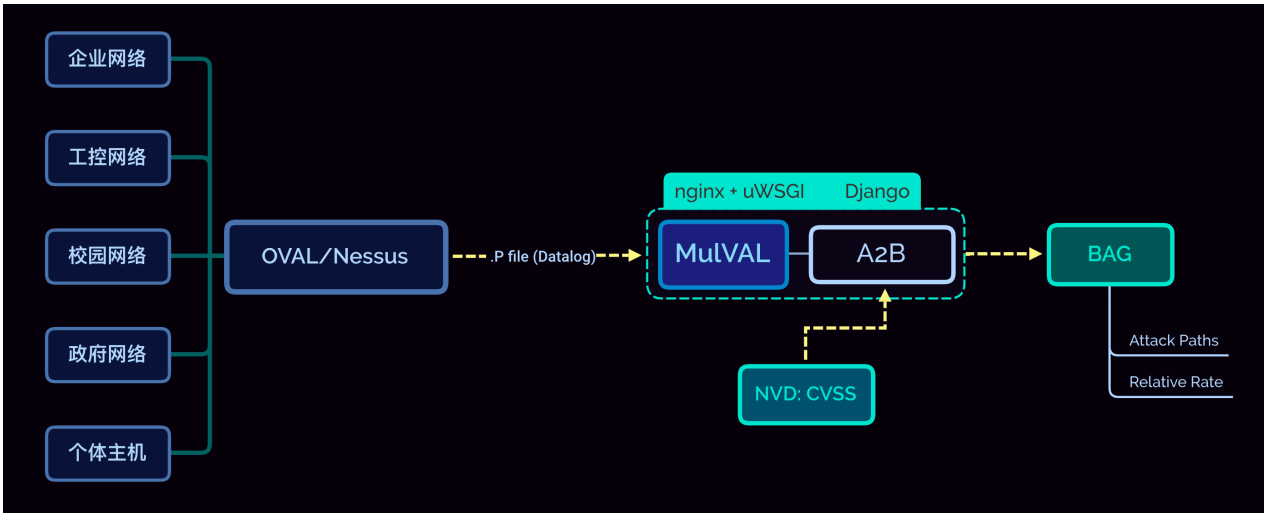
BAG-MulVAL 详细设计说明书



BAG-MulVAL 详细设计说明书

- 输入
- MulVAL模块
  - 攻击图理论(Attack Graph)
  - MulVAL 工具介绍
- A2B模块
  - 贝叶斯攻击图理论(Bayesian Attack Graph)
  - 算法：由属性攻击图生成贝叶斯攻击图(A2B: AG-to-BAG)
    - 数据结构
    - 解析攻击图的 XML 文件
    - 基于 CVSS 的原子攻击难度的度量指标和计算方法
    - 结构转换：消除含圈路径
    - 攻击路径拆分
    - 攻击路径贝叶斯概率计算
      - *LEAF* 型节点
      - *AND* 型节点
      - *OR* 型节点
      - 攻击路径概率计算
    - 可视化贝叶斯攻击图（攻击路径）
- Django 项目
  - 后端控制脚本 view.py
  - 前端静态文件 index.html

输入

输入文件为后缀名为 **.P** 的文本文件，由 *Nessus* 等 *OVAL* 扫描器的 **XML** 格式的扫描报告转化得到，如下所示，详细解读在注释中，具体语法可见论文[1]和原理[2][3]。

