

第 13 章 图算法与知识图谱

图算法广泛应用与社交网络、金融、交通、搜索等存在大量非结构化网状数据的领域，在安全领域也在风控、威胁情报方面有较多应用，是一种非常简单易用且有效的机器学习算法，算法的思想也非常容易被理解。

本章 13.1 节和 13.2 节将介绍图的基本概念，并给出基本的使用方式。

本章 13.3 节将介绍如何使用有向图识别 Webshell。

本章 13.4 节将介绍如何使用有向图识别僵尸网络中的同一黑产团体。

本章 13.5 节将结合搜索广告介绍神秘的知识图谱。

本章 13.6 节将介绍知识图谱的概念。

本章 13.7 节将介绍知识图谱的在风控领域的应用，包括检测账户被盗、撞库以及简单的刷单。

本章 13.8 节将介绍知识图谱的在威胁情报领域的应用，包括挖掘后门文件的潜在联系以及域名的潜在联系。

13.1 图算法概述

在现实世界中，有种关联关系难以用数据库的表结构来表示，比如微博的粉丝关系、偶像剧中的 N 角恋、多个域名之间的注册关系等，于是图这种古老的数据结构就排上了用场。一般认为，如果给图的每条边规定一个方向，那么得到的图称为有向图，其边也称为有向边。在有向图中，与一个节点相关联的边有出边和入边之分，而与一个有向边关联的两个点也有始点和终点之分。相反，边没有方向的图称为无向图。我们拿一个实际的例子来讲解下用途最为广泛的有向图的一些基础知识。

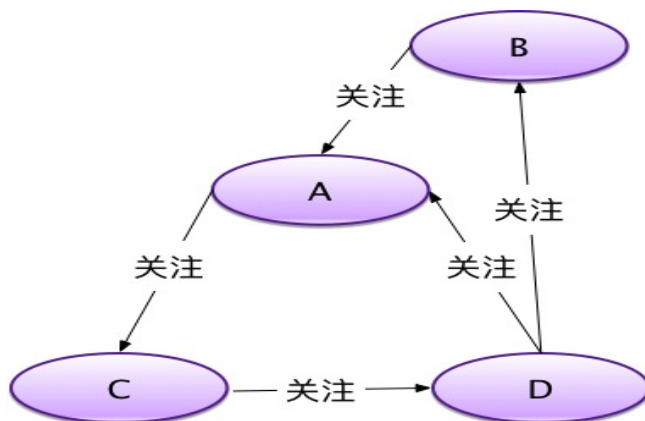


图13-1 图算法原理图

微博中的好友关注关系就是典型的有向图，因为关注是有方向性的，比如我关注了钟丽缇，但是钟丽缇不一定关注了我。假定关注关系如下描述：

- ❑ A 关注了 C
- ❑ B 关注了 A
- ❑ C 关注 D
- ❑ D 关注了 A 和 B

D 关注了一个人，所以他的出度为 1，D 被两个人关注，所以他的入度为 2。

图的聚类算法，最简单的一种实现叫做连通分支。所谓连通分支，指的是途中由边连接在一起的一组顶点，不要求顶点之间必须两两相连，但是连通分支的任意两个定点之间，至少存在一条路径，计算连通分支时不区分有向图和无向图。

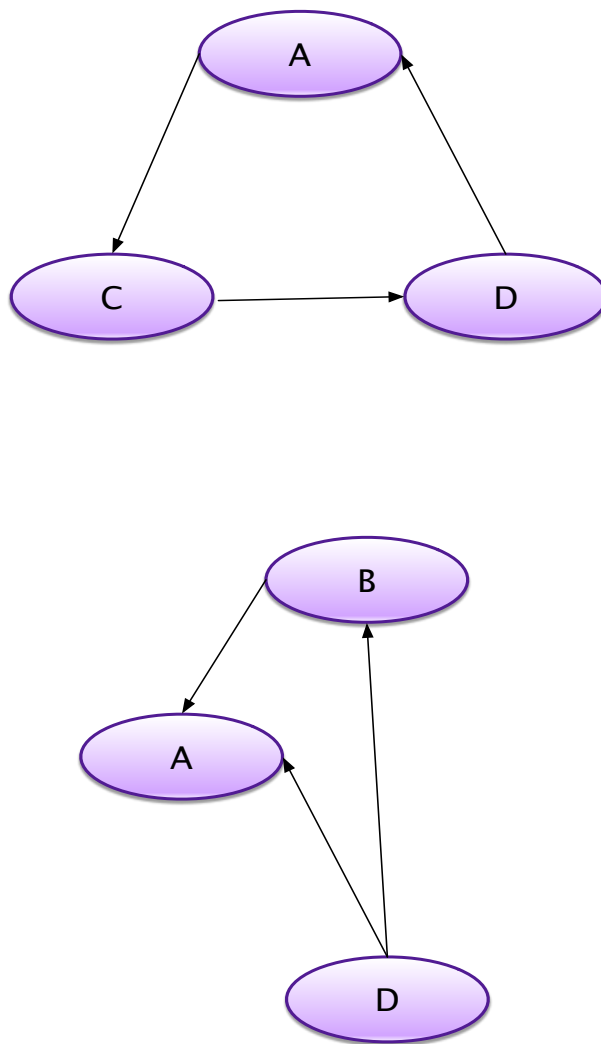


图13-2 图的连通分支原理图

13.2 示例：hello world！有向图

neo4j 是一个高性能的图形数据库，它将结构化数据存储在网络上而不是表中，因其嵌入式、高性能、轻量级等优势，越来越受到关注。我们以 neo4j 为例子，讲解下如下编写我们的第一个有向图程序。完整演示代码请见本书 Github 上的 13-1.py。

13.2.1 neo4j 安装

在 <https://neo4j.com/> 上下载安装包安装，默认配置即可

13.2.2 neo4j 启动

以我的 mac 为例子,通过 gui 启动即可,默认密码为 neo4j/neo4j,第一次登录会要求更改密码。

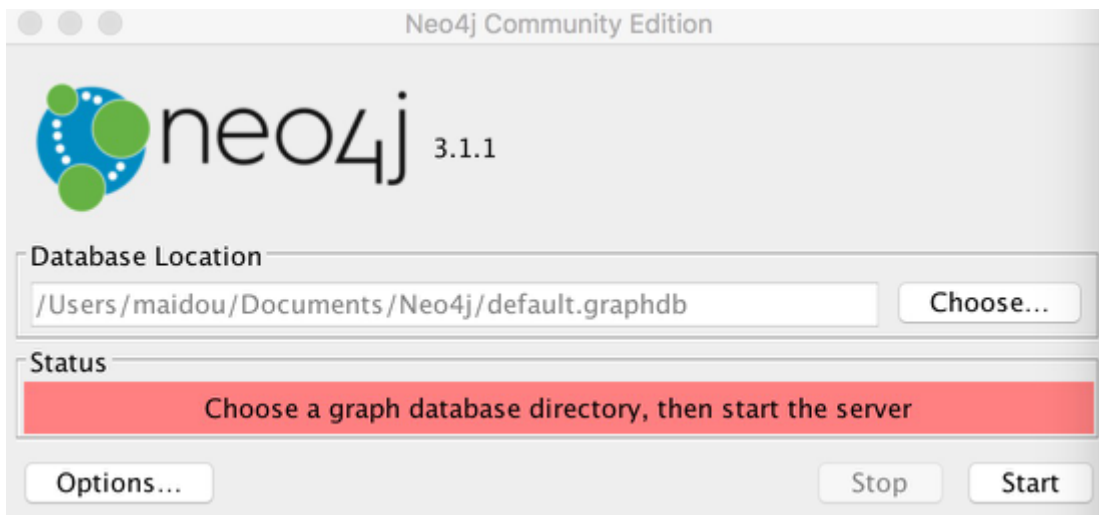


图13-3 neo4j 启动界面

启动管理界面

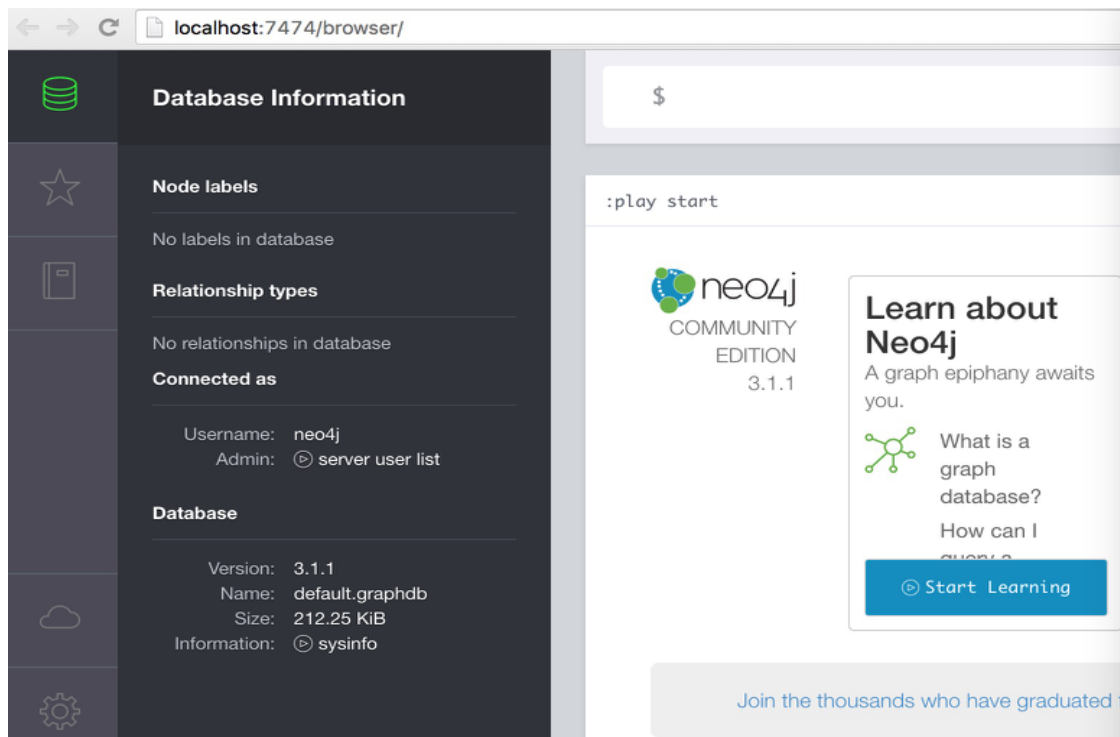


图13-4 neo4j 管理界面

13.2.3 SDK 安装

python api 库安装

```
sudo pip install neo4j-driver
```

下载 JPyype

```
https://pypi.python.org/pypi/JPyype1
```

安装 JPyype

```
tar -zxvf JPyype1-0.6.2.tar.gz  
cd JPyype1-0.6.2  
sudo python setup.py install
```

