利用 Jaccard 系数进行共享代码分析

一、基础知识

1.1 共享代码分析

共享代码分析，也称为相似性分析，是我们通过估计它们共享的预编译源代码的分比来比较两个恶意软件样本的过程。它不同于共享属性分析，共享属性分析是根据意软件的外部属性(例如，它们所使用的桌面图标，或者它们回连的服务器)来比较意软件样本。

在逆向工程中，共享代码分析有助于识别可以一起分析的样本(因为它们来自相的恶意软件工具包，或者是相同恶意软件家族的不同版本)，这可以确定是否为相同的发人员部署的一组恶意软件样本。

首先，介绍本实验将使用的恶意软件测试样本。然后，你将学习数学上的相似性比较和 Jaccard 系数的概念，这是一种根据共享特征来比较恶意软件样本的集合理论方法。接下来，本文将介绍特征的概念，以说明如何将它们与 Jaccard 系数结合使用，来估算两个恶意软件样本共享的代码量。你还将学习如何根据可用性来评价恶意软件的特征。最后，本文通过网络可视化知识，创建多个层次的恶意软件代码共享可视化效果图。

1.2 实验所用的恶意软件样本

在本章中，本文使用真实场景中彼此共享大量代码的恶意软件家族来进行实验，这些数据集来自 Mandiant 和 Mila Parkour 公司。你可能不知道这些恶意软件样本属于哪个家族，也不知道新发现的恶意软件样本与先前看到的样本之间有多大的相似程度。但是通过浏览一些确切知道的样本将是一个很好的实践，因为它允许你验证对样本相似性的自动推断是否与你对哪些样本实际上属于同一组的知识一致。

为了识别 APT1 数据集上样本的家族名称，我将每个样本输入卡巴斯基反病毒引警。卡巴斯基能够用鲁棒的层次分类方法对 30104 个样本进行分类(例如 troian.win32.jorik.skor.akr 表示 jorik.skor 家族)，将 41 830 个样本分配到“未知”类中，给其余 28 481 个样本分配通用的标签(例如，通用的“win32 Trojan”标签)。

由于卡巴斯基的标签具有不一致性(例如 jorik 家族等一些卡巴斯基的标签分组代表了一系列非常广泛的恶意软件，而例如 webprefix 等其他的标签则代表了特定的变种集合)，以及卡巴斯基经常漏报或者错误标记恶意软件的事实，我选择了卡巴斯基检测出的可信度较高的

七类恶意软件，具体包括 dapato、pasta、skor、vbnawebprefix、xtoober 和 zango7 个家族。

1.3 通过特征提取对样本进行比较

在攻击者编译两个恶意二进制文件之前，我们如何开始估计它们共享的代码量呢?你可能考虑使用很多种方法来解决这个问题，但是关于这个问题已经发表的数百篇计算机科学研究论文中，有一个共同的主题是:为了估计二进制文件之间共享代码的数量，在进行比较之前，我们要将恶意软件样本进行“特征袋”（bag offeatures）分组。

我所说的特征是指我们在估计样本之间的代码相似性时任何可能需要考虑的恶意软件属性。例如，我们使用的特征可以是从二进制文件中提取的可打印字符串。我们不是把样本看作是一个由函数、导人的动态库等等组成的相互连接的系统，而是把恶意软件看作是一系列便于计算的独立特征袋(例如，从恶意软件中提取的一组字符串)。

1.4 特征袋模型如何工作

要了解特征袋是如何工作的，请考虑如图 1 所示两个恶意软件样本之间的维恩图。在这里，样本 A 和样本 B 显示为特征袋(特征在维恩图中用圆表示)。我们可以通过检查这两个样本之间共享哪些特征来比较它们。计算两组特征之间的重叠很快，并且可以根据我们提出的任意特征来比较恶意软件样本的相似性。

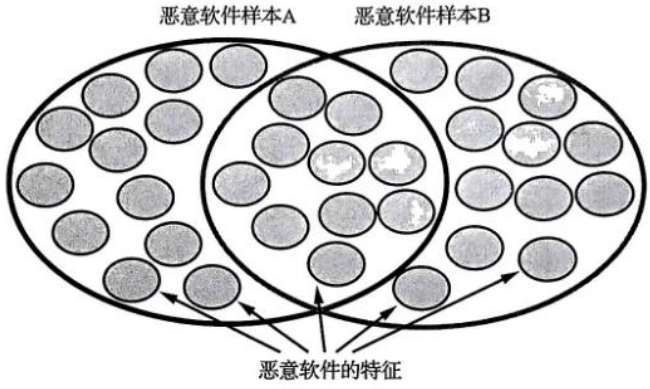

图 1 恶意代码共享分析的“特征袋”模型示意图

例如，在处理加壳的恶意软件时，我们可能希望使用基于恶意软件动态运行日志的特征，这是由于在沙箱中运行恶意软件是让恶意软件自行解压的一种方式。在其他情况下，我们可以使用从静态恶意软件二进制文件中提取的字符串来执行比较。

1.5 使用 Jaccard 系数量化相似性

一旦你将一个恶意软件样本表示成为一个特征袋，你就需要度量该样本的特征袋与其他样本的特征袋之间的相似性程度。为了估计两个恶意软件样本之间的代码共享程度我们使用了一个具有以下属性的相似性函数：

它生成一个归一化后的值，这样恶意软件样本之间的相似性比较可以放在相同的尺度上进行。常规来说，这个函数应该从范围 0(没有代码共享)到 1(样本之间代码进行百分之百共享)之间产生值。

该函数应该帮助我们对两个样本之间的代码共享做出准确的估计(我们可以通过实验从经验上确定这一点)。

我们应该能够很容易地理解为什么函数模型代码相似性很好(它不应该是一个复杂的数学黑盒子，需要花很多精力去理解或解释)。

Jaccard 系数是一个具有这些性质的简单函数。事实上，即使安全研究社区已经尝试过其他代码相似性估计的数学方法(例如，余弦距离、1 距离、欧式[L2]距离，等等)，但是 Jaccard 系数是最广泛采用的方法，并目有充分的理由。它简单而直观地表达了两组恶意软件特征之间的重叠程度，为我们提供了两组恶意软件特征中共有特征的百分比该百分比由两组恶意软件中所有存在特征的百分比进行归一化而来。