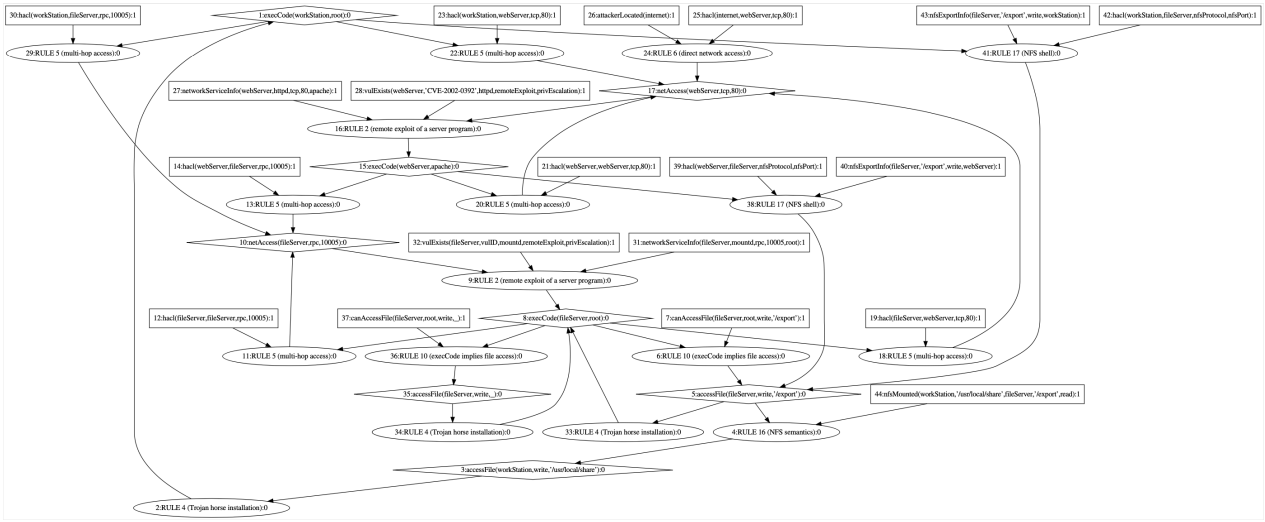
```
/* attacker information: location and goal */
attackerLocated(internet).  /* attacker is on the internet */
attackGoal(execCode(workStation,_)).  /* the goal of the attack is to obtain the privilege to execute any code on the workstation in this network */

/* network interconnection information */
hacl(internet, webServer, tcp, 80). /* there is a connection between the internet and the webserver in the network through port 80 using TCP protocol */
hacl(webServer, _,  _, _).  /* three hosts in the network are connected to each other directly */
hacl(fileServer, _, _, _).
hacl(workStation, _, _, _).
hacl(H,H,_,_).

/* configuration information of fileServer */
networkServiceInfo(fileServer, mountd, rpc, 10005, root). /* service working in root privilege through port 10005 using RPC protocol */
nfsExportInfo(fileServer, '/export', _anyAccess, workStation).
nfsExportInfo(fileServer, '/export', _anyAccess, webServer).
localFileProtection(fileServer, root, _, _).

/* configuration information of webServer */
vulExists(webServer, 'CVE-2002-0392', httpd). /* a vulnerability of webserver with CVE number: 'CVE-2002-0392' */
vulProperty('CVE-2002-0392', remoteExploit, privEscalation).  /* the capability of the vulnerability 'CVE-2002-0392' is being exploited remotely to escalate the privilege of the attacker */
networkServiceInfo(webServer , httpd, tcp , 80, apache).  /* apache service working through port 80 using TCP protocol */

/* configuration information of workStation */
nfsMounted(workStation, '/usr/local/share', fileServer, '/export', read).
```

MulVAL模块

攻击图理论(Attack Graph)

攻击图是一种基于模型的网络安全评估技术。它从攻击者的角度出发，在综合分析多种网络配置和脆弱性信息的基础上，找出所有可能的攻击路径，并提供了一种表示攻击过程场景的可视化方法，从而帮助网络安全管理人员直观地理解目标网络内各个脆弱性之间的关系、脆弱性与网络安全配置之间的关系以及由此产生的潜在威胁。基于攻击图模型的网络安全评估技术是在攻击图的基础上进行深入的安全评估建模和分析，给出安全评估的建议。

攻击图模型是 *Swiler* 等人于 1998 年提出的，其目的是为了在网络安全分析中把网络拓扑信息也考虑在内。攻击图的代表性科研成果有 **TVA**，**NetSPA v2**，**MulVAL** 等原型系统[4]。



