:
        return None


# def get_oldest_version_number_binary_search(patch_dir, kernel_root, rate_str):
#     kernel_name_list = sorted(os.listdir(kernel_root),
#                               key=Repository.cmp_to_key(Repository.kernel_name_compare))
#     low_index = 0
#     high_index = len(kernel_name_list) - 1
#     last_exist_index = None
#     while True:
#         if low_index == high_index:
#             print_current_search(kernel_name_list[low_index], rate_str)
#             if (last_exist_index and low_index == last_exist_index) \
#                     or is_vuln_in_linux_kernel(patch_dir, kernel_name_list[low_index]):
#                 return get_version_number(kernel_name_list[low_index])
#             else:
#                 return None
#         middle_index = int((low_index + high_index) / 2)
#         print_current_search(kernel_name_list[middle_index], rate_str)
#         if (last_exist_index and middle_index == last_exist_index) \
```

```
#                       or is_vuln_in_linux_kernel(patch_dir,
#                                                                                   get_kernel_dir(kernel_root,
kernel_name_list[middle_index])):
#                   last_exist_index = middle_index
#                   high_index = middle_index
#               else:
#                   low_index = middle_index + 1


def get_kernel_dir_path(kernel_root_path, kernel_name):
    kernel_dir_path = os.path.join(kernel_root_path, kernel_name)
    for file_name in os.listdir(kernel_dir_path):
        if re.match('linux', file_name, re.I) and os.path.isdir(file_name):
            return os.path.join(kernel_dir_path, file_name)
    return kernel_dir_path


def get_linux_kernel_version_number_list(path):
    r_workbook = xlrd.open_workbook(path)
    r_sheet = r_workbook.sheet_by_index(0)
    version_number_list = []
    for row_index in range(1, r_sheet.nrows):
        version_number = str(r_sheet.cell(row_index, 0).value).strip()
        version_number_list.append(version_number)
    return version_number_list


def get_linux_kernel_name_list(path):
    r_workbook = xlrd.open_workbook(path)
    r_sheet = r_workbook.sheet_by_index(0)
    kernel_name_list = []
    for row_index in range(1, r_sheet.nrows):
        version_number = str(r_sheet.cell(row_index, 0).value).strip()
        kernel_name = 'linux-' + version_number
        kernel_name_list.append(kernel_name)
    return kernel_name_list
```

```
def get_oldest_version_number(patch_dir, kernel_root, linux_kernel_release_time_file_path, rate_str):
    for kernel_name in get_linux_kernel_name_list(linux_kernel_release_time_file_path):
        match = re.match(r'linux-(\d+(\.\d+)*)', kernel_name, re.I)
        if match:
            version_number = match.group(1)
            kernel_dir = get_kernel_dir_path(kernel_root, kernel_name)
            if kernel_dir:
                print(rate_str + '\tSearch:Linux-' + version_number, end='')
                if is_vuln_in_linux_kernel(patch_dir, kernel_dir):
                    return version_number
    return None


def get_version_number_and_exist_time_tuple(patch_dir, kernel_root,
                                            linux_kernel_release_time_file_path, rate_str='\r'):
    if os.path.exists(patch_dir):
        oldest_kernel_version_number = get_oldest_version_number(patch_dir, kernel_root,

linux_kernel_release_time_file_path,
                                                                 rate_str)
        if oldest_kernel_version_number:
            release_time_tuple = CharacterFactor.get_linux_kernel_release_time_tuple(
                oldest_kernel_version_number,
                linux_kernel_release_time_file_path
            )
            return oldest_kernel_version_number, release_time_tuple
    return ()


def save_vuln_version_number_and_exist_time(save_path, excel_path, patch_root, kernel_root,
                                            linux_kernel_release_time_file_path,
start_row_index):
    r_workbook = xlrd.open_workbook(excel_path)
    r_sheet = r_workbook.sheets()[0]
    Repository.init_workbook(save_path,
                             ['CVE Number', 'Linux Kernel Version Number', 'Vulnerability Exist
Time'],
                             width=1.5)
```

```
    for row_index in range(start_row_index, r_sheet.nrows):
        cve_id = str(r_sheet.cell(row_index, 0).value).strip()
        rate_str = '\rRow Index:' + str(row_index) + '\t' + cve_id
        print(rate_str, end='')
        info_tuple = get_version_number_and_exist_time_tuple(os.path.join(patch_root, cve_id.upper()),
                                                             kernel_root,
linux_kernel_release_time_file_path,
                                                             rate_str)
        if info_tuple:
            version_number = info_tuple[0]
            if info_tuple[1]:
                release_time = datetime.datetime(*info_tuple[1])
            else:
                release_time = None
        else:
            version_number = None
            release_time = None
        Repository.write_workbook(save_path,
                                  {'CVE Number': cve_id, 'Linux Kernel Version Number':
version_number,
                                   'Vulnerability Exist Time': release_time}, row_index - 1,
                                  style_list=[None, None, Repository.set_date_style()])


if __name__ == '__main__':
    save_vuln_version_number_and_exist_time(
        r'C:\Users\79196\Downloads\NVD\CVE Oldest Version Number.xls',
        r'C:\Users\79196\Downloads\NVD\Linux Kernel Character Factor.xls',
        r'C:\Users\79196\Downloads\NVD\Linux Kernel Patch',
        r'D:\Linux Kernel\All',
        r'C:\Users\79196\Downloads\NVD\Linux Kernel Release Time.xls',
        start_row_index=2,
    )
```

附录 B 外文文献与翻译