图 2 展示了 Jaccard 系数值的样例。

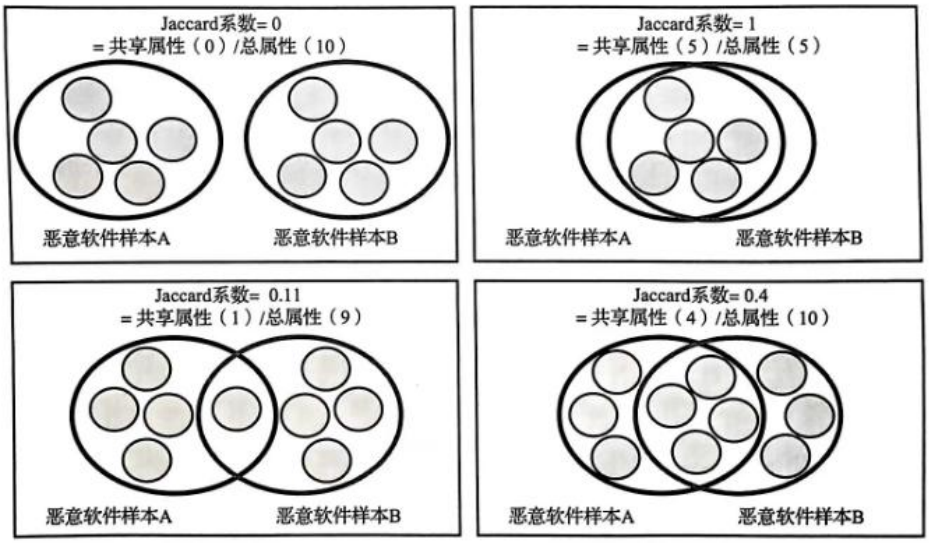

图 2 Jaccard 系数背后思想的示意图

这里显示了从四对恶意软件样本中提取的四对恶意软件特征。每幅图像分别显示了两组样本之间共享的特征、两组之间没有共享的特征以及给定的恶意软件样本对和相关特征的 Jaccard 系数结果。你可以看到，样本之间的 Jaccard 系数就是样本之间共享的特征数量除以

在维恩图中绘制的特征总数。

1.6 构建相似图

现在你已了解识别恶意软件代码共享方法背后的概念，那么让我们构建一个简单的系统，来对恶意软件数据集进行这个分析。

首先，我们通过提取我们想要使用的特征来估计样本共享的代码量。这些可以是前面描述的任何特征，例如基于导人地址表的函数、字符串、N-gram 指令或者是 N-gram 动态行为。在这里，我们将使用可打印字符串作为特征，原因是它们的执行效果很好并且易于提取和理解。

一旦我们提取了字符串特征，我们需要遍历每一对恶意软件样本，使用 Jaccard 系数来比较它们的特征。然后，我们需要构建一个代码共享图。为此，我们首先需要确定一个值，该阈值定义了两个样本共享代码的比例--我在我自己的研究中使用的标准值是 0.8.如果给定的一对恶意软件样本的 Jaccard 系数值比这个值要大，我们将在它们之间创建一个连接来进行可视化。最后一步是研究图表，看看哪些样本通过共享代码关系进行连接。

代码 1 到代码 5 包含了我们的示例程序。由于代码很长，所以我把它分成了几块，并对每一块进行解释。代码 1 导入了我们将使用的库，并声明了 jaccard()函数，该函数计算了两个样本特征集之间的 Jaccard 系数。

```
#!/usr/bin/python

import argparse
import os
import networkx
from networkx.drawing.nx_pydot import write_dot
import itertools

def jaccard(set1, set2):
    """
    计算两个集合之间的 Jaccard 距离，方法是取它们的交集、并
    集，然后将交集中的元素数除以它们的并集中的元素数。
    """
    intersection = set1.intersection(set2)
    intersection_length = float(len(intersection))
    union = set1.union(set2)
    union_length = float(len(union))
    return intersection_length / union_length
```

代码 1 用于计算两个样本之间 Jaccard 系数的导入和辅助函数

接下来，在代码 2 中，我们声明了两个额外的实用程序函数：getstring()和 pecheck()，getstring()在我们将要分析的恶意软件文件中找到可打印的字符串序列集 pecheck()确保目标文件确实是 Windows PE 文件。在稍后对目标恶意软件二进制文件进行特征提取的时候，我

们将使用这些函数。

```
def getstrings(fullpath):
    """
    从 'fullpath' 参数指示的二进制文件中提取字符串，然后返回二进
    制文件中已去重的字符串集。
    """
    strings = os.popen("strings '{0}'".format(fullpath)).read()
    strings = set(strings.split("\n"))

    return strings

def pecheck(fullpath):
    """
    做一个粗略的合理性检查以确保 'fullpath' 指示的文件是 Windows
    PE 可执行文件（PE 可执行文件以 'MZ' 这两个字节开头）
    """
    return open(fullpath).read(2) == "MZ"
```

代码 2 声明我们将在特征提取中所使用的函数

接下来，在代码 3 中，我们解析用户的命令行参数。这些参数包括我们将要分析的恶意软件所在的目标目录、我们将我们构建的共享代码网络所要写人的.dot输出文件，以及 Jaccard 系数的阈值，这个值决定了两个样本之间的 Jaccard 系数要达到多少，才能让程序决定它们是彼此共享一个公共代码库。

```
If __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description=" 识别恶意软件样本的相似性并构建相似性图 "
    )

    parser.add_argument(
        "target_directory",
        help=" 包含恶意软件的目录 "
    )

    parser.add_argument(
        "output_dot_file",
        help=" 保存 DOT 输出图文件的位置 "
    )

    parser.add_argument(
        "--jaccard_index_threshold", "-j", dest="threshold", type=float,
        default=0.8, help=" 在样本间建立“边”的阈值 "
    )

    args = parser.parse_args()
```

