计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(12):147−154 [doi: 10.15888/j.cnki.csa.008211]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# 基于 XGBoost 和社区发现的主机攻击行为检测①

朱元庆, 李赛飞, 李洪赭

(西南交通大学 信息科学与技术学院, 成都 611756)
通讯作者: 李赛飞, E-mail: lisaifei@swjtu.edu.cn

**摘　要**: 在大规模网络环境下, 主机面临的安全威胁也愈发多样. 随着基于机器学习检测恶意文件的技术快速崛起, 极大的提升了对恶意软件的检测能力, 也迫使对手改变了攻击策略. 其中“Living off the land”策略通过调用操作系统工具或者执行任务的自动化管理程序来实现恶意行为. 威胁检测可以从父子进程的上下文中发现可疑行为, 将父子进程链及其派生的相关事件看作无向图, 应用监督学习 XGBoost 算法进行权重分配, 生成无向加权图. 最后使用社区发现算法从图中识别出更大的攻击序列, 在 MIRTE ATT & CK 仿真攻击数据集上进行验证.

**关键词**: 主机攻击行为; XGBoost; 社区发现; 图分析

引用格式: 朱元庆,李赛飞,李洪赭.基于 XGBoost 和社区发现的主机攻击行为检测.计算机系统应用,2021,30(12):147–154. http://www.c-s-a.org.cn/1003-3254/8211.html

## Host Attack Detection Based on XGBoost and Community Discovery

ZHU Yuan-Qing, LI Sai-Fei, LI Hong-Zhe

(School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China)

**Abstract**: In a large-scale network, the security threats faced by the host are becoming increasingly diverse. With the rapid rise of technology based on machine learning to detect malicious files, the ability to detect malware has been greatly improved, and it has also forced adversaries to change their attack strategies. Among them, the “Living off the land” strategy achieves malicious behavior by calling operating system tools or automated management programs that perform tasks. Threat detection can find suspicious behavior in the context of parent and child processes. The parent-child process chain and the related events derived from it are regarded as an undirected graph, and the supervised learning XGBoost algorithm is used for weight distribution to generate an undirected weighted graph. Finally, a community discovery algorithm is employed to identify larger attack sequences from the graph. The above algorithm is verified on the simulated attack dataset of MIRTE ATT & CK.

**Key words**: host attack; XGBoost; community discovery; graph analysis

根据赛门铁克在 2019 年发布的互联网安全报告[1]来看, “Living off the land, Lotl”的攻击技术存在上升的趋势, 已然成为目前网络安全威胁中的主流模式. 此外根据该报告所述的情况, 这种类型的攻击在 2018 年飙升了 78%. 报告中还提到 2018 年发现的 4 个新的威胁群体中, 有两个是使用 Lotl 技术被发现的. 近些年来越来越多的威胁群体喜欢使用这类技术[1]. Lotl 技术将其活动隐藏在合法的系统进程中, 让攻击者保持足够的隐匿, 例如大量使用 PowerShell、Cmd 等工具. 这些程序作为系统工具存在, 其本身并不具有主动攻击性, 攻

① 基金项目: 四川省科技计划 (2021YJ0372, 2019ZDZX0007); 中央高校基本科研业务费专项 (2682019CX63)
Foundation item: Science and Technology Program of Sichuan Province (2021YJ0372, 2019ZDZX0007); Special Fund of Fundamental Research Funds for the Central Universities of China (2682019CX63)
收稿时间: 2021-02-22; 修改时间: 2021-03-28; 采用时间: 2021-04-06

击者利用其两面性来进行威胁. 2017 年 Petya 的变种勒索软件大规模袭击世界各国组织. 其利用的漏洞与勒索软件 WannaCry 利用的漏洞相同, 区别是 Petya 通过使用 Windows 管理工具 WMI、命令行等工具远程执行一系列系统级的命令来实现相关功能[2]. 在此过程中攻击者使用系统工具执行既定策略达到目的, 将恶意行为隐藏在系统活动中逃避了现有安全产品的检测. 虽然这些行为不会被安全产品探测到, 却被系统日志记录了起来. 研究从主机日志提取恶意行为的模型对现有安全问题极具现实意义.