13.2.4 导入数据

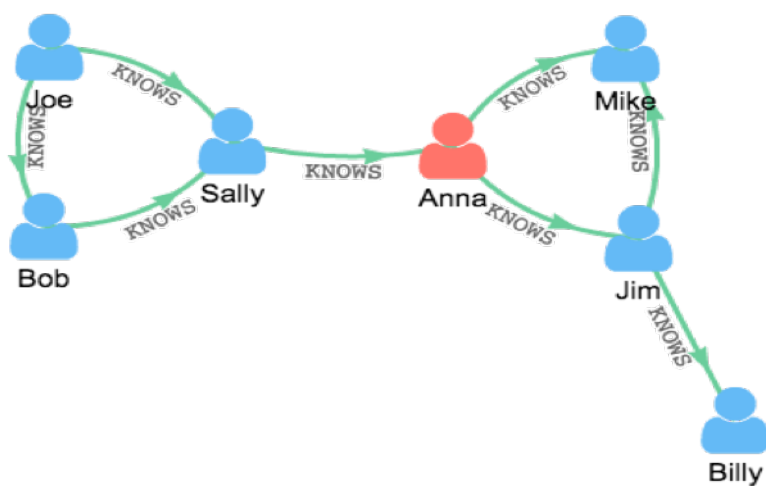


图13-5 微博关注关系图

假设要将上面的微博好有关系导入数据库。

导入库并连接数据库：

```
from neo4j.v1 import GraphDatabase, basic_auth

driver = GraphDatabase.driver("bolt://localhost", auth=basic_auth("neo4j",
"maidou"))

session = driver.session()
```

插入人物结点以及关注关系数据：

```
insert_query = '''

UNWIND {pairs} as pair

MERGE (p1:Person {name:pair[0]})

MERGE (p2:Person {name:pair[1]})

MERGE (p1)-[:KNOWS]-(p2);

'''

data = [
    ["Jim", "Mike"], ["Jim", "Billy"], ["Anna", "Jim"],
    ["Anna", "Mike"], ["Sally", "Anna"], ["Joe", "Sally"],
    ["Joe", "Bob"], ["Bob", "Sally"]]

session.run(insert_query, parameters={"pairs": data})
```

13.2.5 查询数据

查询 1：

```
foaf_query = '''

MATCH (person:Person)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)

WHERE person.name = {name}

AND NOT (person)-[:KNOWS]-(foaf)

RETURN foaf.name AS name

'''

results = session.run(foaf_query, parameters={"name": "Joe"})
```

```
for record in results:

    print(record["name"])
```

运行结果为:

```
B0000000B60544:code liu.yan$ python 13-1.py

Anna
```

查询 2:

```
common_friends_query = """

MATCH (user:Person)-[:KNOWS]-(friend)-[:KNOWS]-(foaf:Person)

WHERE user.name = {user} AND foaf.name = {foaf}

RETURN friend.name AS friend

"""

results = session.run(common_friends_query, parameters={"user": "Joe", "foaf":
"Sally"})

for record in results:

    print(record["friend"])
```

查询结果为:

```
B0000000B60544:code liu.yan$ python 13-1.py

Bob
```

查询 3:

```
connecting_paths_query = """

MATCH path = shortestPath((p1:Person)-[:KNOWS*..6]-(p2:Person))

WHERE p1.name = {name1} AND p2.name = {name2}

RETURN path

"""

results = session.run(connecting_paths_query, parameters={"name1": "Joe",
"name2": "Billy"})
```

```
for record in results:  
    print (record["path"])
```

结果为:

```
B0000000B60544:code liu.yan$ python 13-1.py  
  
<Path start=7 end=16 size=4>
```