代码 3 解析用户的命令行参数

接下来，在代码 4 中，我们使用前面声明的辅助函数来完成程序的主要工作:在目标目录中查找 PE 二进制文件，从中提取特征，并初始化一个用来表示二进制文件之间相似性关系的网络。

```
malware_paths = []  # 我们将存储恶意软件文件的路径
malware_features = dict()  # 我们将存储恶意软件字符串的路径
graph = networkx.Graph()  # 相似性图

for root, dirs, paths in os.walk(args.target_directory):
    # 遍历目标目录树，并存储所有文件路径
    for path in paths:
        full_path = os.path.join(root, path)
        malware_paths.append(full_path)

# 过滤掉所有不是 PE 文件的路径
malware_paths = filter(pecheck, malware_paths)

# 获取并存储所有 malware PE 文件中的字符串
for path in malware_paths:
    features = getstrings(path)
    print "Extracted {0} features from {1} ...".format(len(features), path)
    malware_features[path] = features

    # 将每个恶意软件添加到图里
    graph.add_node(path, label=os.path.split(path)[-1][:10])
```

代码 4 从目标目录的 PE 文件中提取特征并初始化共享代码网络

从我们的目标样本中提取特征后，我们需要迭代每对恶意软件样本，使用 Jaccard 系数比较它们的特征。我们在代码 5 中执行此操作。我们还构建了一个代码共享图如果样本的 Jaccard 系数高于某个用户定义的阈值，则将它们连接在一起。在我的研究中，我发现最有效的阈值是 0.8。

```
# 遍历所有恶意软件对
for malware1, malware2 in itertools.combinations(malware_paths, 2):

    # 对当前对计算 jaccard 距离
    jaccard_index = jaccard(malware_features[malware1], malware_features[malware2])

    # 如果 jaccard 距离大于阈值，增加一条边
    if jaccard_index > args.threshold:
        print malware1, malware2, jaccard_index
        graph.add_edge(malware1, malware2, penwidth=1+(jaccard_index-args.threshold)*10)

# 将图写入磁盘便于我们进行可视化
write_dot(graph, args.output_dot_file)
```

代码 5 用 Python 创建代码共享图

在使用代码 1 到代码 5 中的代码对 APT1 恶意软件样本处理后，就生成如图 3 所示的图表，注意图 3 只展示了完整相似图的一部分。要展现这个图表，你需要使用 Graphviz 工具的 fdp 并输人命令 fdp -Tpng network.dot -o network.png 来完成。

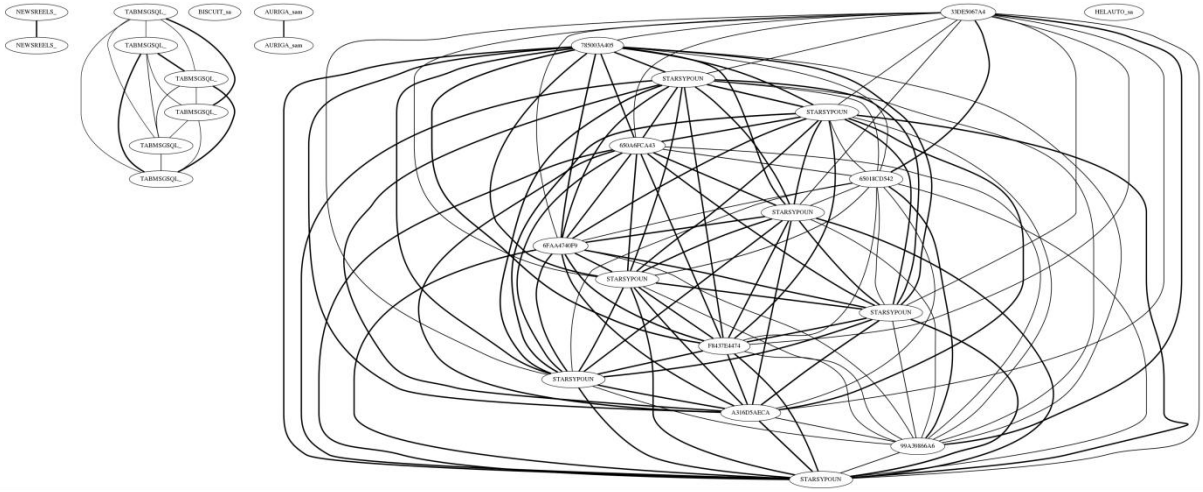

图 3 基于字符串的 APT1 样本相似性图（部分）

二、实验步骤

2.1 实验环境

实验环境为 linux 系统的 python2.7 版本。

可在虚拟机中使用 linux 操作系统，比如 ubuntu，centos。在虚拟机中安装好 python2.7 之后，我们也可以安装 python 开发环境，比如 pycharm，vscode。以上均在网上有相关教程，可自行搜索完成。

2.2 导入相关库

搭建好实验环境后，新建 python 项目，将附件 1 的代码和附件 2 的样本数据放入项目中，并将附件 1 的代码更改为英文名，比如我将其改为了 listing_5_1.py。打开附件 1 的代码，下载并导入相关库。

```
import argparse
import os
import networkx
from networkx.drawing.nx_pydot import write_dot
import itertools
import pprint
import matplotlib.pyplot as plt
import numpy as np
```

2.3 计算 Jaccard 系数示例

为了计算恶意软件样本之间的 Jaccard 系数，我们首先通过提取我们想要使用的特征来估计样本共享的代码量。这些可以是前面描述的任何特征，例如基于导人地址表的函数、字符串、N-gram 指令或者是 N-gram 动态行为。在这里，我们将使用可打印字符串作为特征，原因是它们的执行效果很好并且易于提取和理解。

我们挑选了三个恶意软件样本来计算他们的 Jaccard 系数，首先从恶意软件样本源码提取它们的可打印字符串。以下是样本 1“F6655E39465C2FF5B016980D918EA028”的源码片段：

样本 1 源码片段

```
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00
```

我们对这个源码片段进行分析：

1、4D 5A: 代表 "MZ"，是 Microsoft Executable (MZ) 文件的标识符。这是历史遗留问题，因为早期的 DOS 可执行文件使用这个标识符。

2、90 00 03 00 00 00 04 00 00 00: 这是 PE 文件头的偏移量，表示 PE 文件头位于 MZ 头部之后的第 0x0000000390 个字节处。