## 1 引言

虽然分析用户系统日志中的威胁和入侵不是新的趋势, 却是入侵检测中研究最多的领域之一. 现阶段针对主机日志中的恶意行为研究主要集中在以下的几种研究方向. 研究方向一是通过构造检测规则或使用启发式的算法来检测日志数据. 近年来 MIRTE 公司推出了 ATT & CK 模型, 其目标是创建网络攻击中已知的攻击策略、战术与技术的详尽列表, 对每一种战术的意义影响、利用方式以及检测数据来源做了详尽阐述[3]. 研究人员可以根据该框架的知识制定每种技术的检测规则[4]. 文献 [5] 中利用检测规则提取出日志中的恶意活动, 映射到攻击策略, 抽象出策略层的场景图. 使用祖先节点覆盖的概念来评估节点之间依赖强度, 实现了不同攻击技术之间的关联, 进而提取出其中的攻击序列. 文献 [6] 中利用系统日志构建起源图, 使用规则检测出攻击的及其所处阶段, 通过信息流之间的相关性, 在图中将恶意事件关联成攻击路径.

上述文献中基于规则来发现多步攻击的研究, 检测精度高, 误报率低, 但基于规则的检测对于未知威胁的检测能力受限. 因此, 主机日志中的恶意行为研究方向二, 对事件日志压缩找出其中攻击序列. 文献 [7] 通过对系统日志进行压缩, 删除其中未对系统造成持久影响的事件, 例如进程运行时产生的没有被其他进程访问的临时文件, 提取出日志中的重要行为. 文献 [8] 进一步提出动态污点分析的日志压缩方案, 通过追踪污点在主机中的扩散, 将此过程中的调用保存下来达到压缩目的, 分析后可得到恶意行为[9].

研究方向三: 有研究人员将机器学习与日志分析结合, 利用机器学习强大的分类能力寻找日志中潜在的恶意模式. 文献 [10] 中利用逻辑回归等机器学习算法给日志社区中的恶意行为提取特征, 通过社区检测算法发现日志中的恶意社区, 完成攻击序列的发现. 机器学习的研究方法拥有对异常威胁模式强大的发现能力, 对于新型威胁有更好的检测能力.

综上机器学习方法的优势, 本文利用机器学习在 Sysmon[11] 日志的基础上提出一种基于来源图的框架, 关注日志中事件行为. 将多种事件日志统一提取解析[10], 具体提取内容第 2 节所述. 以进程为实体建立无向图, 结合图中进程与文件操作等事件, 使用分类效果更好的 XGBoost 算法进行有监督分类, 预测恶意事件概率值作为图中边的权重, 进而得到一个无向加权图, 使用社区发现算法找出其中的恶意攻击社区[12−14]. 最后在 MIRTE ATT & CK 提供的评估数据集[15,16] 上进行验证. 实验显示使用 XGBoost 结合社区发现的模型, 检测结果更稳定, 效果更好, 具体实验如下文详述.

## 2 攻击行为检测模型

对于 Lotl 类型的攻击行为, 其使用的是就地取材式的战术策略, 传统以恶意文件为载体的安全检测不再奏效, 因此研究的重点应转移到攻击行为上来. 对于每一条日志来讲, 其记录的是单一的动作行为或事件信息. 为了关注攻击行为的上下文关系, 我们使用基于图的模型来表达这种关系. 所以日志中提取的事件数据以图的数据模型进行存储, 其示意图如图 1 所示.

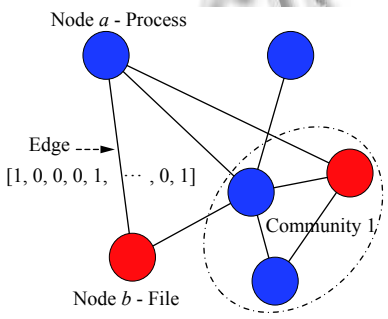



图 1 日志实体无向图示意图

节点 (node): 正在建模的日志对象实体 (例如: 进程名、文件名等);

边 (edge): 对象实体之间关系特征向量 (例如: 父子进程、文件操作等);

元数据 (matedata): 节点、边的特性或参数 (例如: 进程 ID、命令行、哈希值等).