13.2.6 可视化展现

1 展示原始数据

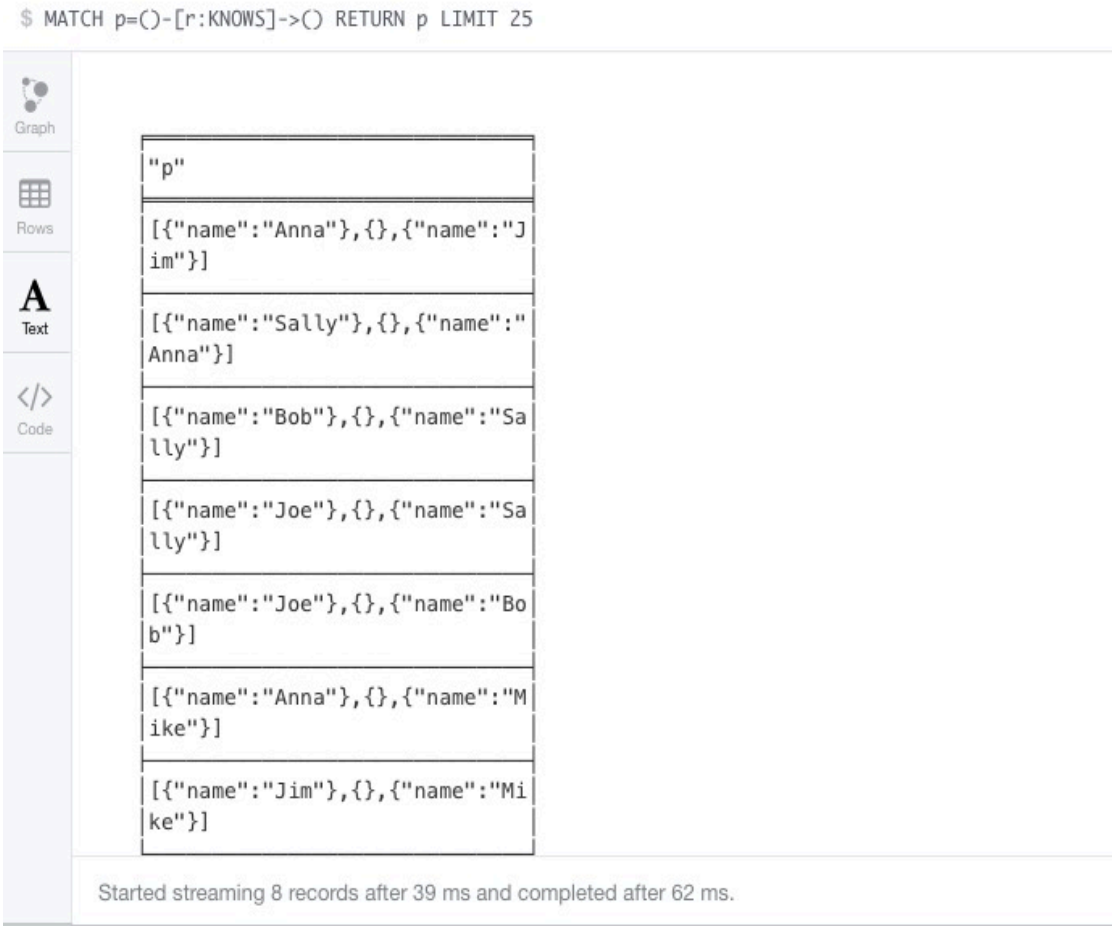


图13-6 neo4j 原始数据图

2 展示关联关系

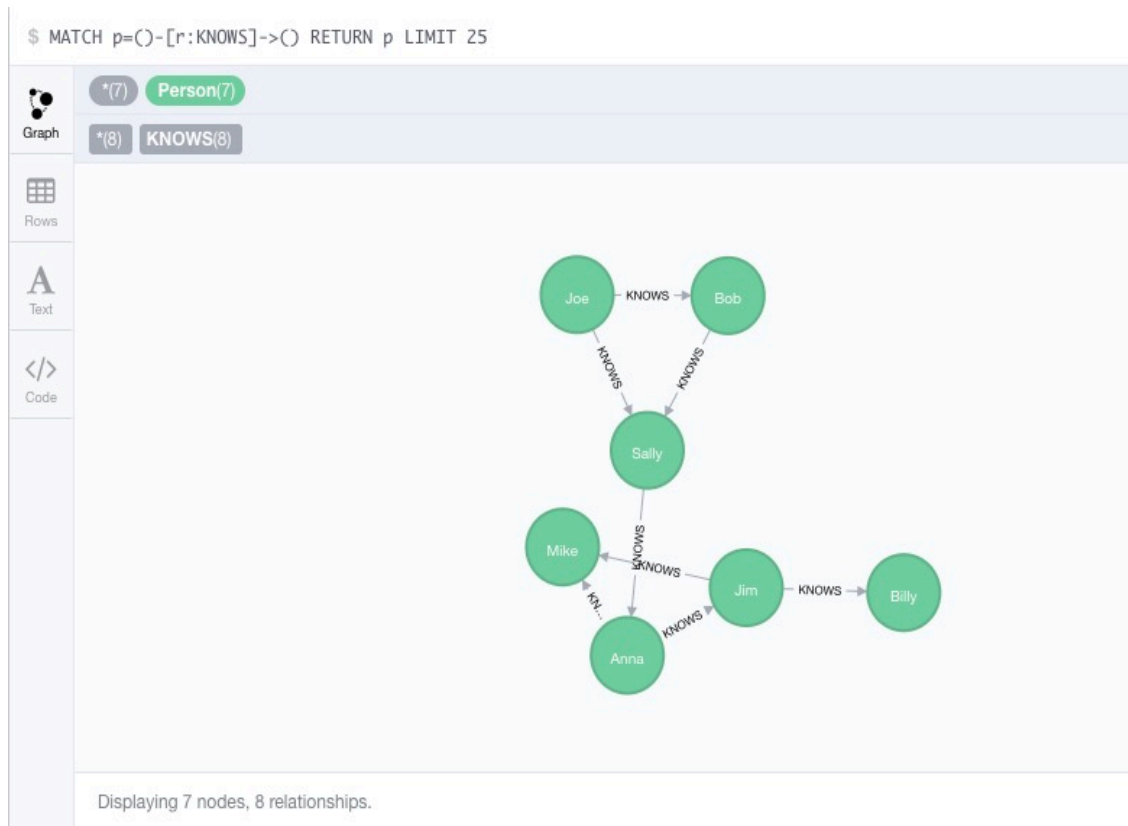


图13-7 neo4j 可视化图

13.3 示例：有向图识别 Webshell

完整演示代码请见本书 Github 上的 13-2.py 以及 13-3.py。

Webshell 具有很多访问特征，其中和有向图有关的为：

- ☐ 入度出度均为 0
- ☐ 入度出度均为 1 且自己指向自己

完整处理流程为：

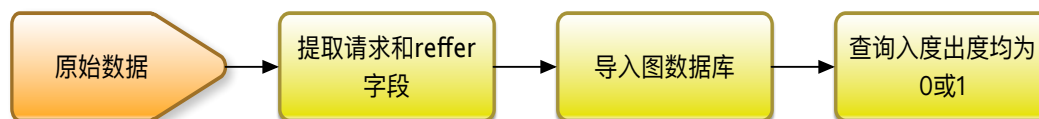
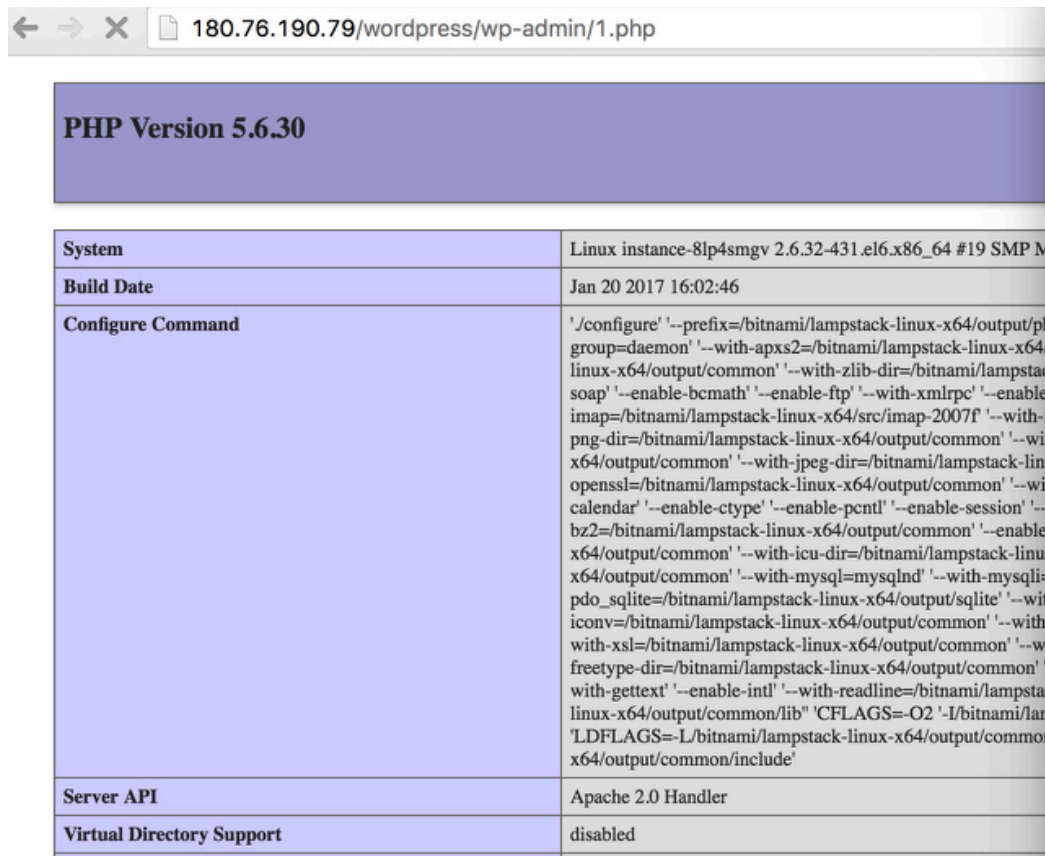


图13-8 http 日志数据处理流程

13.3.1 数据整理

我们在新安装的 wordpress 网站下面安装一个简单的后门 1.php，内容为 phpinfo。



PHP Version 5.6.30	
System	Linux instance-8lp4smgv 2.6.32-431.el6.x86_64 #19 SMP M
Build Date	Jan 20 2017 16:02:46
Configure Command	'./configure' '--prefix=/bitnami/lampstack-linux-x64/output/pl group=daemon' '--with-apxs2=/bitnami/lampstack-linux-x64 linux-x64/output/common' '--with-zlib-dir=/bitnami/lampsta soap' '--enable-bcmath' '--enable-ftp' '--with-xmlrpc' '--enable imap=/bitnami/lampstack-linux-x64/src/imap-2007f' '--with- png-dir=/bitnami/lampstack-linux-x64/output/common' '--wi x64/output/common' '--with-jpeg-dir=/bitnami/lampstack-lin openssl=/bitnami/lampstack-linux-x64/output/common' '--wi calendar' '--enable-ctype' '--enable-pcntl' '--enable-session' '-- bz2=/bitnami/lampstack-linux-x64/output/common' '--enable x64/output/common' '--with-icu-dir=/bitnami/lampstack-linu x64/output/common' '--with-mysql=mysqlnd' '--with-mysqli pdo_sqlite=/bitnami/lampstack-linux-x64/output/sqlite' '--wit iconv=/bitnami/lampstack-linux-x64/output/common' '--with with-xsl=/bitnami/lampstack-linux-x64/output/common' '--w freetype-dir=/bitnami/lampstack-linux-x64/output/common' ' with-gettext' '--enable-intl' '--with-readline=/bitnami/lampsta linux-x64/output/common/lib' 'CFLAGS=-O2' '-I/bitnami/lar ' 'LDFLAGS=-L/bitnami/lampstack-linux-x64/output/commo x64/output/common/include'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled

图13-9 测试环境中放置的后门文件

apache 默认不记录 refer 字段，需要修改默认配置，开启 httpd 自定义日志格式，记录 User-Agent 以及 Referer:

```
<IfModule logio_module>

# You need to enable mod_logio.c to use %I and %O

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O"
combinedio

</IfModule>

CustomLog "logs/access_log" combined
```

针对 1.php 的访问日志为:

```
[root@instance-8lp4smgv logs]# cat access_log | grep 'wp-admin/1.php'
```

```
125.33.206.140 - - [26/Feb/2017:13:09:47 +0800] "GET
/wordpress/wp-admin/1.php HTTP/1.1" 200 17 "-" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102
Safari/537.36"

125.33.206.140 - - [26/Feb/2017:13:11:19 +0800] "GET
/wordpress/wp-admin/1.php HTTP/1.1" 200 17 "-" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102
Safari/537.36"
```

逐行处理日志，生成请求和的对应关系，聚合结果举例为：

```
- -> http://180.76.190.79/wordpress/wp-admin/1.php
- -> http://180.76.190.79/wordpress/wp-admin/admin-ajax.php
- -> http://180.76.190.79/wordpress/wp-admin/customize.php
- -> http://180.76.190.79/wordpress/wp-admin/load-styles.php
- -> http://180.76.190.79/wordpress/wp-admin/post-new.php
- -> http://180.76.190.79/wordpress/wp-login.php

http://180.76.190.79/wordpress/ ->
http://180.76.190.79/wordpress/wp-admin/edit-comments.php

http://180.76.190.79/wordpress/ ->
http://180.76.190.79/wordpress/wp-admin/profile.php

http://180.76.190.79/wordpress/ ->
http://180.76.190.79/wordpress/wp-login.php

http://180.76.190.79/wordpress/ ->
http://180.76.190.79/wordpress/xmlrpc.php

http://180.76.190.79/wordpress/wp-admin/ ->
http://180.76.190.79/wordpress/wp
```

13.3.2 数据导入

连接数据库

```
driver =  
  
GraphDatabase.driver("bolt://localhost:7687",auth=basic_auth("neo4j","maidou"  
))  
  
session = driver.session()
```

逐行读取，生成结点以及关联关系：

```
for line in file_object:  
  
    matchObj = re.match( r'(\S+) -> (\S+)', line, re.M|re.I)  
  
if matchObj:  
  
    path = matchObj.group(1);  
  
    ref = matchObj.group(2);  
  
if path in nodes.keys():  
  
    path_node = nodes[path]  
  
else:  
  
    path_node = "Page%d" % index  
  
    nodes[path]=path_node  
  
sql = "create (%s:Page {url:\"%s\" ,  
id:\"%d\",in:0,out:0})" %(path_node,path,index)  
  
index=index+1  
  
session.run(sql)
```

把入度出度作为结点的属性，更新结点的出度入度属性：

```
if ref in nodes.keys():  
  
    ref_node = nodes[ref]  
  
else:  
  
    ref_node = "Page%d" % index  
  
    nodes[ref]=ref_node  
  
sql = "create (%s:Page {url:\"%s\",id:\"%d\",in:0,out:0})" %(ref_node,ref,index)  
  
index=index+1  
  
session.run(sql)
```

```

sql = "create (%s)-[:IN]->(%s)" % (path_node, ref_node)

session.run(sql)

sql = "match (n:Page {url:\"%s\"}) SET n.out=n.out+1" % path

session.run(sql)

sql = "match (n:Page {url:\"%s\"}) SET n.in=n.in+1" % ref

session.run(sql)

```

13.3.3 查询结果

The screenshot shows the Neo4j web interface. On the left, the 'Database Information' sidebar displays details about the 'default.graphdb' database, including its version (3.1.1), size (674.09 KiB), and the user 'neo4j' is connected as 'server user list'. The main panel shows a Cypher query: `$ MATCH (n:Page) RETURN n LIMIT 25`. The results are displayed in a table with 6 rows, each containing a JSON object representing a Page node with its 'in', 'id', 'url', and 'out' properties.

id	in	url	out
1	0	-	0
2	0	http://180.76.190.79/wordpress/wp-admin/1.php	0
3	0	http://180.76.190.79/wordpress/wp-admin/admin-ajax.php	0
4	0	http://180.76.190.79/wordpress/wp-admin/customize.php	0
5	0	http://180.76.190.79/wordpress/wp-admin/load-styles.php	0
6	0	http://180.76.190.79/wordpress/wp-	0