2、FF FF 00 00: 这是 MZ 头部的保留字段，通常是空的，可以忽略。

3、B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00: 这是 MOV EAX, 0x40 这条汇编指令，它将十六进制数 0x40 写入 EAX 寄存器。在 PE 文件头部中，这条指令的作用是设置 PE 文件头部的地址，通常它会指向 PE 文件头的开始位置。

4、E8 00 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21: 这一部分包含了 CALL 指令，MOV 指令，MOV AH, 0x09 指令，以及 INT 21H 指令。 这些指令可能用于加载 PE 文件头的地址和执行一些简单的操作，例如打印错误信息。

5、54 68 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00: 这是一个 ASCII 字符串，它代表着 "This program cannot be run in DOS mode." 这句话。

其中"This program cannot be run in DOS mode."就是我们需要提取的可打印字符串。注意前面的 CD 21 指令中的 21 对应着 ASCII 码表中的 !(感叹号)，这里 21 并不直接代表 !，而是代表着 INT 21H 指令的立即数，用来指定中断号。但是 INT 21H 指令可能会调用一些 DOS 系统服务，这些服务可能会最终将 ! 符号输出到屏幕上。所以我们提取出来的可打印字符串变成了"！This program cannot be run in DOS mode."，我们将其放入特征袋中。将所有源码中的可打印字符串提取出来，放入集合中，我们就得到了这个样本的特征袋。

以下分别是我们提取的三个恶意软件样本的特征集合构成的特征袋：

样本 1“F6655E39465C2FF5B016980D918EA028”：

{__p__fmode，%X @，WaitForSingleObject，*(SY)#，=H @，D$ RP，SSPS，_except_handler3，DisconnectNamedPipe，GetCurrentProcess，_XcptFilter，USER32.dll，h !@，D$dSV，= @ @，sprintf，WriteFile，L$(h，=, @，ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-/，%p @，t2Ht，PeekNamedPipe，%d @，T$\h@0@，D$(Qj，.rsrc，MSVCRT.dll，PADDINGXXPADDINGPADD，Rich，T$4hT0@，D$DP，%` @，D$ h，D$\QPU，_beginthread，%\ @，=(@，exit，T$(Q，D$8D，_acmdln，cmd.exe，LoadStringA，CreateThread，XPVSS，free，__getmainargs，ExitThread，D$\h，_^][，TerminateProcess，\$Hf，L$(hT0@，@.data，_exit，t$p3，TerminateThread，SetEvent，KERNEL32.dll，T$4hp0@，%h @，[t V，_itoa，GetStartupInfoA，L$,h，T$ QRP，WaitForMultipleObjects，SUVWj:h，T$0f，send = %d，*(SY)# ，atol，GetComputerNameA，DuplicateHandle，__setusermatherr，QRSSSj，UVWj，!This program cannot be run in DOS mode.，CreateEventA，_strnicmp，_controlfp，WS2_32.dll，%l @，T$,h，malloc，hSVW，D$$V，\$lf，.text，L$$VQR，%x @，|$XS，`.rdata，__p__commode，strchr，_adjust_fdiv，GetModuleHandleA，ReadFile，L$\hL0@，=8 @，CloseHandle，*(SY)# cmd，_initterm，5< @，SVWj，CreatePipe，>"u:F，CreateProcessA，Sleep，L$pj，__set_app_type}

样本 2“STARSYPOUND_sample_9EA3C16194CE354C244C1B74C46CD92E”：

{__p__fmode，%X @，WaitForSingleObject，*(SY)#，=H @，D$ RP，SSPS，_except_handler3，DisconnectNamedPipe，GetCurrentProcess，_XcptFilter，USER32.dll，h !@，D$dSV，= @ @，sprintf，WriteFile，L$(h，=, @，ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-/，%p @，t2Ht，PeekNamedPipe，%d @，T$\h@0@，D$(Qj，.rsrc，MSVCRT.dll，Rich，T$4hT0@，D$DP，%` @，D$ h，D$\QPU，_beginthread，%\ @，=(@，exit，T$(Q，D$8D，_acmdln，PAPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGP

ADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDIN
GPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADD
INGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPA
DDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXX
PADDINGPADDINGXXPADDINGPADDINGXXPADDING，cmd.exe，LoadStringA，CreateTh
read，XPVSS，free，__getmainargs，ExitThread，D$\h，_^][，TerminateProcess，\$Hf，L$(h
T0@，@.data，_exit，t$p3，TerminateThread，SetEvent，KERNEL32.dll，T$4hp0@，%h @，
[t V，_itoa，GetStartupInfoA，L$,h，T$ QRP，WaitForMultipleObjects，SUVWj:h，T$0f，
send = %d，*(SY)# ，atol，GetComputerNameA，DuplicateHandle，__setusermatherr，QRSS
Sj，UVWj，!This program cannot be run in DOS mode.，CreateEventA，_strnicmp，_contr
olfp，WS2_32.dll，%l @，T$,h，malloc，hSVW，D$$V，\$lf，.text，L$$VQR，%x @，|$X
S，`.rdata，__p__commode，strchr，_adjust_fdiv，GetModuleHandleA，ReadFile，L$\hL0@，
=8 @，CloseHandle，*(SY)# cmd，_initterm，5< @，SVWj，CreatePipe，>"u:F，CreatePro
cessA，Sleep，L$pj，__set_app_type}