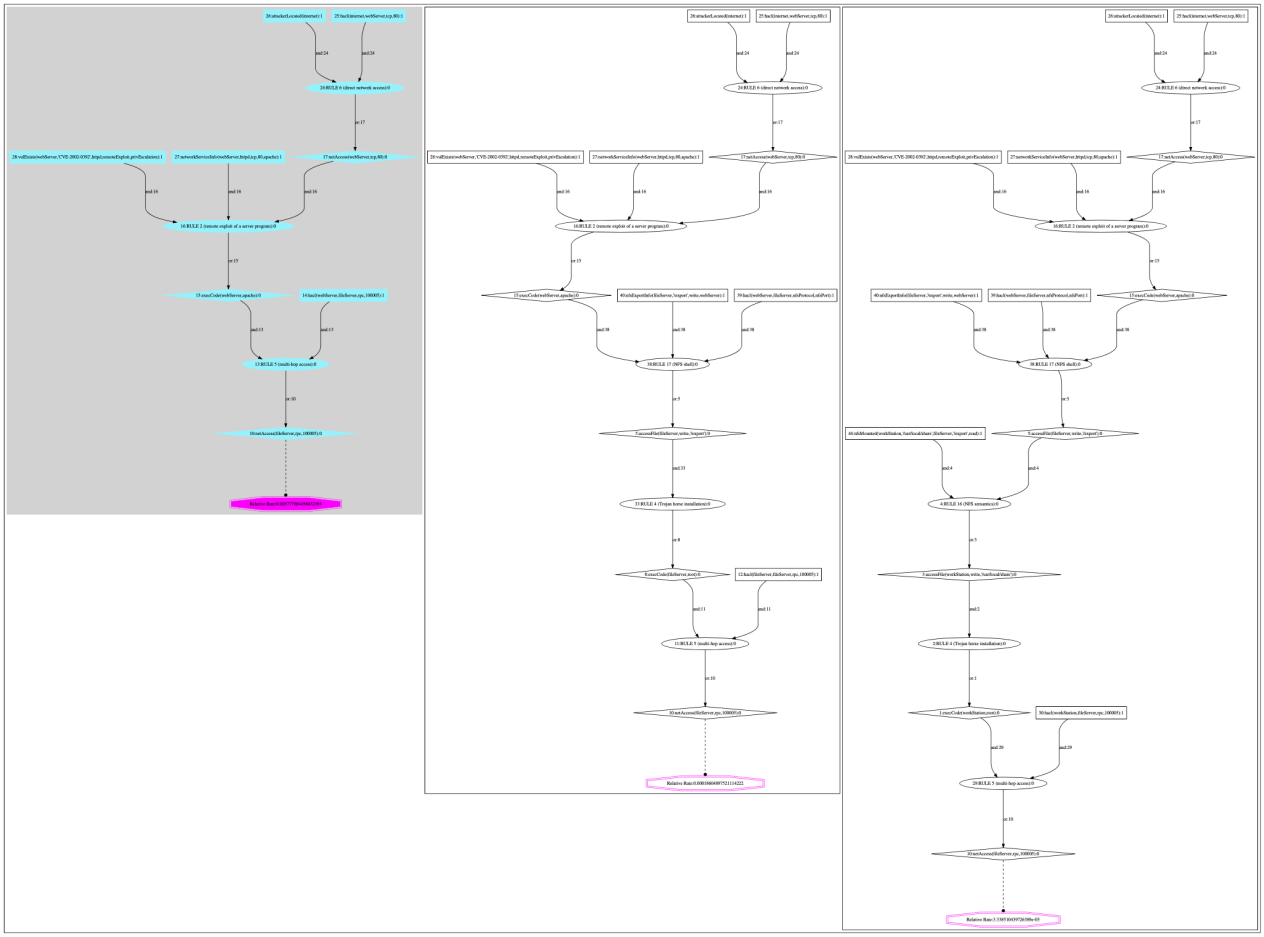
MulVAL 工具介绍

MulVAL 是一个基于攻击图理论的开源企业网络安全分析工具。它能基于 **Datalog**[2] 和逻辑规则对整个网络的信息（网络主机互连信息，配置信息，漏洞信息，攻击者信息）进行综合分析，最后生成所有可能的攻击路径，即攻击图，从而辅助进行**企业网络安全态势感知**。由于其算法比较复杂，这里不做具体介绍。该工具的具体算法可参考论文[1][3]，安装使用及其他资料可参考官方网站[5]和说明书[6]。