图13-10 网页关联关系原始数据

可视化结果为:

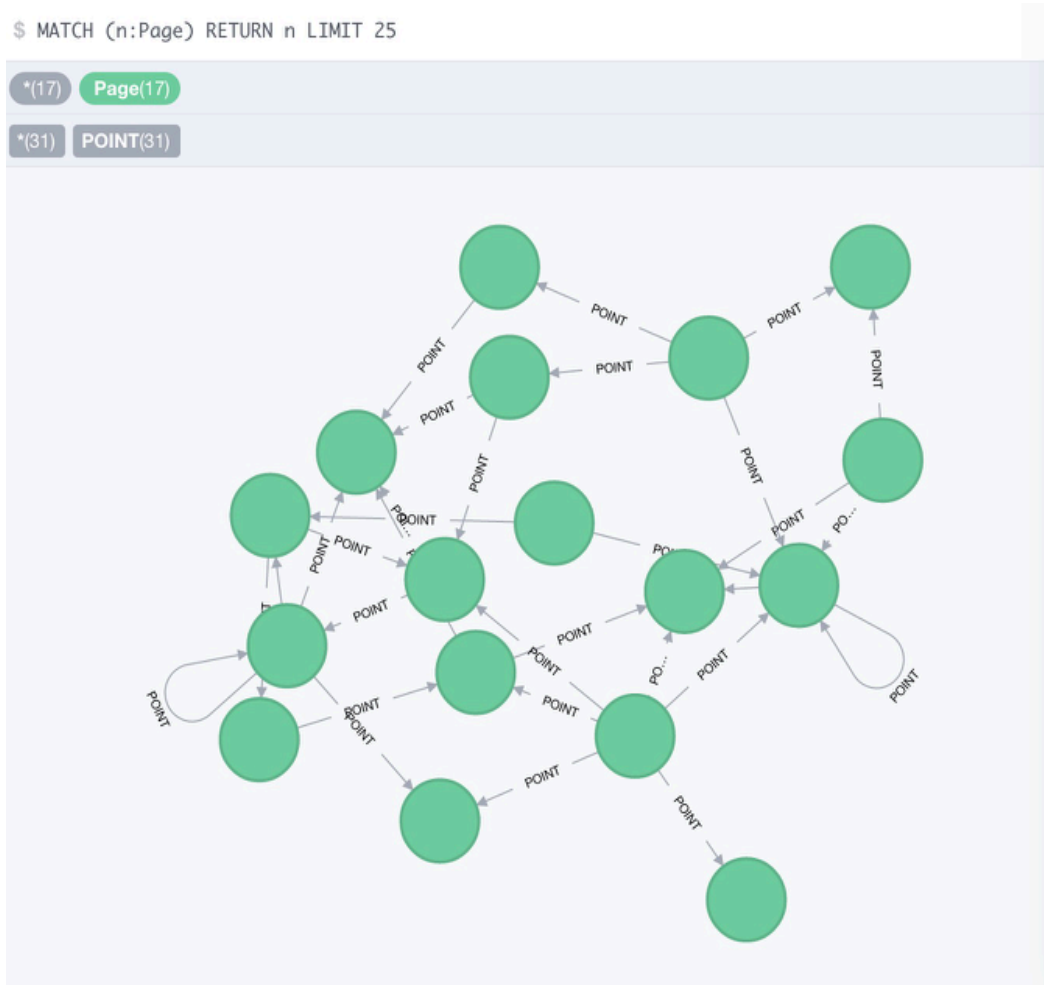


图13-11 网页关联关系可视化图

查询入度为 1 出度均为 0 的结点或者查询入度出度均为 1 且指向自己的结点，由于把 ref 为空的情况也识别为“-”结点，所以入度为 1 出度均为 0。查询满足条件的疑似 Webshell 链接：

```
$ match (n:Page) where ( n.in=1 and n.out=0 ) or ( n.in=1 and n.out=1 ) return n.url
```

	n.url
Rows	http://180.76.190.79/wordpress/wp-admin/1.php
Text	http://180.76.190.79/wordpress/wp-admin/profile.php
Code	http://180.76.190.79/wordpress/wp-admin/index.php

图13-12 查询疑似 Webshell 的链接

生产环境实际使用中，我们遇到误报分为以下几种：

- ❑ 主页，各种 index 页面

- ❑ phpmyadmin、zabbix 等运维管理后台
- ❑ hadoop、elk 等开源软件的控制台
- ❑ api 接口

这些通过短期加白可以有效解决，比较麻烦的是扫描器对结果的影响，这部分需要通过扫描器指纹或者使用高大上的人机算法来去掉干扰。

13.4 示例：有向图识别僵尸网络

黑产团伙通过控制僵尸网络，主要达到以下几个目的：

- ❑ 发送广告邮件
- ❑ 发起 DDoS 攻击
- ❑ 发起恶意广告点击展现请求
- ❑ 全网扫描漏洞并自动化渗透具有漏洞的主机，不断扩充肉鸡规模

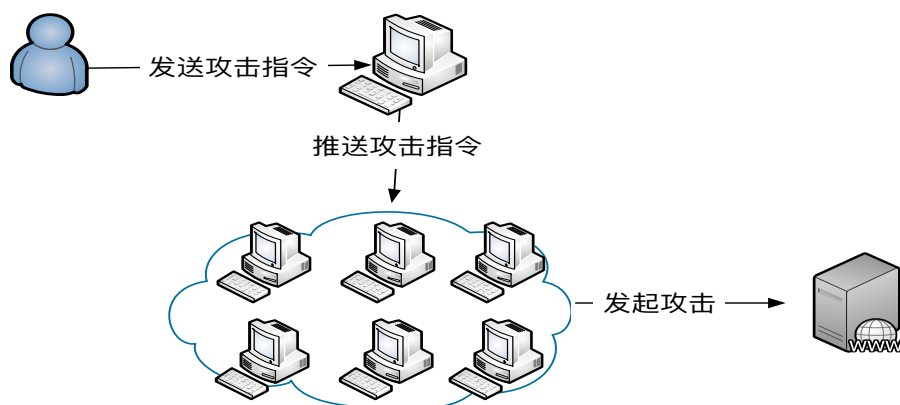


图13-13 黑产控制僵尸网络发起攻击示意图

黑产通常会对整个僵尸网络发布相同的控制指令，所以通过统计一段时间内攻击源 IP 和被攻击者域名之间的关联关系，可以初步确定哪些 ip 可能是被同一个黑产团体控制。

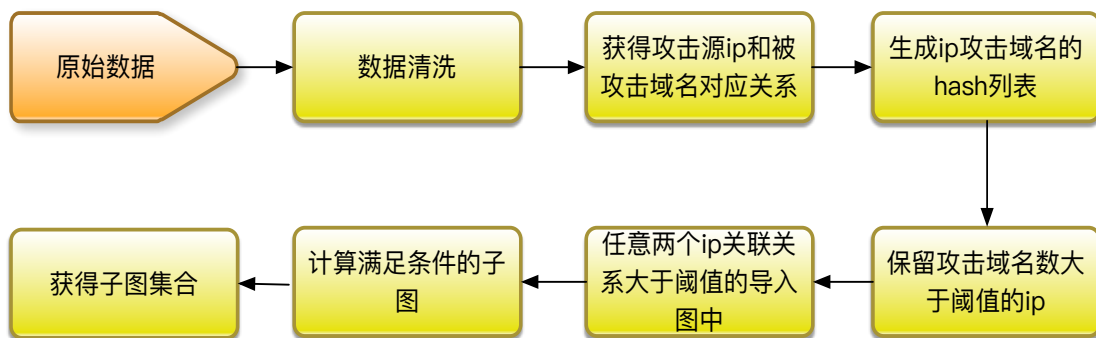


图13-14 ip 与被攻击域名处理流程图

完整演示代码请见本书 Github 上的 13-4.py。

13.4.1 数据整理

整理开源攻击数据网站 <http://www.secrepo.com/> 上的数据并整理成训练集合，格式如下：

```
97.74.144.16* *nggui.cn
97.74.144.16* *lfinancetips.*
97.74.144.16* *zc.*
97.74.237.19* *000.*
98.150.210.24* *qi-steel-shot.*
98.21.4.24* *dapower.*
```

其中 IP 地址和域名做了脱敏处理。

脱敏函数实现为：

```
with open(filename) as f:

    for line in f:

        line=line.strip('\n')

        ip, domain=line.split()

        ip=re.sub(r'\d$', '*', ip)

        domain= re.sub(r'\w{3}$', '*', domain)

        domain = re.sub(r'^\w{3}', '*', domain)

        print "%s %s" % (ip, domain)
```

13.4.2 数据导入

逐行读取攻击数据，按照攻击源 ip 建立 hash 表，hash 表的键值为被攻击的域名。

```
with open(filename) as f:

    for line in f:

        (ip, domain)=line.split(" ")

        if not ip=="0.0.0.0":
```

```

        if not iplist.has_key(ip):

            iplist[ip]={}

            iplist[ip][domain]=1

```

定义阈值 R，攻击的域名超过 R 的 ip 才列入统计范围。

```

for ip in iplist.keys():

    if len(iplist[ip]) >= R:

        goodiplist[ip]=1

```

定义计算 jarccard 系数的函数，作为衡量两个 ip 攻击集合相似度的方式。所谓 jarccard 系数，指的是两个集合交集除以并集的比值。定义阈值 N，当两个 ip 攻击的域名 jarccard 大于等于 N 时才列入统计范围。

```

#jarccard 系数
def get_len(d1,d2):

    ds1=set()

    for d in d1.keys():

        ds1.add(d)

    ds2=set()

    for d in d2.keys():

        ds2.add(d)

    return len(ds1&ds2)/len(ds1|ds2)

```

满足阈值的 ip 导入图数据库。

```

for ip1 in iplist.keys():

    for ip2 in iplist.keys():

        if not ip1 == ip2 :

            weight=get_len(iplist[ip1],iplist[ip2])

            if (weight >= N) and (ip1 in goodiplist.keys()) and (ip2 in
goodiplist.keys()):

                #点不存在会自动添加

                G.add_edge(ip1,ip2,weight=weight)