样本 3“WEBC2-AUSOV_sample_097B5ABB53A3D84FA9EABDA02FEF9E91”：

{__p__fmode，vX |iR ，h0A@，_except_handler3，%P0@，ExpandEnvironmentStringsA，_
XcptFilter，%X0@，GetTempPathA，h8A@，%temp%，h0@@，-$81# 6%2+:，%x0@，strs
tr，PhxA@，strncmp，VWh|@@，LZ32.dll，RegCreateKeyExA，GetModuleFileNameA，strc
hr，2??,，exit，=<0@，strrchr，_acmdln，GetModuleHandleA，XPVSS，h$@@，__getmaina
rgs，hD@@，t _^，RegSetValueExA，@.data，_exit，%\0@，RegCloseKey，vX Rich，L
ZOpenFileA，hP$@，KERNEL32.dll，MSVCRT.dll，LoadLibraryA，%l0@，strncpy，h A@，
GetStartupInfoA，urlmon.dll，InternetCloseHandle，DeleteFileA，__setusermatherr，InternetRe
adFile，__p__commode，LZCopy，atoi，hh@@，GetProcAddress，hX@@，URLDownloadT
oFileA，InternetOpenA，!This program cannot be run in DOS mode.，CopyFileA，ADVAPI
32.dll，_controlfp，%L0@，vX |i\ ，hSVW，i]\@GZ.，.text，U^CD，hD$@，`.rdata，_adj
ust_fdiv，%H0@，wininet.dll，\svchost.exe，CloseHandle， -s ，_initterm，InternetOpenUrl
A，>"u:F，CreateProcessA，GetLongPathNameA，Sleep，LZClose，vY ;vX %PS ，__set
_app_type}

各样本提取的特征数如表 1 所示。

表 1 各样本特征数

样本名	特征数量
F6655E39465C2FF5B016980D918EA028	113
STARSYPOUND_sample_ 9EA3C16194CE354C244C1B74C46CD92E	113
WEBC2-AUSOV_sample_ 097B5ABB53A3D84FA9EABDA02FEF9E91	89

根据 Jaccard 系数计算公式：

Jaccard 系数=共享特征数/总特征数

我们分别计算这三个样本之间的 Jaccard 系数，如表 2 所示。

表 2 各样本对的 Jaccard 系数

样本对	共享特征数	总特征数	Jaccard 系数
样本 1 和样本 2	112	114	0.98
样本 1 和样本 3	29	173	0.17
样本 2 和样本 3	29	173	0.17