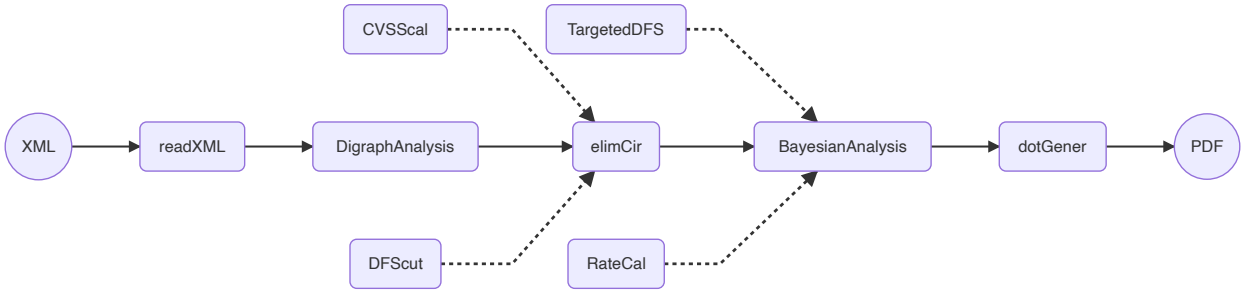
A2B模块

贝叶斯攻击图理论(Bayesian Attack Graph)

贝叶斯攻击图由属性攻击图转化而来。**属性攻击图**是一种有向无环图，它含有 2 种节点：属性节点和原子攻击节点。**贝叶斯网络**是一个有向无环图，具有因果关系和概率语义，可以根据已知的信息对未知的结果或原因进行推理和预测。在贝叶斯网络中，节点的状态及发生概率只与其父节点有关，在攻击图中网络脆弱点是否被利用也只与攻击路径中的父节点有关，这种节点关系在贝叶斯网络和攻击图中相对应；贝叶斯网络和攻击图都是一种有向无环图，并且有向边表示一种因果关系。因此可以将贝叶斯网络和攻击图进行结合，对网络的脆弱性进行量化评估，即贝叶斯攻击图。本项目中，对贝叶斯攻击图进行了进一步的路径拆分，生成可视化的贝叶斯攻击路径，使结果更直观易懂。



算法：由属性攻击图生成贝叶斯攻击图(A2B: AG-to-BAG)



数据结构

贝叶斯攻击图表示为 $BAG < A, E, P >$。其中，A 表示属性节点集合；E 表示有向边集合，即原子攻击；P 表示属性节点被利用的条件概率表。

BAG 满足以下条件：

- $\forall E \subseteq A \times A$ 对 $\forall e \epsilon E$，都有 $e = E_{in}(e) \rightarrow E_{out}(e)$，其中 $E_{in}(e)$ 表示原子攻击的结果属性，“$\rightarrow$” 表示两个属性节点之间的因果关系。
- 条件独立假设：任意节点只与其父节点有关，与其非父节点均相互独立。
- 属性节点 A 是一个二元变量，只有两种状态，A 为 $false$ 时表示攻击者已经获得该属性，A 为 $true$ 时表示攻击者还未获得该属性。

攻击图结构：

```
# 图结构
class Graph:
    def __init__(self):
        self.nodgrp = []    # 图的所有节点集合
        self.arcgrp = []    # 图的边集合
        self.attacker = []  # 攻击起点集合
        self.aim = []       # 攻击目标集合
        self.rate = 1       # 攻击路径相对贝叶斯概率
```

节点结构：

```
# 图的节点结构
class Node:
    def __init__(self, ID, fact, metric, TYPE):
        self.id = ID          # 节点id
        self.fact = fact      # 节点内容
        self.metric = metric  # 节点是否被利用
        self.type = TYPE      # 节点类型，有LEAF、AND、OR三种
        self.cve = ''         # 漏洞的cve编号。若节点内容中有漏洞内容，则该节点有此值
        self.prior = []       # 节点的父节点集合
        self.next = []        # 节点的子节点集合
        self.priarc = []      # 节点的入度集合
        self.nexarc = []      # 节点的出读集合
        self.D = 0            # 攻击难度，若为攻击节点则该节点有值
        self.rate = 0.9       # 利用概率。初始值为0.9
        self.flag = 0         # 标记，用于后续消除含圈路径时的染色算法，减小算法时间复杂度
        self.tempnext = []    # 零时子节点集合，用于后续深度优先搜索
```

边结构：

```
# 边结构
class Edge:
    def __init__(self, src, dst):
        self.src = src  # 边的起始节点
        self.dst = dst  # 边的终止节点
        self.fact = ''  # 边的类型，有and、or两种
        self.subg = 0   # 边的归属
```

构建贝叶斯网络属性攻击图模型主要涉及两方面：模型网络结构的构建和节点发生概率的计算。

解析攻击图的 XML 文件

使用 *Python* 的 **xml** 库解析攻击图的 **XML** 文件，利用上述数据结构创建一张图结构，并依据对图的内容的全面分析将所有有效的文本信息与逻辑信息存入图中。