### 2.1 日志解析与关联

整个算法模型的基础是日志中事件数据建立的社

区网络图, 之后的边权重分配与社区发现也是在此基础上对数据进行处理. Sysmon 作为一款微软官方提供的日志工具[11], 监控系统的动态行为, 生成事件日志. 从 Sysmon 日志数据中解析出图中节点所需字段如表 1 所示. 主要包括进程创建 (process creation)、进程终止 (process termination)、文件创建 (file create) 与删除 (file delete) 等事件记录. 在所有事件的类型中又可以划分为关联事件与信息事件, 其中的关联事件是指提供两个实体之间联系的事件, 主要包括内容如表 2 所示.

表 1 预定义日志实体格式

| 字段 | Event ID | 字段 | Event ID |
|---|---|---|---|
| Hostname | 主机 | CurrentDirectory | 1 |
| UtcTime | 1/5/11/23 | Company | 1 |
| ProcessGuid | 1/5/11/23 | Product | 1 |
| Image | 1/5/11/23 | IntegrityLevel | 1 |
| ParentProcessGuid | 1 | Hashes | 1/23 |
| ParentImage | 1 | TargetFilename | 11/23 |
| CommandLine | 1 | CreationUtctime | 11 |
| ParentCommandLine | 1 | User | 1 |
| OriginalFileName | 1 | EventID | — |

表 2 关联事件列表

| Event ID | 事件名 | 关联字段 |
|---|---|---|
| Event ID 1 | Process Creation | ProcessId & ParentProcessId |
| Event ID 5 | Process Termination | ProcessId |
| Event ID 11 | File Create | ProcessId & TargetFilename |
| Event ID 23 | File Delete | ProcessId & TargetFilename |

表 2 的关联事件中 Event ID1 与 Event ID 11 的日志格式如图 2 所示, 更多类型日志详细格式可查阅官方文档[11]. 其中 Event ID 1 记录的是进程创建事件, 主要的日志字段包括进程 ID (ProcessGuid)、父进程 ID (ParentProcessGuid)、创建时间戳 (UtcTime)、进程映像 (Image)、父进程映像 (ParentImage)、进程原始文件名 (OriginalFileName)、命令行 (CommandLine) 以及进程哈希值 (Hashes) 等字段; EventID 11 跟踪的是文件创建事件, 其主要字段包括进程 ID (ProcessGuid) 和目标文件名 (TargetFilename) 等.

根据表 2 中列出的关联事件日志类型及其关联字段, 将表 2 中的关联事件提取出统一的日志实体格式, 模型以 ProcessGuid 字段作为日志实体的节点字段, 其他提取的相关日志字段作为该节点的元数据. 提取出如表 1 的预定义日志实体字段及该字段的来源.

多类型的日志格式经过统一提取后, 形成表 1 所示日志实体格式, 此时的日志实体集合作为无向图 $G_0$ 中的节点集合 $V$.

日志解析之后通过观察节点集合 $V$ 中任意两个日志实体(*Node a*, *Node b*)的元数据字段之间存在的关系, 例如两个实体是否存在父子进程、兄弟进程的关系, 进程映像文件是否相同, 路径是否存在关联等等特征, 选取出如表 3 中所示的 20 维关系特征. 迭代产生节点之间的关系向量集合, 作为无向图 $G_0$ 中的边集合 $E$. 在 Sysmon 日志中 ProcessID 字段存在复用情况, 使用唯一值 ProcessGuid 来表示进程实体.

```
{
  Process Create:
  UtcTime: 2020-05-02 02:55:56.157
  ProcessGuid: {47ab858c-e13c-5eac-a903-000000000400}
  ProcessId: 8524
  Image: C:\ProgramData\victim\â€®cod.3aka3.scr
  FileVersion: -
  Description: -
  Product: -
  Company: -
  OriginalFileName: -
  CommandLine: "C:\ProgramData\victim\â€®cod.3aka3.scr" /S
  CurrentDirectory: C:\ProgramData\victim\
  User: DMEVALS\pbeesly
  LogonGuid: {47ab858c-dabe-5eac-f331-370000000000}
  LogonId: 0x3731F3
  TerminalSessionId: 2
  IntegrityLevel: Medium
  Hashes:SHA1=4B7FA56A4E85F88B98D11A6E01869
  ParentProcessGuid: {47ab858c-dac4-5eac-f202-000000000400}
  ParentProcessId: 4440
  ParentImage: C:\Windows\explorer.exe
  ParentCommandLine: C:\windows\Explorer.EXE
}
```