```

13.4.3 查询分析

定义阈值 M，当同一团伙的 ip 大于等于 M 时才显示结果。

```
n_sub_graphs=nx.number_connected_components(G)

sub_graphs=nx.connected_component_subgraphs(G)

for i,sub_graph in enumerate(sub_graphs):

    n_nodes=len(sub_graph.nodes())

    if n_nodes >= M:

        print("Subgraph {0} has {1} nodes

{2}".format(i,n_nodes,sub_graph.nodes()))
```

定义全局阈值，包括相似度、黑客团伙 ip 最小个数等：

```
#相似度

N=0.5

#黑客团伙 IP 最少个数

M=3

#黑客 IP 攻击目标最小个数

R=2
```

结果举例为：

```
Subgraph 2 has 20 nodes ['58.219.226.23*', '58.208.34.6*', '58.255.121.17*',
'60.220.65.4*', '61.188.148.12*', '59.56.44.*', '58.241.12.14*', '58.255.125.22*',
'60.161.88.13*', '59.55.101.24*', '59.63.58.*', '60.168.24.11*', '60.206.64.11*',
'59.59.165.22*', '58.209.126.20*', '58.52.199.*', '61.180.116.22*',
'59.56.126.3*', '61.154.37.20*', '60.186.111.3*']

Subgraph 3 has 3 nodes ['58.49.86.23*', '60.22.103.2*', '59.53.67.20*']

Subgraph 5 has 5 nodes ['58.217.185.12*', '59.63.28.17*', '59.47.7.11*',
'59.47.7.12*', '59.63.248.4*']
```

可视化聚类结果为：

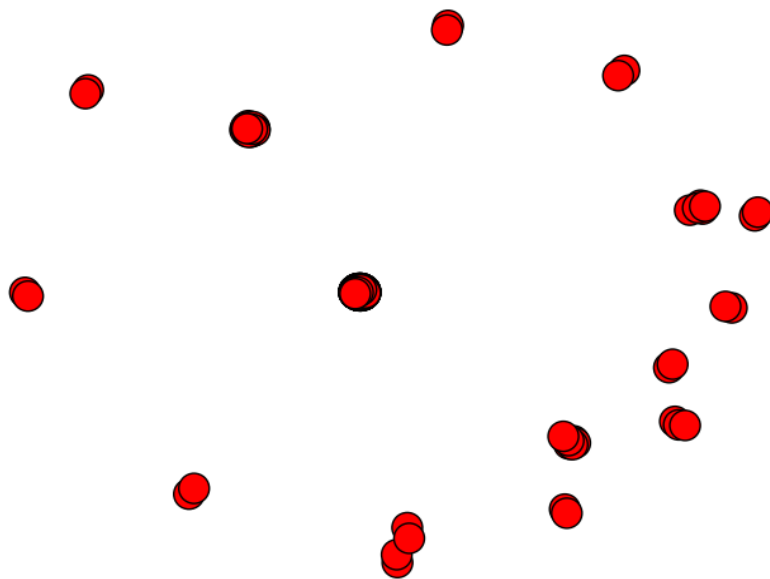


图13-15 僵尸网络聚类效果图

13.5 生活中的知识图谱

当你在百度搜索“孙悟空的师傅”时，会直接展现出唐僧和菩提老祖的百度百科介绍。

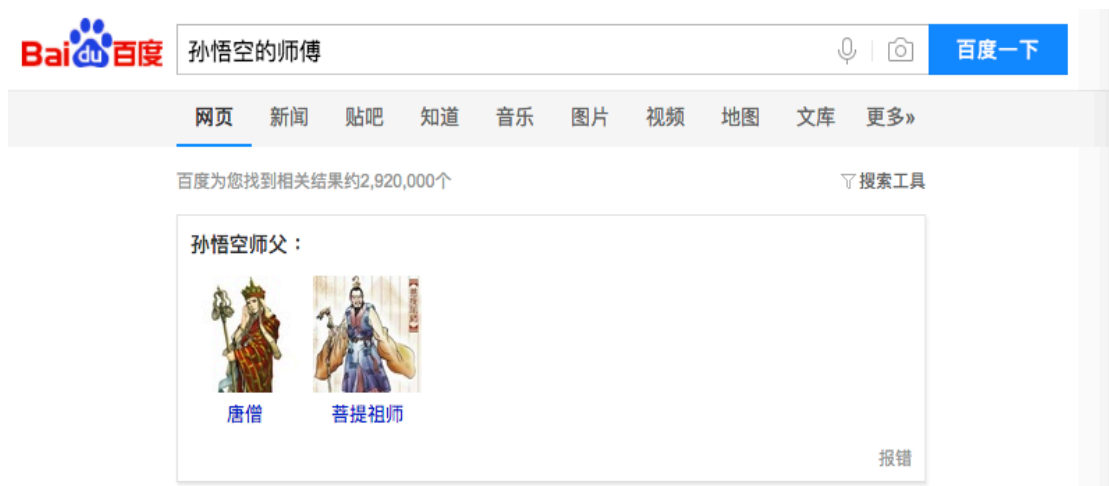


图13-16 百度搜索孙悟空的师傅

这是因为百度搜索通过知识图谱建立实体之间的属性与关系，让搜索引擎更懂用户的意图，直接解答用户的疑惑。

当你拿些手机搜索”全聚德”时，会自动展现你附近的全聚德店的位置。



图13-17 百度搜索全聚德

这是因为百度搜索借助于知识图谱，结合用户行为信息，为用户提供更符合当前场景的搜索结果。

当你在百度搜索”达芬奇”时，除了会展现达芬奇相关的信息，同时也会自动展现关注达芬奇的人同时也关注的其他任务以及作品。



图13-18 百度搜索达芬奇

这是因为百度搜索通过知识图谱建立事物之间的关联，扩展用户搜索结果，发现更多内容。

13.6 知识图谱概述

2012 年，Google 在其官方博客中宣称：为了让用户能够更快更简单的发现新的信息和知识，Google 搜索将发布“知识图谱”（Knowledge Graph）——可以将搜索结果进行知识系统化，任何一个关键词都能获得完整的知识体系。比如搜索“Amazon”（亚马逊河），一般的搜索结果会给出和 Amazon 最相关的信息。比如 Amazon 网站，因为网上关于它的信息最多，但 Amazon 并不仅仅是一个网站，它还是全球流量最大的 Amazon 河流。如果在追溯历史，它可能还是希腊女战士一族的代称。而这些结果未来都会 Google 搜索的“知识图谱”中展现出来。Google 的“知识图谱”不仅仅会从 Freebase、维基百科或全球概览中获得专业的信息，同时还通过大规模的信息搜索分析来提高搜索结果的深度和广度。现在 Google 数据库中包含超过 5 亿个事物，不同事物之间的关系超过 35 亿条。同时人们搜索的越多，Google 获得的信息也就越多越全面，整个知识图谱也就会达到更好的效果。

知识图谱本质上可以认为是图的一种具体应用，它大量集成了互联网上的各类数据，从而进一步挖掘出了数据的潜在联系与价值。在安全领域应用知识图谱，可以挖掘出数据之间潜在的联系，结合这些潜在的联系可以大大扩展我们的数据分析思路。

13.7 示例：知识图谱在风控领域的应用

传统的风控策略主要基于模型和策略，这个在传统行业已然有着很大的市场，但是在互联网环境下，职业黑产团伙以及羊毛党的加入，传统的风控策略已经大打折扣。如何利用互联网上各类数据，挖掘蛛丝马迹，进行风控呢？知识图谱，作为关系的直接表示方式，可以很好地解决这两个问题。首先，知识图谱提供非常便捷的方式来添加新的数据源。其次，知识图谱本身就是用来表示关系的，这种直观的表达方法可以帮助我们更有效地分析复杂关系中存在的特定的潜在风险。下面我们结合几个典型场景介绍下知识图谱在风控领域的应用。完整演示代码请见本书 Github 上的 13-5.py。

13.7.1 检测疑似帐号被盗

盗号在互联网业务中常见的恶意行为，盗号的渠道也非常多，从暴力破解到撞库都可能导致账户被盗，单纯依赖用户名和密码认证已经无法保障账户的安全。我们使用脱敏的测试数据来演示下疑似账户被盗的情况。测试样本记录了微软的邮件系统的手机客户端成功登录日志。

```
uid=mike,ip=ip1,tel=tel1,activesyncid=1
uid=mike,ip=ip2,tel=tel1,activesyncid=2
uid=mike,ip=ip3,tel=tel1,activesyncid=2
uid=john,ip=ip1,tel=tel2,activesyncid=2
uid=john,ip=ip4,tel=tel2,activesyncid=2
uid=john,ip=ip5,tel=tel2,activesyncid=2
```

字段含义分别为：

- ❑ uid，用户名
- ❑ ip，登录 ip 地址
- ❑ tel，安装微软邮件客户端的手机的手机号
- ❑ activesyncid，安装微软邮件客户端的手机对应 activesyncid，该 id 全局唯一，与硬件绑定，类似微软在 pc 上的 guid

逻辑上对应的拓扑图为：

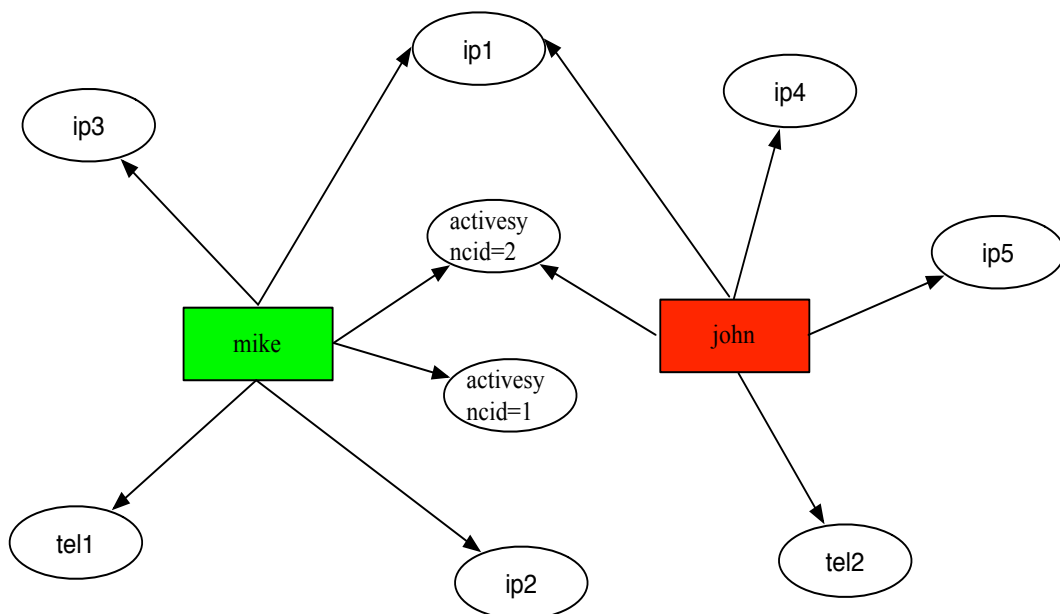


图13-19 疑似账户被盗示意图

从拓扑图可以看出，activesyncid 为 2 的硬件登录了 mike 和 john 两个账户，mike 历史上登录成功的包括 activesyncid 为 1 的硬件以及 activesyncid 为 2 的硬件，初步判定 activesyncid 为 2 的硬件盗取了 mike 的账户登录。