基于 CVSS 的原子攻击难度的度量指标和计算方法

原子攻击是利用网络中的 1 个脆弱点进行的一次攻击，可以将脆弱点被利用的难易度进行量化作为原子攻击难度的度量值，该值可以用通用漏洞评分系统 *(CVSS, common vulnerability scoring system)* 评分计算得到[7]。

CVSS，全称 *Common Vulnerability Scoring System*，即“通用漏洞评分系统”，是一个 “行业公开标准，其被设计用来评测漏洞的严重程度，并帮助确定所需反应的紧急度和重要度” 。*CVSS* 是安全内容自动化协议（*SCAP*）的一部分，通常 *CVSS* 同 *CVE* 一同由美国国家漏洞库（NVD）发布并保持数据的更新。

CVSS 中基本评分 *(BM，base metrics)* 部分描述的是脆弱点的基本属性，其中包含 6 个指标，前 3 个指标 是 *AV (access vector)* 、*AC (access complexity)* 、*AU (authentication)* ，用于描述脆弱点的可用性。3 个 *CVSS* 指标的度量值根据可利用性分为 3 个等级: **Low**、**Mid**、**High**。分值越高，该脆弱点越容易被利用。**由低到高，*AV* 值依次为 0. 359、0. 646、1. 0； *AC* 值依次为 0. 35、0. 61、0. 71； *AU* 值依次为 0. 45、0. 56、 0. 704。**

在 *CVSS* 中，脆弱点的可用性指标定义为 $E = 20VCU(0 \leq E \leq 10)$。 Ex 的值越小，表示原子攻击的难度越大. 由于可用性与攻击难度成反比关系，因此根据这 3 个指标计算原子攻击的难度，用 D 表示相应原子攻击的难度，其值越大攻击难度越大，计算公式为

$$D = \frac{1}{VCU}, D \geq 1 \tag{1}$$

```python
def CVSSCal(cveid):
    '''根据CVE查询AV、AC、AU'''
    file = './cveid.xls'    # 目前爬取到的数据库存储在Exel表中
    data = xlrd.open_workbook(file)
    table = data.sheets()[0]
    cve = table.col_values(0)
    av = table.col_values(2)
    ac = table.col_values(3)
    au = table.col_values(4)

    try:
        result = cve.index(cveid)
    except:
        print('Unknown vulnerability.', cveid)
        return 1.0, 0.71, 0.704
    else:
        if av[result] == 'N':
            AV = 1.0
        elif av[result] == 'A':
            AV = 0.646
        else:
            AV = 0.359
        if ac[result] == 'L':
            AC = 0.71
        elif ac[result] == 'M':
            AC = 0.61
        else:
            AC = 0.35
        if au[result] == 'N':
```

```
            AU = 0.704
        elif au[result] == 'S':
            AU = 0.56
        else:
            AU = 0.45
        return AV, AC, AU
```

结构转换：消除含圈路径

在攻击图中经常会出现攻击环路。由于攻击者是智能的，即攻击者不会重复获得已经具有的攻击能力，且随着攻击的进行攻击者的能力是单调递增的，因此在攻击图中出现攻击环是不合理的，也不便于理解攻击过程。

最远原子攻击消环算法思想：该算法的输入是一个可能含环路的攻击图，将攻击图的入口节点加入到根节点数组中，初始化一个栈用来存放找到的环路。从一个入口根节点进行**深度优先遍历**，不断地访问子节点，将访问到的子节点进行标记并压入栈，直到不存在子节点或访问的节点在栈中已经出现为止。当子节点在栈中已经存在时，栈中存储了一个环路，此时**删除环路中最晚出现的攻击节点**，消除环路。在遍历时，如果子节点不存在，则进行回溯，将栈中节点出栈，查询新的未被访问的子节点重复上述深度优先遍历过程。当栈为空时，则以该入口根节点为起始节点的遍历完成，从下一个新的入口根节点开始按上述方法遍历. 直到所有的入口根节点都遍历完，则整个攻击图消环完成，此时得到一个无环的攻击图。在该无环攻击图中，保留的原子攻击是最可能发生的，符合攻击的实际情况，即在同一条路径中最远的原子攻击成功的先验概率最小，而容易成功的原子攻击最可能被攻击者利用[8][9]。