(a) EventID 1

```
{
  File created:
  RuleName: -
  UtcTime: 2020-05-02 02:57:00.933
  ProcessGuid: {47ab858c-e13c-5eac-a903-000000000400}
  ProcessId: 8524
  Image: C:\ProgramData\victim\â€®cod.3aka3.scr
  TargetFilename: C:\Users\pbeesly\Downloads\monkey.png
  CreationUtcTime: 2020-05-02 02:57:00.933
}
```

(b) EventID 11

图 2 EventID 1 和 EventID 11 的日志格式

关于表 3 中实体间的关系向量, $f0$ 和 $f19$ 表示事件时间关系, 实体 $a$、$b$ 之间的时间差是否小于间隔 $\Delta$, 是则该项为 1, 否则为 0. $f1$–$f4$ 项表示实体 $a$ 和 $b$ 之间的进程关系, 以 $f1$、$f2$ 为例, 当实体 $a$ 的 ProcessGuid 等于实体 $b$ 的 ProcessGuid, 说明两个实体属于同一个进程, $f2$ 表示实体 $a$、$b$ 属于同一个父进程创建的兄弟进程, 如果成立该项为 1, 否则为 0; 同理 $f5$ 表示两个日志实体是否来自同一主机; $f6$–$f9$ 表示实体中进程映像的关系; $f10$–$f13$ 表示两个实体之间进程命令的关系; $f14$ 表示进程实体的原始文件名, 该项可以检测进程重命名情况; $f15$ 表示当前日志实体所在路径, 对于恶意活动中的文件扫描行为, 提取文件操作之间的关系; $f16$ 表示实体间的哈希值是否一致; $f17$、$f18$ 表示文件

创建过程中路径与文件名的关系.

经过上述过程的关联, 我们将实体集合 $V$ 中的任意两个实体间关系提出, 作为两个实体($Node\ a$,$Node\ b$)之间的边. 所以该图可表示为式 (1), 其示意如图 1 所示.

$$G_0 = (V, E) \tag{1}$$

其中,

$$E \subseteq \left\{\{a,b\} : (a,b) \in V^2, a \neq b\right\} \tag{2}$$

表 3 日志实体间关系特征

| 特征 | 关系 |
|---|---|
| $f0$ | \|$a$.UtcTime – $b$.UtcTime\| <Δ |
| $f1$ | $a$.ProcessGuid == $b$.ProcessGuid |
| $f2$ | $a$.ParentProcessGuid == $b$.ParentProcessGuid |
| $f3$ | $a$.ParentProcessGuid == $b$.ProcessGuid |
| $f4$ | $a$.ProcessGuid == $b$.ParentProcessGuid |
| $f5$ | $a$.HostName == $b$.HostName |
| $f6$ | $a$.Image == $b$.Iamge |
| $f7$ | $a$.ParentImage == $b$.ParentImage |
| $f8$ | $a$.ParentImage == $b$.Image |
| $f9$ | $a$.Image== $b$.ParentImage |
| $f10$ | $a$.CommandLine == $b$.CommandLine |
| $f11$ | $a$.ParentCommdLine == $b$.ParentCommdLine |
| $f12$ | $a$.ParentCommdLine == $b$.CommdLine |
| $f13$ | $a$.CommdLine == $b$.ParentCommdLine |
| $f14$ | $a$.OriginalFileName == $b$.OriginalFileName |
| $f15$ | $a$.CurrentDirectory == $b$.CurrentDirectory |
| $f16$ | $a$.Hashes == $b$.Hashes |
| $f17$ | $a$.TargetFilename.Path == $b$.TargetFilename.Path |
| $f18$ | $a$.TargetFilename.Name == $b$.TargetFilename.Name |
| $f19$ | \|$a$.CreationUtctime – $b$.CreationUtctime\| <Δ |

## 2.2 恶意社区发现

日志解析之后生成了事件关联图$G_0$, 其中包括正常良性日志社区与恶意日志社区. 因为恶意行为与良性行为在父子进程、兄弟进程、文件操作等行为上存在差异, 受到社交网络分类的启示, 在事件关联图$G_0$上使用社区发现算法发现$G_0$中的恶意行为社区.