以下是计算三个样本之间 Jaccard 系数的韦恩图示例：

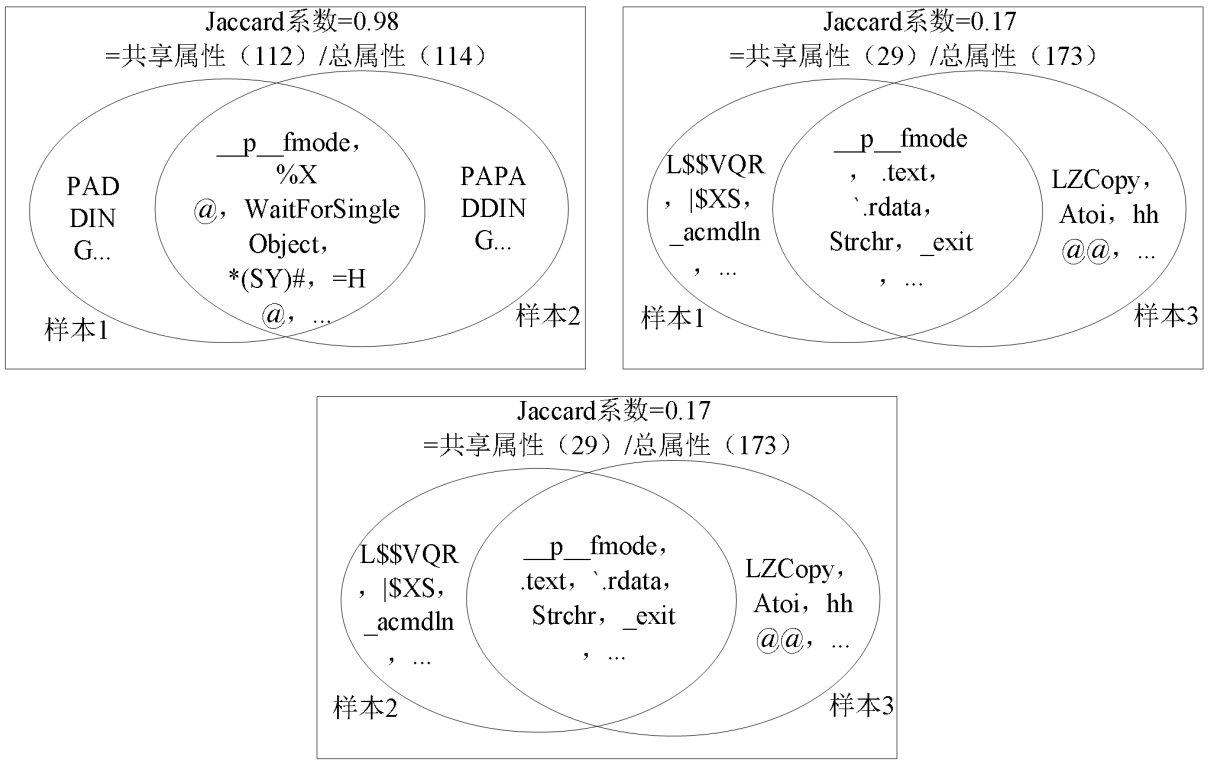

图 4 三个样本之间的 Jaccard 系数计算韦恩图示例

各软件样本之间的 Jaccard 系数代表它们的代码共享程度，我们通过样本间 Jaccard 系数表格来直观地查看这种代码共享关系，如表 3 所示。

表 3 样本间 Jaccard 系数

Jaccard 系数	样本 1	样本 2	样本 3
样本 1	1	0.98	0.17
样本 2	0.98	1	0.17
样本 3	0.17	0.17	1

在图 11 中，色块颜色的深浅反映了样本间的代码共享程度，颜色越深代表共享程度越高。我们通常把共享程度高于某一阈值的样本归类为同一家族。

2.4 绘制相似图

在终端运行代码：python code/listing_5_1.py data output.dot -j 0.8，如图 5 所示。

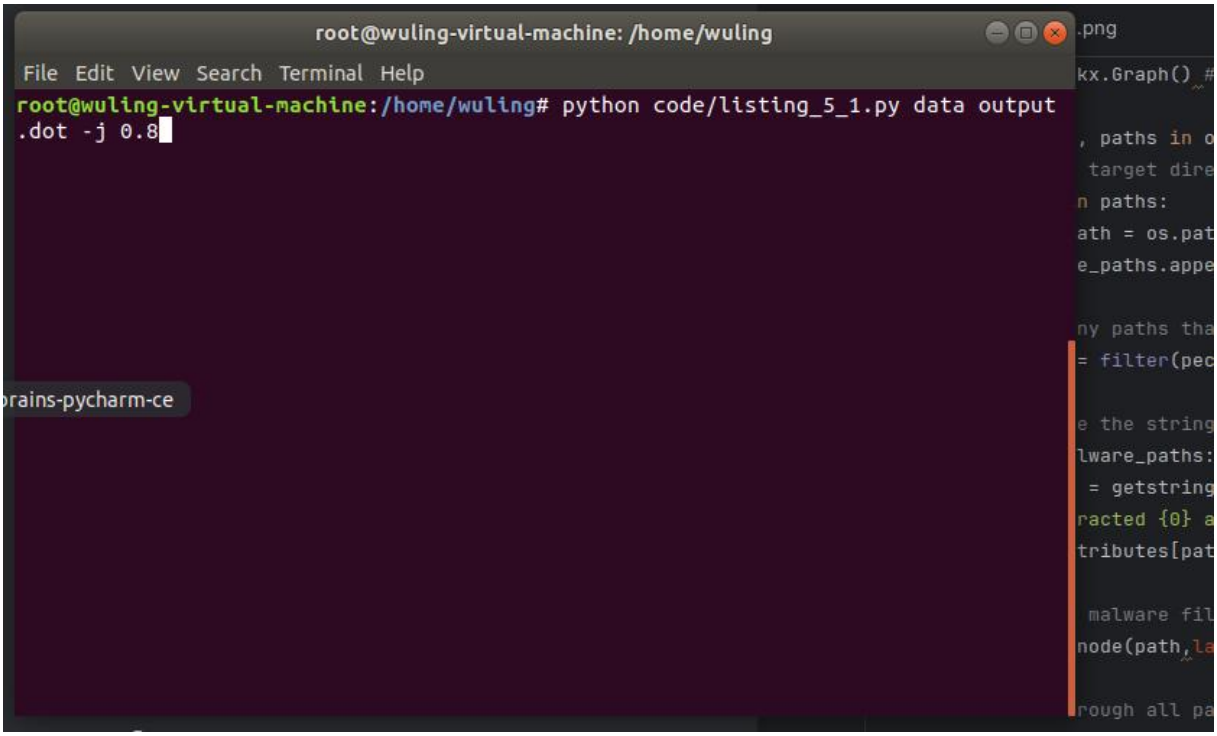

图 5 终端运行代码

如果此时报错，提示 ImportError: No module named pydot，是因为你的 Python 环境中缺少 pydot 库。pydot 是一个用于与 Graphviz 图形库交互的 Python 库，用于将 NetworkX 图转换为 DOT 文件，以便用 Graphviz 可视化。

我们首先在终端输入 sudo apt-get install graphviz，这会安装 Graphviz 图形库，它是 pydot 的依赖。然后使用 pip install pydot 命令安装 pydot 库。

运行代码之后我们会得到一个 dot 文件：output.dot。想要查看此 dot 文件，我们可以使用 dot 命令将 DOT 文件转换为图像文件，例如 PNG 或 SVG，然后使用图像查看器打开。例如，将 output.dot 文件转换为 PNG 图像文件，可以在终端中输入以下命令：dot output.dot -Tpng -o output.png。这将在当前目录生成一个名为 output.png 的 PNG 图像文件，你可以使用图像查看器打开它，如图 6 所示。

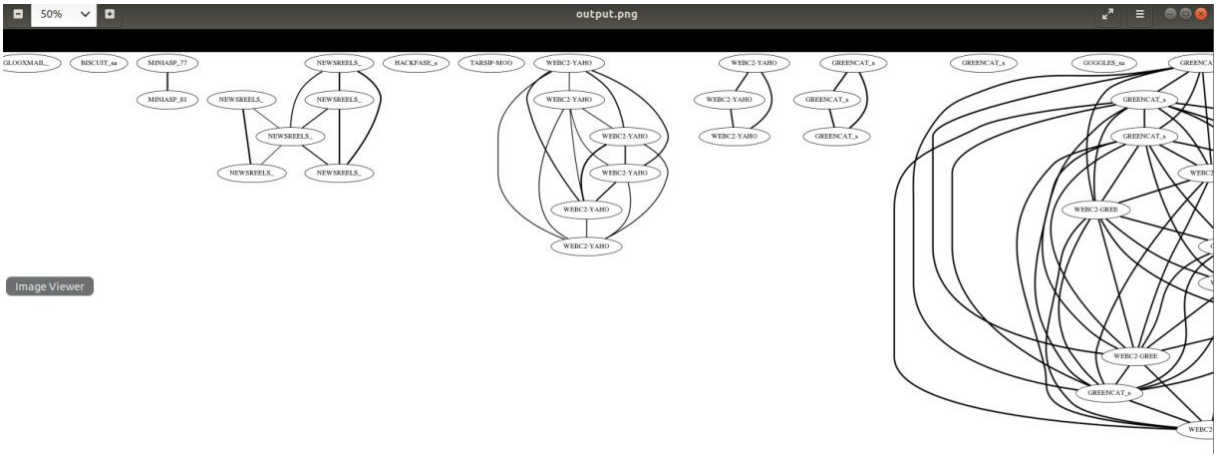

图 6 output.png

图 6 展示了恶意软件代码共享可视化效果。

2.5 制作 jaccard 系数表格

我们在修改如下代码创建 jaccard 系数矩阵，如图 7 所示：

```
jaccard_matrix = {}
for malware1 in malware_paths:
    malware1_name = os.path.split(malware1)[-1]
    jaccard_matrix[(malware1_name, malware1_name)] = 1


for malware1, malware2 in itertools.combinations(malware_paths, 2):
    jaccard_index = jaccard(malware_attributes[malware1], malware_attributes[malware2])


    if jaccard_index > args.threshold:
        print(malware1, malware2, jaccard_index)
        graph.add_edge(malware1, malware2, penwidth=1 + (jaccard_index - args.threshold) * 10)


    malware1_name = os.path.split(malware1)[-1]
    malware2_name = os.path.split(malware2)[-1]
    jaccard_matrix[(malware1_name, malware2_name)] = jaccard_index
    jaccard_matrix[(malware2_name, malware1_name)] = jaccard_index
```