由于攻击图并非简单的有向图。其节点与边有较为复杂的逻辑关系，删除节点会连带很多其他节点，影响整个图。所以还设计了递归算法来处理删除节点对其他节点的影响。

攻击路径拆分

以 **attackerLocated** 节点为起始节点对消除含圈路径后的攻击图进行**深度优先遍历**，使用栈来暂存。到达目标节点时视作发现一条攻击路径，立即创建一个新的攻击图结构来保存找到的攻击路径。

```
def TargetedDFS(graph, attacker, terminal):
    '''深度优先搜索找到所有攻击路径'''
    stack = Stack()
    InitExceptStack(graph, stack)
    stack.PUSH(attacker)
    attacker.flag = 1
    subgraphlist = []
    while stack.isnot_empty():
        if stack.peek().tempnext:
            temp = stack.peek().tempnext[-1]
            stack.peek().tempnext.pop()
            temp.flag = 1
            stack.PUSH(temp)
        else:
            temp = stack.peek()
            if temp.type == 'OR':
                if temp == terminal:
                    subgraph = Graph()
```

```
                    subgraph.aim.append(temp)
                    subgraph.attacker.append(attacker)
                    subgraph.nodgrp = stack.copy()
                    temp = subgraph.nodgrp.copy()
                    for nod in temp:
                        if nod.type == 'AND':
                            eatAcient(graph, subgraph, nod)
                    subgraph.nodgrp = list(set(subgraph.nodgrp))
                    subgraphlist.append(subgraph)
            stack.POP()
            InitExceptStack(graph, stack)

    return subgraphlist
```

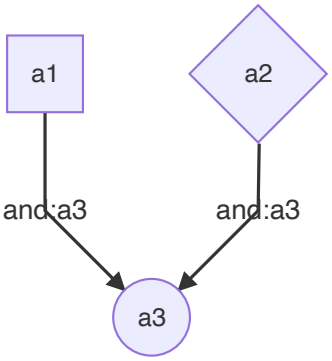
攻击路径贝叶斯概率计算

***LEAF* 型节点**

计算贝叶斯网络中 a_3 节点（**LEAF** 节点：方形）的概率。

$$P(a_3) = CVSScal(a_3.fact) \tag{2}$$

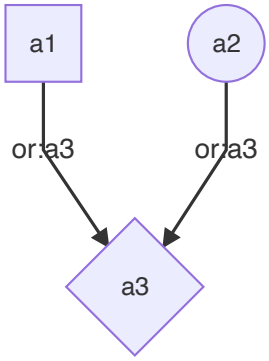
***AND* 型节点**



计算贝叶斯网络中 a_3 节点（**AND** 节点：椭圆形）的概率。

$$P(a_3) = P(a_3|a_1, a_2)P(a_1, a_2) \tag{3}$$

***OR* 型节点**



计算贝叶斯网络中 a_3 节点（**OR** 节点：菱形）的概率。

$$P(a_3) = P(a_3|a_1, a_2)P(a_1, a_2) + P(a_3|a_1, \bar{a_2})P(a_1, \bar{a_2}) + P(a_3|\bar{a_1}, a_2)P(\bar{a_1}, a_2) \tag{4}$$

攻击路径概率计算

在贝叶斯网络中属性节点之间的有向边代表原子攻击被利用的过程，原子攻击难度越大脆弱点被利用的概率就越低，两者成反比关系。因此，可以用原子攻击的难度代表属性节点之间的条件概率，定义条件概率为

$$P_s = \frac{1}{D}, 0 < P_s < 1 \tag{5}$$

设攻击者要获得的目标属性为 obj，在攻击路径上 obj 的所有直接和间接父节点为 $Pre(obj)$，直接父节点为 $DPre(obj)$，则目标节点受到攻击的概率为