社区检测 Louvain 算法[17] 适用于大规模网络结构中快速发现其中的社区模块. 该算法社区发现效果较好, 复杂度较低和计算速度更快. Louvain 算法本身是一种基于模块度的社区发现算法. 其中模块度的概念由 Newman[18] 提出, 作为一种在复杂网络图中评估社区划分情况的数学指标, 其定义为式 (3):

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j) \tag{3}$$

其中, $A_{i,j}$表示顶点 $i$ 与 $j$ 之间的权重, $k_i$与$k_j$分别表示连接到点 $i$ 与点 $j$ 的权重和, $m$ 是图中所有节点的边权重总和, $\delta(c_i,c_j)$的 $c_i$ 与 $c_j$ 表示顶点 $i$ 和 $j$ 所属的社区, 其中如果 $c_i$=$c_j$, 则$\delta(c_i,c_j)=1$, 反之$\delta(c_i,c_j=0)$[17]. 之后进一步将式 (3) 化简, 可以得到式 (4) 所示形式, 其中$\Sigma_{in}$表示为网络图社区内部边权值总和, $\Sigma_{tot}$为与网络图社区内节点相连的外部边的权值总和.

$$Q = \sum_c\left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2\right] \tag{4}$$

算法过程主要分为两个阶段: 首先算法初始化认为网络中每个节点都是一个独立的模块. 在第一阶段的迭代过程中, 对于每一个顶点 $i$, 将其依次分配到相邻顶点 $j$, 计算式 (5) 所示的分配前后模块度变化$\Delta Q$. 如果此过程中最大的$\Delta Q>0$, 则将顶点 $i$ 分配到最大$\Delta Q$对应顶点 $j$ 的社区, 否则不做任何变化, 直到所有节点所属的社区不再发生变化.

$$\Delta Q = \left[\frac{\Sigma_{in}+k_{i,in}}{2m} - \left(\frac{\Sigma_{tot}+k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \tag{5}$$

算法的第二阶段迭代之前, 先将第一阶段中的子社区整体看作一个超节点, 该社区内的节点到其他社区内节点的连接看作为超节点的边. 由此创建出新的图后, 将算法第一阶段应用于新图. 直到整个图的模块度不再发生变化后, 算法结束, 此时就产生了最终的社区. 上述过程如算法 1 所示.

算法 1. 社区发现 Louvain 算法

输入: network $G$=($V$, $E$)
输出: network $G'$=($V'$, $E'$)

```
1. G.V:{node 0, node 1,⋯, node n}
2.   for i in G.V do
3.       for j in iadjacent_node do
4.           根据式 (5) 计算 Max (ΔQ)
5.           if Max (ΔQ)>0 do
6.               i⇒community of j
7.           if G change do
8.               G.V={community 0, ⋯, community m}
9.               goto step 2
10.          else do
11.              G'=G
```

## 2.3 权重生成

由式 (4) 可知, 社区内的权重越大于社区间的权重时, 模块度越大. 如果将良性日志和恶意日志分为不同的社区, 社区内部边的权重越大于社区间边的权重, 两种不同社区的发现效果才会越好. 但如果直接在 2.1 节中生成的无向图中使用社区检测时. 社区检测算法此

时认为图中每一条边的权重都是 1. 观察发现恶意行为社区与良性社区之间存在着很多边的连接, 这种情况降低了社区内的集群密度, 影响到了社区发现算法的有效性. 针对这种情况我们对边的特征向量应用分类算法, 得出向量隶属于恶意社区的概率, 以此概率作为该边的权重, 将 2.1 节生成的无向图$G_0$转换为无向加权图$G_1$. 使得图中边满足式 (6) 所示条件:

$$w_{\text{in}} \cdot |e_{\text{in}}| >> w_{\text{tot}} \cdot |e_{\text{tot}}| \tag{6}$$