图 7 修改的 jaccard 系数矩阵代码

修改后的代码新增了以下功能：

1、创建字典存储 Jaccard 系数: 在主程序中，创建了一个名为 jaccard_matrix 的字典，用

于存储 Jaccard 系数。

2、存储 Jaccard 系数: 在计算 Jaccard 系数后，将 (malware1_name, malware2_name) 作为键，Jaccard 系数作为值存储到 jaccard_matrix 中。

然后我们添加以下代码制作 jaccard 系数表格，表格的横纵坐标代表恶意样本，表格内容则代表样本之间的 jaccard 系数。代码如图 8 所示。

```
with open("jaccard_table.csv", "w") as f:
    malware_names = sorted(set([name for pair in jaccard_matrix.keys() for name in pair]))

    f.write('jaccard_index,'+",".join(malware_names) + "\n")
    for malware1_name in malware_names:
        row = [malware1_name]
        for malware2_name in malware_names:
            if (malware1_name, malware2_name) in jaccard_matrix:
                row.append(jaccard_matrix[(malware1_name, malware2_name)])
            else:
                row.append(0)
        f.write(",".join(str(x) for x in row) + "\n")
```

图 8 制作 jaccard 系数表格的代码

代码解释：

1、打开文件: 使用 open("jaccard_table.csv", "w") 打开文件 jaccard_table.csv，并以写入模式 "w" 打开。

2、写入表头: 使用 f.write('jaccard,'+",".join(malware_names) + "\n") 写入表头，表头包含所有恶意软件名称。

3、写入数据: 遍历所有恶意软件名称，对于每个恶意软件，将它的名称和与其他恶意软件的 Jaccard 系数写入文件。

4、关闭文件: 使用 f.close() 关闭文件。

此时在终端运行代码后，我们会得到一个名为 jaccard_table.csv 的 csv 表格，表格内容如图 9 所示。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	jaccard_index	1F2EB7B09(	33DE5067A4	36CD49AD6	65018CD54	650A6FCA4	6FAA4740F	785003A40	8442AE37B	99A39866A	A316D5AEC	AURIGA_sar	AURIGA_sar	B07322743	BANGAT_sar	BANGAT
2	1F2EB7B090018D97	1	0.5897436	0.0484293	0.5897436	0.6986301	0.6986301	0.6986301	0.9824561	0.6	0.7103448	0.023622	0.023622	1	0.0075363	0.0065
3	33DE5067A433A6EC	0.5897436	1	0.0537772	0.9852941	0.8493151	0.8493151	0.8493151	0.6	0.9852941	0.8493151	0.0217391	0.0217391	0.5897436	0.0084781	0.0074
4	36CD49AD631E9913	0.0484293	0.0537772	1	0.0537772	0.0537772	0.0537772	0.0537772	0.0484293	0.0537772	0.0537772	0.0121212	0.0121212	0.0484293	0.0170622	0.0184
5	65018CD542145A37	0.5897436	0.9852941	0.0537772	1	0.862069	0.8493151	0.8493151	0.5897436	0.9852941	0.8493151	0.0217391	0.0217391	0.5897436	0.0084781	0.0074
6	650A6FCA433EE245	0.6986301	0.8493151	0.0537772	0.862069	1	0.9852941	0.9852941	0.6986301	0.8493151	0.9852941	0.0217391	0.0217391	0.6986301	0.0083402	0.0074
7	6FAA4740F99408D4	0.6986301	0.8493151	0.0537772	0.8493151	0.9852941	1	1	0.6986301	0.8493151	0.9852941	0.0217391	0.0217391	0.6986301	0.0083402	0.0074
8	785003A405BC7A4E	0.6986301	0.8493151	0.0537772	0.8493151	0.9852941	1	1	0.6986301	0.8493151	0.9852941	0.0217391	0.0217391	0.6986301	0.0083402	0.0074
9	8442AE37B91F2798	0.9824561	0.6	0.0484293	0.5897436	0.6986301	0.6986301	0.6986301	1	0.5897436	0.6986301	0.023622	0.023622	0.9824561	0.0075363	0.0065
10	99A39866A657A10C	0.6	0.9852941	0.0537772	0.9852941	0.8493151	0.8493151	0.8493151	0.5897436	1	0.862069	0.0217391	0.0217391	0.6	0.0084781	0.0074
11	A316D5AECA269CA8	0.7103448	0.8493151	0.0537772	0.8493151	0.9852941	0.9852941	0.9852941	0.6986301	0.862069	1	0.0217391	0.0217391	0.7103448	0.0083402	0.0074
12	AURIGA_sample_6E	0.023622	0.0217391	0.0121212	0.0217391	0.0217391	0.0217391	0.0217391	0.023622	0.0217391	0.0217391	1	0.96	0.023622	0.0020347	0.0019
13	AURIGA_sample_CI	0.023622	0.0217391	0.0121212	0.0217391	0.0217391	0.0217391	0.0217391	0.023622	0.0217391	0.0217391	0.96	1	0.023622	0.0020347	0.0019
14	B07322743778B586	1	0.5897436	0.0484293	0.5897436	0.6986301	0.6986301	0.6986301	0.9824561	0.6	0.7103448	0.023622	0.023622	1	0.0075363	0.0065
15	BANGAT_sample_4(	0.0075363	0.0084781	0.0170622	0.0084781	0.0083402	0.0083402	0.0083402	0.0075363	0.0084781	0.0083402	0.0020347	0.0020347	0.0075363	1	0.4885
16	BANGAT_sample_4(	0.0065649	0.0074042	0.0184246	0.0074042	0.0074042	0.0074042	0.0074042	0.0065649	0.0074042	0.0074042	0.0019794	0.0019794	0.0065649	0.4885576	
17	BANGAT_sample_7(	0.0075363	0.0084781	0.0170622	0.0084781	0.0083402	0.0083402	0.0083402	0.0075363	0.0084781	0.0083402	0.0020347	0.0020347	0.0075363	0.9997238	0.4885
18	BANGAT_sample_8E	0.009131	0.0098917	0.0192616	0.0098917	0.0098917	0.0098917	0.0098917	0.009131	0.0098917	0.0098917	0.0020215	0.0020215	0.009131	0.5242653	0.8202
19	BANGAT_sample_BI	0.0057522	0.0064895	0.0175936	0.0064895	0.0064895	0.0064895	0.0064895	0.0057522	0.0064895	0.0064895	0.0018606	0.0018606	0.0057522	0.8040195	0.4703
20	BANGAT_sample_DE	0.0057522	0.0064895	0.0175936	0.0064895	0.0064895	0.0064895	0.0064895	0.0057522	0.0064895	0.0064895	0.0018606	0.0018606	0.0057522	0.8042341	0.4702
21	BANGAT_sample_E1	0.0057651	0.0065041	0.017631	0.0065041	0.0065041	0.0065041	0.0065041	0.0057651	0.0065041	0.0065041	0.0018647	0.0018647	0.0057651	0.8061748	0.4707
22	BANGAT_sample_EF	0.0065332	0.0073686	0.0183413	0.0073686	0.0073686	0.0073686	0.0073686	0.0065332	0.0073686	0.0073686	0.0019699	0.0019699	0.0065332	0.4871312	0.8576