$$P(obj) = P(obj|Pre(obj))P(Pre(obj)) \tag{6}$$

由于贝叶斯网络中条件独立性的假设，式(5)可以改写为

$$P(obj) = P(obj|DPre(obj))P(DPre(obj)) \tag{7}$$

迭代式 (7)，就可求得目标属性在该攻击路径中被攻击的概率。

从攻击目标节点开始往上进行递归运算，最后路径的贝叶斯概率等于返回的攻击目标节点的绝对概率。

```
def OrBayesian(node, parents, subgraph):
  '''OR型节点概率计算'''
    rates = []
    for nod in parents:
        rates.append(RateCal(nod, subgraph))
    rate = 0
    i = 0
    a = []
    while i < len(rates):
        a.append(i)
        i = i + 1
    i = 0
    while i < len(rates):
        rates_temp = []
        for b in combinations(a, i):
```

```
            for j in b:
                rates_temp.append(1 - rates[j])
            for j in a:
                if j not in b:
                    rates_temp.append(rates[j])
            rate_temp = 1
            for rat in rates_temp:
                rate_temp = rate_temp * rat
            rate = rate + rate_temp
        i = i + 1
    rate = rate * node.rate
    return rate

def AndBayesian(node, parents, subgraph):
  '''AND型节点概率计算'''
    rate = node.rate
    for nod in parents:
        rate = rate * RateCal(nod, subgraph)
    return rate

def RateCal(node, subgraph):
    '''递归计算攻击路径相对概率'''
    if node.type == 'LEAF':
        return node.rate
    elif node.type == 'OR':
        parents = []
        for nod in node.prior:
            if nod in subgraph.nodgrp:
                parents.append(nod)
        rate = OrBayesian(node, parents, subgraph)
        return rate
    else:
        rate = AndBayesian(node, node.prior, subgraph)
        return rate
```

可视化贝叶斯攻击图（攻击路径）

使用 *Python* 的 **graphviz** 库将整个贝叶斯攻击图录入一个 **.dot** 文件中，调用 **dot.render** 函数生成 **PDF** 文件。

完整A2B.py文件路径：./代码/MulVAL_BAG/mulvala2b/src/A2B.py

Django 项目

后端控制脚本 view.py

```
from django.shortcuts import render, redirect
from django.urls import reverse
from mulvala2b import models
```

```python
from mulvala2b.src.A2B import A2B, aimSel
from django.shortcuts import HttpResponse
import os
import shutil

# Create your views here.

def window(req):
  '''首页展示'''
    return render(req, "index.html")

def mulval(req):
  '''调用MulVAL生成攻击图'''
    shutil.rmtree('./mulvala2b/src/mulvalsrc')
    os.mkdir('./mulvala2b/src/mulvalsrc') # 每次工作前清空工作目录
    print("前端数据: ", req.POST)
    print("file:", req.FILES)
    if req.method == "POST":
        file = req.FILES.get("upload", None)
        if not file:
            return render(req, "index.html", {"errinf":"No files for upload!"})
        f = open("./mulvala2b/src/mulvalsrc/input.P", 'wb')
        for line in file.chunks():      # 分块写入
            f.write(line)
        f.close()
    root = os.getcwd()
    path = root + "/mulvala2b/src/mulvalsrc"
    if os.system("cd "+ path + " && ls && graph_gen.sh input.P -v"):  # 调用MulVAL
        return redirect('/mulval/mulvalerror1/')
    elif os.path.exists('./mulvala2b/src/mulvalsrc/AttackGraph.pdf'):
        return redirect('/mulval/mulvalsuccess/')
    else:
        return redirect('/mulval/mulvalerror2/')

def mulvalerror1(req):
  '''文件类型错，收到文件但MulVAL无法解析'''
    return render(req, "index.html", {"errinf":"Wrong file! Please check your file type or grammer."})

def mulvalsuccess(req):
  '''MulVAL生成攻击图成功，可以进行A2B'''
    aimlist = aimSel()
    return render(req, "index.html", {"goodnews": "An AG was generated successfully.", "retcode": 1, "aimlist": aimlist})

def mulvalerror2(req):
  '''网络健康，无攻击图'''
```

```
    return render(req, "index.html", {"errinf": "No attack path find."})

def download(req):
  '''下载MulVAL生成的原始攻击图'''
    if os.path.exists('./mulvala2b/src/mulvalsrc/AttackGraph.pdf'):
        pdfFileObj = open('./mulvala2b/src/mulvalsrc/AttackGraph.pdf', 'rb')
        response = HttpResponse(pdfFileObj.read(), content_type='application/pdf')
        response['Content-Disposition'] = 'attachment; filename="AttackGraph.pdf"'
        return response

def a2b(req):
  '''调用A2B模块'''
    if req.method == "POST":
        aim = req.POST.get("Attack Goal", None)
        print(aim)
        A2B(aim)
        if os.path.exists('./mulvala2b/src/mulvalsrc/result.dot.pdf'):
            pdfFileObj = open('./mulvala2b/src/mulvalsrc/result.dot.pdf', 'rb')
            response = HttpResponse(pdfFileObj.read(), content_type='application/pdf')
            response['Content-Disposition'] = 'attachment; filename="BAG.pdf"'
            return response
        else:
            return redirect('/mulval/a2berror/')

def a2berror(req):
  '''该目标节点健康，无攻击路径能到达'''
    aimlist = aimSel()
    return render(req, "index.html",{"goodnews": "An AG was generated successfully.", "a2berror": "No attack path to this target!", "retcode": 1, "aimlist": aimlist})
```