逐行处理样本文件，获取对应的 uid,ip,tel,activesyncid

```
with open("../data/KnowledgeGraph/sample1.txt") as f:

    G = nx.Graph()

    for line in f:

        line=line.strip('\n')

        uid,ip,tel,activesyncid=line.split(',')

```

以 uid 为中心，添加对应的 ip,tel,activesyncid 结点：

```
G.add_edge(uid, ip)

G.add_edge(uid, tel)

G.add_edge(uid, activesyncid)

```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)

plt.show()

```

对应知识图谱为：

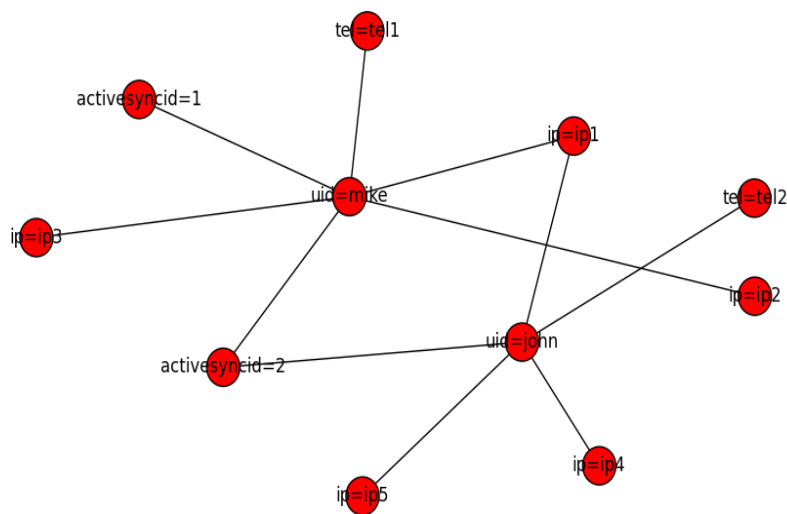


图13-20 疑似账户被盗知识图谱

13.7.2 检测疑似撞库攻击

撞库是黑客通过收集互联网已泄露的用户和密码信息，生成对应的字典表，尝试批量登陆其他网站后，得到一系列可以登录的用户。很多用户在不同网站使用的是相同的帐号密码，因此黑客可以通过获取用户在 A 网站的账户从而尝试登录 B 网址，这就可以理解为撞库攻击。2014 年 12 月 25 日，12306 网站用户信息在互联网上疯传。对此，12306 官方网站称，网上泄露的用户信息系经其他网站或渠道流出。据悉，此次泄露的用户数据不少于 131653 条。该批数据基本确认为黑客通过“撞库攻击”所获得。我们使用脱敏的测试数据来演示下疑似撞库攻击的情况。测试样本记录了微软的邮件系统的网页版登录日志，其中即包含登录成功也包含登录失败的情况。

```
uid=mike,ip=ip1,login=yes,ua=ua1
uid=mike,ip=ip1,login=no,ua=ua1
uid=lily,ip=ip1,login=yes,ua=ua1
uid=willy,ip=ip1,login=no,ua=ua1
uid=tony,ip=ip1,login=yes,ua=ua1
uid=charly,ip=ip1,login=yes,ua=ua1
uid=steven,ip=ip1,login=no,ua=ua1
uid=mery,ip=ip1,login=no,ua=ua1
uid=john,ip=ip1,login=no,ua=ua1
```

字段含义分别为：

- ❑ uid，用户名
- ❑ ip，登录 ip 地址
- ❑ login，登录状态，yes 表明登录成功，no 表明登录失败
- ❑ ua，浏览器的 ua 字段，通常集合 ip 和 ua 字段可以一定程度标识一个用户或者设备

逻辑上对应的拓扑图为：

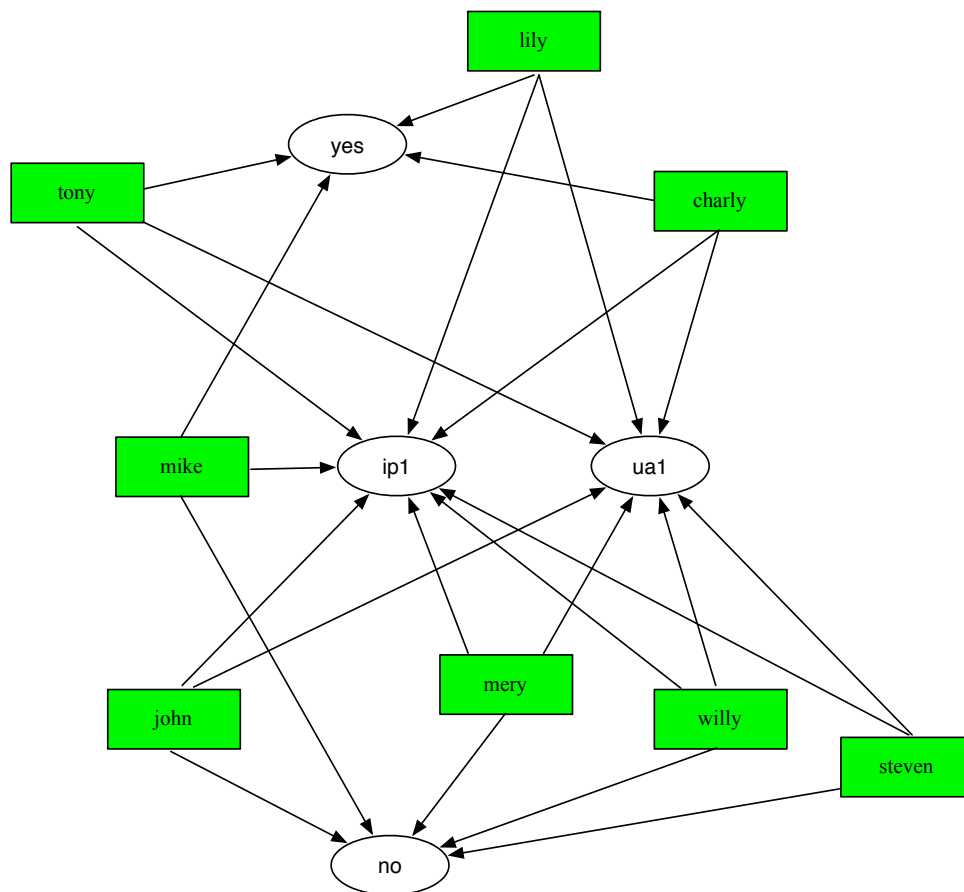


图13-21 疑似拖库攻击示意图

从拓扑图可以看出来，大量账户从 ip1 登录，并且 ua 字段相同，登录失败和成功的情况均存在，疑似发生了撞库攻击行为。通常情况下，同一 ip 不会出现大量登录行为，即使办公网出口这种人员密集的地方，也应该主要是登录成功为主，不应该登录成功与失败数量均比较大。

逐行处理样本文件，获取对应的 uid,ip,login,ua 字段：

```
with open("../data/KnowledgeGraph/sample2.txt") as f:

    G = nx.Graph()

    for line in f:

        line=line.strip('\n')

        uid,ip,login,ua=line.split(',')

```

以 uid 为中心，添加对应的 ip,login,ua 结点：

```
G.add_edge(uid, ip)

G.add_edge(uid, login)

G.add_edge(uid, ua)

```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)

plt.show()
```

对应知识图谱为：

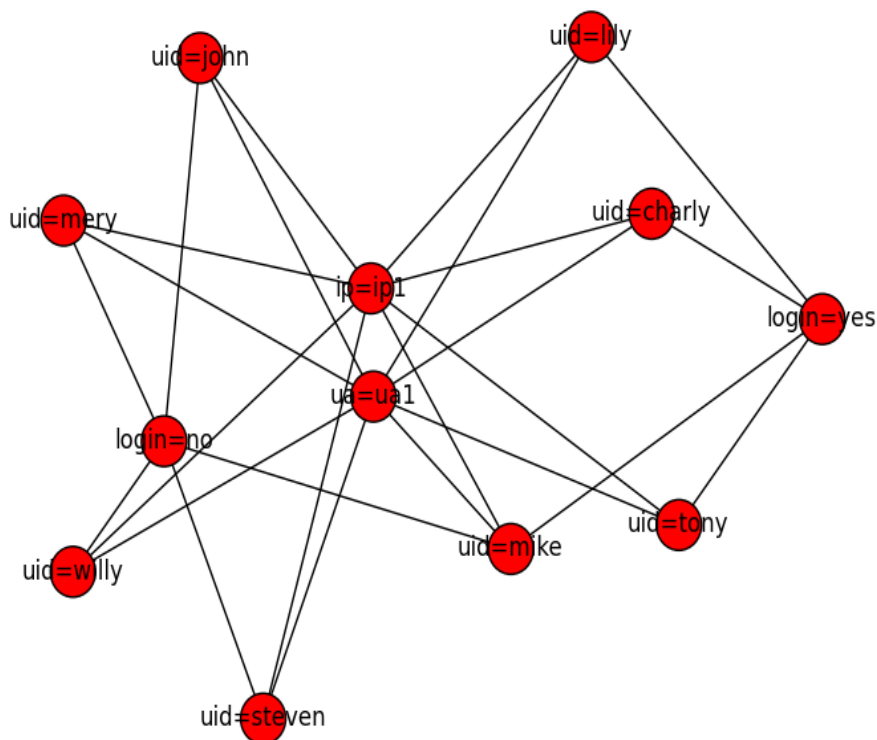


图13-22 疑似撞库攻击知识图谱

13.7.3 检测疑似刷单

国内 O2O 的大肆补贴，滋生了一个产业链，即所谓的羊毛党和刷单。羊毛党：以 80 后为代表的白领，对搜集各大电子商城、银行、实体店等各渠道的优惠促销活动、免费业务之类的信息产生了浓厚的兴趣。他们有选择地参与活动，从而以相对较低成本甚至零成本换取物质上的实惠。这一行为被称为薅羊毛，而关注与热衷于薅羊毛的群体就被称作羊毛党。早前，“羊毛党”们主要活跃在 O2O 平台或电商平台。另外随着 2015 年互联网金融的发展，一些网贷平台为吸引投资者常推出一些收益丰厚的活动，如注册认证奖励、充值返现、投标返利等，催生了以此寄生的投资群体，他们也被称为 P2P 羊毛党。这部分用户与网购羊毛党不同，只关注互联网金融产品。