图 9 jaccard_table.csv

2.6 绘制相似性灰度热图

我们再添加以下代码以绘制恶意样本的相似性灰度热图。添加的代码如图 10 所示。

```
fig, ax = plt.subplots()
malware_names = sorted(set([name for pair in jaccard_matrix.keys() for name in pair]))
num_malware = len(malware_names)
matrix = np.zeros((num_malware, num_malware))

for i, malware1_name in enumerate(malware_names):
    for j, malware2_name in enumerate(malware_names):
        if (malware1_name, malware2_name) in jaccard_matrix:
            # if jaccard_matrix[(malware1_name, malware2_name)] > args.threshold:
            matrix[i, j] = jaccard_matrix[(malware1_name, malware2_name)]

fig.set_size_inches(20, 20)
ax.imshow(matrix, cmap="gray_r", interpolation="nearest")
# ax.imshow(matrix, cmap="gray", interpolation="nearest")
ax.set_xticks(np.arange(num_malware))
ax.set_yticks(np.arange(num_malware))
ax.set_xticklabels(malware_names, rotation=45, ha="right", fontsize=4)
ax.set_yticklabels(malware_names, fontsize=4)
ax.set_title("Jaccard Similarity Heatmap")
plt.tight_layout()
plt.savefig("jaccard_heatmap_r.svg", format='svg')
plt.show()
```

图 10 绘制相似性灰度热图的代码

这段代码会绘制一个灰度热图，其中 Jaccard 系数越大的位置将显示为越接近黑色，而 Jaccard 系数越小的位置将显示为越接近白色，从而直观地展示出恶意软件样本之间的相似度关系。

运行代码后如果报错，提示 ImportError: No module named _tkinter, please install the python-tk package，这个错误信息 ImportError: No module named _tkinter, please install the python-tk package 表明你的 Python 环境中缺少 tkinter 库。tkinter 是 Python 的标准 GUI 库，它依赖于 _tkinter 模块，该模块是 Python 与 Tcl/Tk GUI 工具包的接口。

解决方法：在 Ubuntu 中，使用 apt-get 命令安装 python-tk 包：sudo apt-get install python-tk。

运行之后得到 jaccard_heatmap_r.svg，打开之后如图 11 所示。

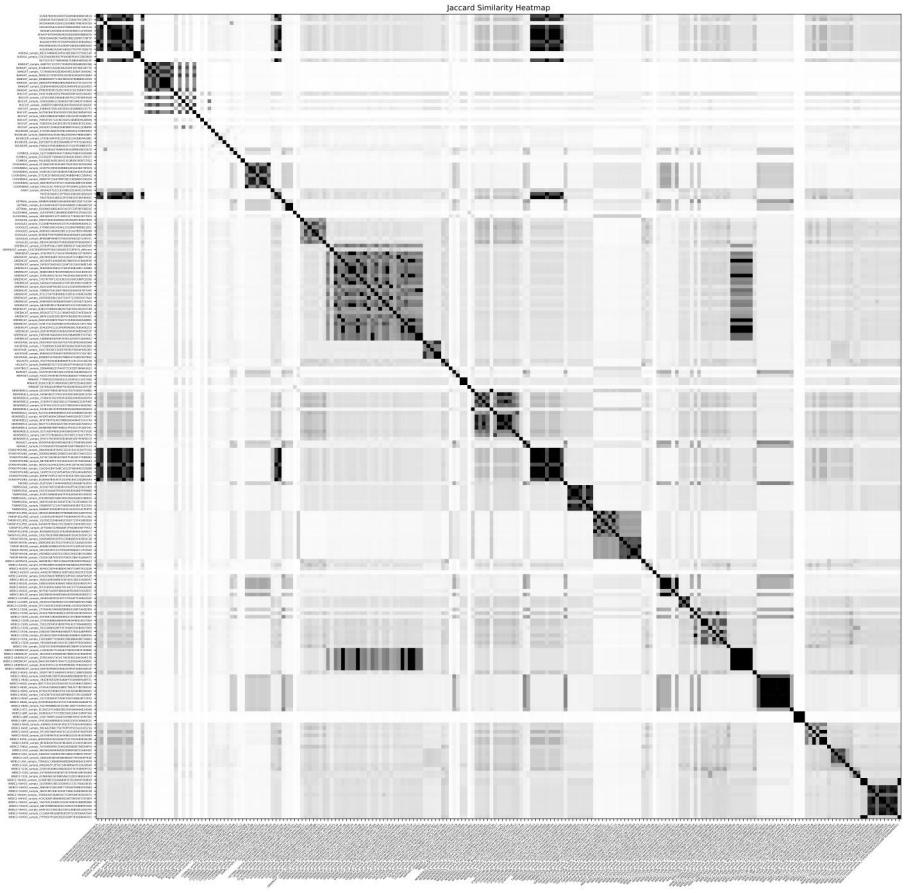

图 11 jaccard_heatmap_r.svg