前端静态文件 index.html

```
<!DOCTYPE html>
<html>
{% load static %}
<style>
    html{text-align:center}
</style>
<embed lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <title>test</title>
</head>
<body style="background-image: url({% static '/imgs/S-ICS.png' %});background-size: auto 700px;background-position: 0px 0px;background-repeat:no-repeat;background-attachment:fixed;">
    <h1 style="background: wheat;color: #000000;">BAG-MulVAL</h1>
    <h3>Please upload a .P file</h3>
    <form action="/mulval/mulval/" method="post" enctype="multipart/form-data">
        <input type="file" name="upload"/>
        <input type="submit" value="MulVAL"/>
    </form>
    <small style="color: red">{{ errinf }}</small>
    <small style="color: black">{{ goodnews }}</small>
    <br>

    {% if retcode == 1 %}
        <form action="/mulval/download/" method="post">
            <input type="submit" value="Download the AG">
        </form>
        <br>
        <small style="color: red">{{ a2berror }}</small>
        <form action="/mulval/a2b/" method="post">
            <select name="Attack Goal">
                {% for aim in aimlist %}
                    <option>{{ aim }}</option>
                {% endfor %}
                <option value="_">ALL</option>
            </select>
            <input type="submit" value="Generate a BAG!"/>
        </form>
    {% endif %}
    <h2 style="color: black">Attack Graph Example</h2>
    <embed src="/static/imgs/AttackGraph.pdf" width="1000" type="application/pdf" height="500" wmode='transparent'/>
    <h2 style="color: black">Bayesian Attack Graph Example</h2>
    <embed src="/static/imgs/result.dot.pdf" width="1000" type="application/pdf" height="500" wmode='transparent'/>

    <!--<img src="/static/imgs/IMG_2175.jpeg"/>-->
    <script src="//cdn.bootcss.com/canvas-nest.js/1.0.1/canvas-nest.min.js"></script>
    <script src="https://cdn.bootcss.com/jquery/3.4.1/jquery.min.js"></script>
</body>
</html>
```

1. **MulVAL: A logic-based network security analyzer.** Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. In *14th USENIX Security Symposium*, Baltimore, Maryland, U.S.A., August 2005. ↩ ↩

2. **基于 Datalog 的知识推理**. https://blog.csdn.net/oanqoanq/article/details/78932517 ↩ ↩

3. **A scalable approach to attack graph generation.** Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. In *13th ACM Conference on Computer and Communications Security (CCS 2006)*, Alexandria, VA, U.S.A., October 2006. ↩ ↩

4. **基于攻击图模型的多目标网络安全评估研究**. 程叶霞, 姜文, 薛质, 程叶坚. *计算机研究与发展*. ISSN 1000-1239/CN 11-1777/TP 49(Suppl.):23-31, 2012 ↩

5. **Ou X.MulVal[EB/OL]**. (2011-04-24)[2020-4-30]. http://people.cis.ksu.edu/~xou/mulval/. ↩

6. **MulVAL: A logic-based, data-driven enterprise security analyzer.** http://people.cs.ksu.edu/~xou/argus/software/mulval/readme.html ↩

7. **基于贝叶斯攻击图的最优安全防护策略选择模型**. 高妮, 高岭, 贺毅岳, 王帆. *Computer Engineering and Applications*. 2016, 52(11):125-130 ↩

8. **贝叶斯属性攻击图网络脆弱性评估**. 王秀娟, 孙博, 廖彦文, 相从斌. *Journal of Beijing University of Posts and Telecommunications*. 10.13190/j.jbupt.2015.04.022 ↩

9. **基于攻击图的网络安全概率计算方法[J]**. 叶云, 徐锡山. *计算机学报*, 2010 (10) : 1987-1996. ↩