刷单是店家付款请人假扮顾客,用以假乱真的购物方式提高网店的排名和销量获取销量及好评吸引顾客。刷单,一般是由买家提供购买费用,帮指定的网店卖家购买商品提高销量和信用度,并填写虚假好评的行为。通过这种方式,网店可以获得较好的搜索排名,比如,在平台搜索时“按销量”搜索,该店铺因为销量大(即便是虚假的)会更容易被买家找到。一般可分为单品刷销量为做爆款等做准备和刷信誉以提高店铺整体信誉度两种。还有一种刷单更为直接,以外卖软件为例子,同一个人使用两台手机,分别安装客户下单软件和商家接单软件,一下一接,捞取补贴。我们使用脱敏的测试数据来演示下疑似刷单的情况。

测试样本分为两部分,一部分记录了设备指纹对应安装的 app 以及 app 的登录名:

```
hid=1,uid=mike,app=appl
hid=2,uid=tony,app=appl
hid=1,uid=john,app=app2
hid=2,uid=john,app=app2
```

字段含义分别为:

- ❑ hid, 硬件指纹, 唯一标识一台设备
- ❑ uid, app 的登录用户名
- ❑ app, app 的名称

一部分记录了 app 的登录名以及下单接单情况:

```
hid=1,uid=mike,action=buy
hid=2,uid=tony,action=sell
```

字段含义分别为:

- ❑ hid, 硬件指纹, 唯一标识一台设备
- ❑ uid, app 的登录用户名
- ❑ action, 用户行为, 是下单还是接单

逻辑上对应的拓扑图为:

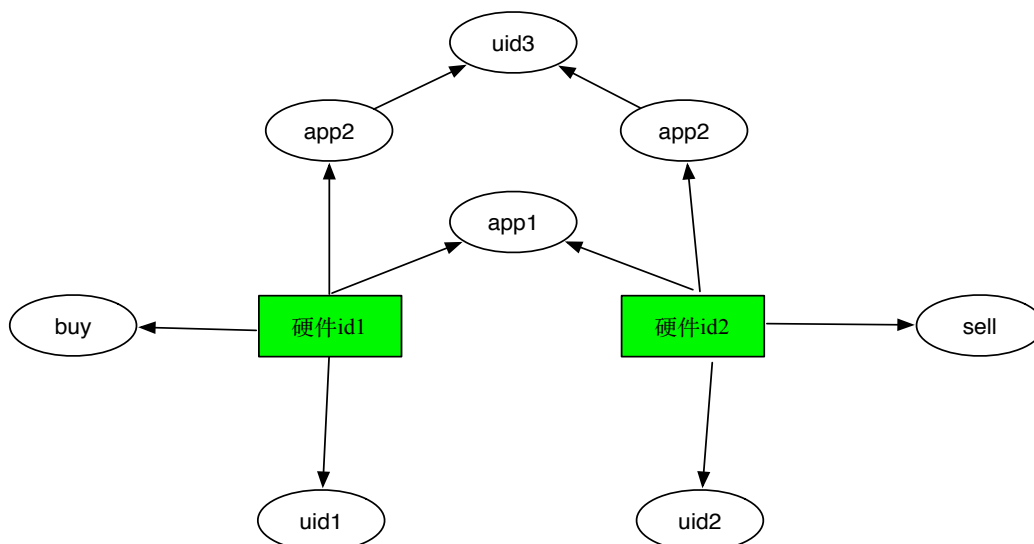


图13-23 疑似刷单示意图

从拓扑图可以看出来，虽然两台设备 hid1 和 hid2 登录账户不一样，但是他们共同安装的 app2 上的登录用户名相同，从而可以判断这两台设备属于同一个人，该人疑似使用这两台设备分别扮演买家和卖家进行刷单行为。

逐行处理样本文件，获取对应的 `hid,uid,app` 字段：

```
with open("../data/KnowledgeGraph/sample3.txt") as f:
    for line in f:
        line=line.strip('\n')
        hid,uid,app=line.split(',')
        # print(hid,uid,app)
```

以 hid 为中心，添加对应的 uid,app 结点:

```
G.add_edge(hid, uid)
G.add edge(hid, app)
```

逐行处理样本文件，获取对应的 `hid,uid,action` 字段:

```
with open("../data/KnowledgeGraph/sample4.txt") as f:
    for line in f:
        line=line.strip('\n')
        hid,uid,action=line.split(',')
        # print(hid,uid,action)
```

以 hid 为中心，添加对应的 uid,action 结点:

```
G.add_edge(hid, uid)

G.add_edge(hid, action)
```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)

plt.show()
```

对应知识图谱为：

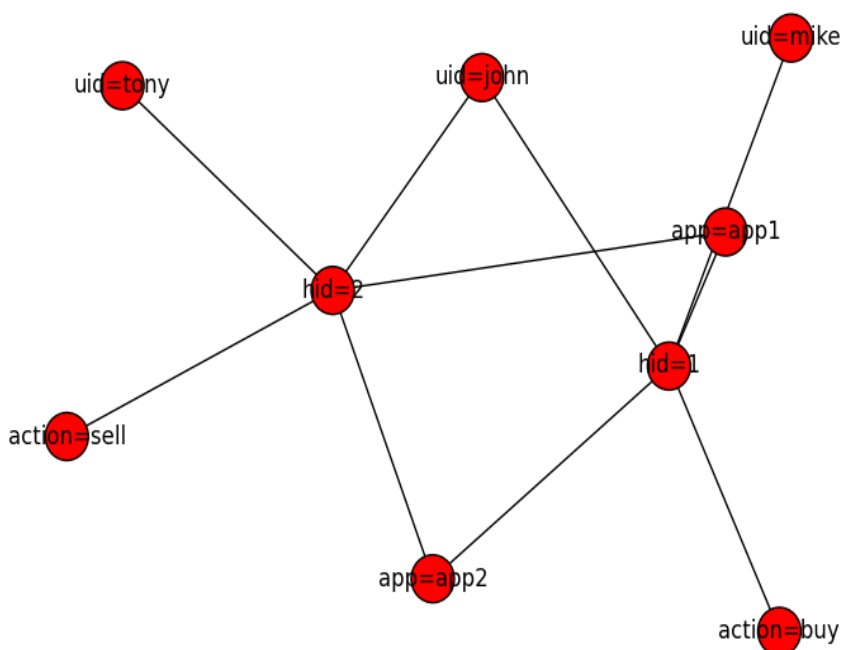


图13-24 疑似刷单行为知识图谱

13.8 示例：知识图谱在威胁情报领域的应用

威胁情报是信息安全领域近几年非常热的一个领域，威胁情报基于海量的数据分析，挖掘潜在的关联关系，为其他安全产品提供强大的数据情报支撑，目前以前涌现了大量的创业企业投入到威胁情报领域。



图13-25 老牌安全公司 Symantec

在万物互联的当今时代，攻击入口无处不在，单纯基于漏洞或者关键资产的防御方式早已力不能及。因此，企业想要安全的开展公司业务，就必须采取更加全面、高效的防御方式。威胁情报的出现弥补了这一不足，为传统防御方式带来了有效补充。威胁情报立足于攻击者的视角，依靠其广泛的可见性以及对整个互联网风险及威胁的全面理解，帮助我们更好的了解威胁，包括：可能的攻击目标、使用工具、方法以及所掌握的传输武器的互联网基础设施情况等，使遇到威胁时能够准确、高效的采取行动。在 2016 年的 RSA 大会上出现了 10 家威胁情报公司，其中包括老牌安全公司 Symantec、Dell Security，也包括大量新秀 Webroot、CrowdStrike，其中还包含国内的一家创业公司 ThreatBook。



图13-26 威胁情报创业公司 ThreatBook

ThreatBook 成立于 2015 年 7 月，致力于提供及时、准确的威胁情报，用来拦截攻击、发现威胁、溯源追踪和消除风险。ThreatBook 的主要产品 TIC(威胁情报中心)的威胁应用解决方案，使客户在面对关键威胁时可以快速发现并采取有效的行动。

完整演示代码请见本书 Github 上的 13-6.py。

13.8.1 挖掘后门文件潜在联系

黑产通常通过传播后门文件，入侵主机组织起庞大的僵尸网络，后门文件通常通过连接 C&C 服务器的域名来监听控制指令，后门文件中硬编码少量 C&C 服务器的域名，然后自动化下载最近的 C&C 服务器列表。通过静态分析后门文件中硬编码的域名，关联分析域名和文件之间的关系，可以挖掘出后门文件之间的潜在联系。我们使用脱敏的测试数据来挖掘后门文件之前潜在联系的原理。测试样本记录了若干组文件和 C&C 域名的对应关系。

```
md5=file1,domain=domain1
```



```
md5=file1,domain=domain2  
  
md5=file1,domain=domain3  
  
md5=file1,domain=domain6  
  
md5=file2,domain=domain2  
  
md5=file2,domain=domain3  
  
md5=file2,domain=domain4  
  
md5=file2,domain=domain5
```

❑ md5, 后门文件 md5 值

❑ domain, C&C 域名

逻辑上对应的拓扑图为:

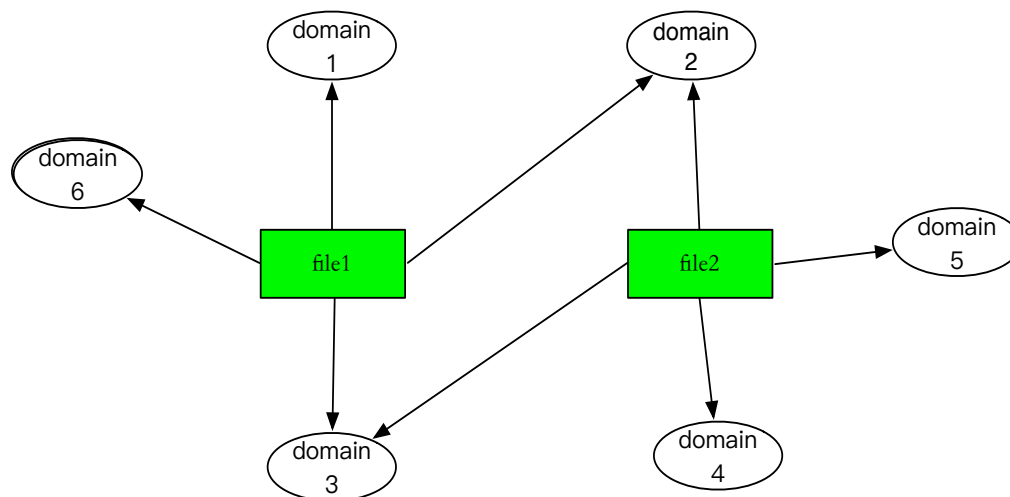


图13-27 后门文件潜在关系示意图

从拓扑图可以看出，后门文件 file1 和 file2 分别对应 C&C 域名 domain1、domain2、domain3、domain6 和 domain2、domain3、domain4、domain5，其中 domain2 和 domain3 同时被 file1 和 file2 使用，初步怀疑邮箱 file1 和 file1 被同一黑产团体控制的后门文件，domain1-domain4 均疑似黑产同时控制，并很可能是同一用途，比如 DDoS。

逐行处理样本文件，获取对应的 file，domain:

```
with open("../data/KnowledgeGraph/sample6.txt") as f:  
  
    for line in f:  
  
        line=line.strip('\n')  
  
        file,domain=line.split(',')
```

以 file 为中心，添加对应的 domain 结点：

```
G.add_edge(file, domain)
```

可视化知识图谱：

```
nx.draw(G, with_labels=True, node_size=600)
plt.show()
```

对应知识图谱为：

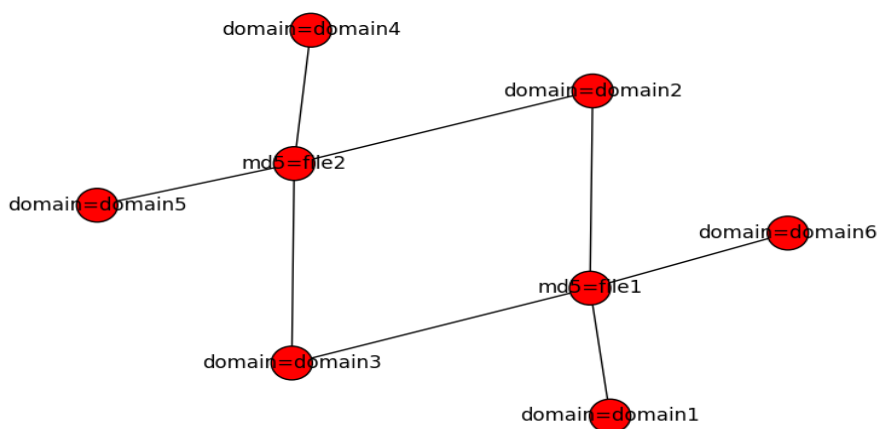


图13-28 后门文件潜在关系知识图谱

13.8.2 挖掘域名潜在联系

黑产通常会注册大量的域名用于 C&C 服务器、钓鱼等，注册域名时会登记注册人的邮箱信息，通过关联 ip、注册邮箱、域名可以挖掘潜在的关联关系。我们使用脱敏的测试数据来挖掘域名之前潜在联系的原理。测试样本记录了若干组 ip、注册邮箱、域名的对应关系。

```
mail=mail1, domain=domain1, ip=ip1
mail=mail1, domain=domain3, ip=ip2
mail=mail2, domain=domain2, ip=ip1
mail=mail2, domain=domain4, ip=ip2
mail=mail2, domain=domain5, ip=ip3
```

字段含义分别为：

- ❑ mail, 注册邮箱
- ❑ domain, 注册域名
- ❑ ip, 注册域名对应的 ip

逻辑上对应的拓扑图为:

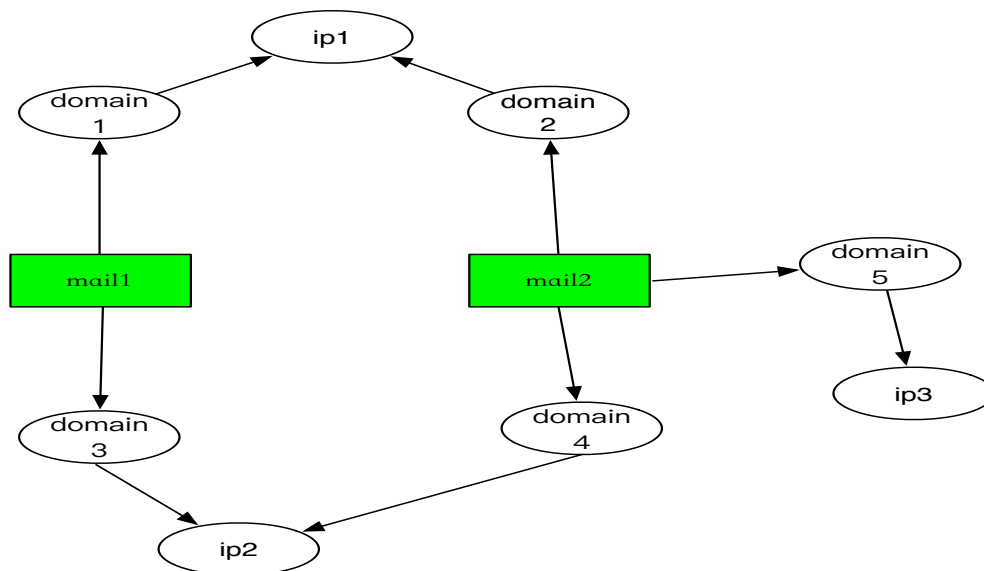


图13-29 域名潜在关系示意图

从拓扑图可以看出, 邮箱 mail1 和 mail2 分别注册了域名 domain1、domain3 和 domain2、domain4、domain5, 其中 domain1 和 domain2 都指向同一个 ip1, domain3 和 domain4 都指向同一个 ip2, 初步怀疑邮箱 mail1 和 mail2 被同一黑产团体控制, domain1-domain4 均疑似黑产同时控制, 并很可能是同一用途, 比如 C&C 服务器或者钓鱼网站。

逐行处理样本文件, 获取对应的 mail, domain, ip:

```

with open("../data/KnowledgeGraph/sample5.txt") as f:

    for line in f:

        line=line.strip('\n')

        mail, domain, ip=line.split(',')
  
```

以 mail 为中心, 添加对应的 domain 结点:

```
G.add_edge(mail, domain)
```

以 domain 为中心, 添加对应的 ip 结点:

```
G.add_edge(domain, ip)
```

可视化知识图谱:

```
nx.draw(G, with_labels=True, node_size=600)

plt.show()
```

对应知识图谱为：

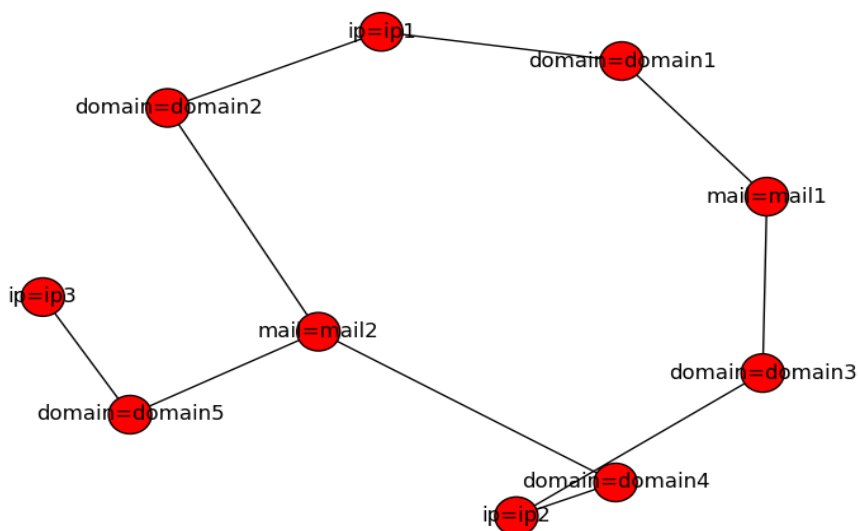


图13-30 域名潜在关系知识图谱

13.9 本章小节

本章重点介绍图算法和知识图谱的基础知识以及在黑产团体挖掘、Webshell 检测方面的相关应用。知识图谱其实可以理解作为一种特殊的图结构，它被广泛应用于各个领域，在风控和威胁情报领域的应用远比本章例子举的复杂，需要大家结合实际情况多去挖掘。

13.9.1 参考文献

- 1) <https://neo4j.com/>
- 2) <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
- 3) <http://www.ll.mit.edu/ideval/data/1999data.html>
- 4) <http://www.secrepo.com/>
- 5) 卜月华，王维凡，吕新忠.图论及其应用 第2版：东南大学出版社，2015
- 6) 李航．统计学习方法．北京：清华大学出版社，2012

- 7) 李文哲 <http://www.infoq.com/> 知识图谱在互联网金融中的应用
- 8) 唯品会, 知识图谱在风控的应用简述
- 9) <http://netsecurity.51cto.com/art/201603/506654.htm>
- 10) <https://x.threatbook.cn>
- 11) A Uyar, FM Aliyu, Online Information Review,Evaluating search features of Google Knowledge Graph and Bing Satori: Entity types, list searches and query interfaces