其中, $w_{\text{in}}$表示的是社区内边$e_{\text{in}}$对应的权重, $w_{\text{tot}}$表示的是社区之间边$e_{\text{tot}}$对应的权重. 在使用 XGBoost 算法对事件关联图$G_0$中的边进行分类训练之前, 假设$G_0$共有 $n$ 条边, 记为$e_i = x_i, y_i, i \in [1,n]$, 其中$x_i$作为边的特征向量, $y_i$为该边对应的标签, 其取值如式 (7) 所示:

$$y_i = \begin{cases} 1, & x_i \in e_{\text{in}} \\ 0, & x_i \in e_{\text{tot}} \end{cases} \tag{7}$$

XGBoost 是一个树集成模型, 将多个弱分类器一起组合成一个强分类器, 这也是其相比传统模型分类能力更强的原因. 将多颗树的预测结果进行结合, 获得比单一的树更加优越的泛化能力. 在 XGBoost 中的目标函数如式 (8) 表示:

$$obj^t = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{k=1}^{t} \Omega(f_t) \tag{8}$$

目标函数由第一部分训练损失和第二部分正则化组成, 式中$\hat{y}_i^{(t)}$是第 $t$ 次的预测值, 其与 $t$–1 次预测值存在式 (9) 所示关系, $\Omega(f)$是正则化部分, 定义了每棵树的复杂度, 其具体定义参考文献 [19]:

$$\hat{y}_i{}^{(t)} = \hat{y}_i{}^{(t-1)} + f_t(x_i), f_k(x) = w_{q(x)} \tag{9}$$

目标函数通过二阶泰勒展开沿着梯度下降的方向优化损失, 优化速度大大加快. 将损失函数$l\left(y_i, \hat{y}_i^{(t-1)}\right)$关于$\hat{y}_i^{(t-1)}$的一阶偏导和二阶偏导带入进一步化简[19], 最终得到的目标函数为:

$$obj^t \approx -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{10}$$

式 (10) 也被称为打分函数, 是衡量树结构好坏的标准, 其值越小, 表示这样的树结构越好. 有了上述的打分函数, 最终就可以求出树的最佳结构. 在本次实验中的二分类问题使用对数损失函数, 如式 (11) 所示:

$$loss = \sum_i \left[ y_i \log_2\left(1 + e^{-\hat{y}_i}\right) + (1 - y_i) \log_2\left(1 + e^{\hat{y}_i}\right) \right] \tag{11}$$

对式 (7) 所示标记的 $n$ 条训练边采用 $k$ 折交叉验证的方式进行学习, 以此提高学习效果, 最终将 XGBoost 算法对于每条边属于社区内部边的预测作为该边的权重, 从而完成无向加权图$G_1$的建立. 此时$G_1$将作为社区发现的输入, 利用社区发现 Louvain 算法发现$G_1$存在的恶意社区.

### 2.4 复杂度分析

上述模型的整体复杂度分析分为 3 步. 首先, 在日志解析与关联阶段中需要提取任意两个日志之间特征关系, 其时间复杂度为$O(M^2)$, 其中 $M$ 为日志实体数. 紧接着在权重分配过程中, XGBoost 使用精确贪心算法建树的时间复杂度是$O(Kd\|x\|_0 \log_2 n)$, 但可以将特征项排序并存储为 Block 结构对算法进行优化, 在计算过程中复用该结构, 而不用每次都对特征项排序. 经过这样的优化后, 特征项排序复杂度将会均分到权重分配过程中, 此时该部分时间复杂度变为$O(Kd\|x\|_0)$. 上述中 $K$ 为 XGBoost 中树的总数, $d$ 为树的最大深度, $\|x\|_0$表示训练边集中非缺失值样本的数量. 模型最后的恶意社区发现中, 算法中先将节点合并为社区相当于对图中边进行遍历, 时间复杂度为$O(n)$, 之后为得到最大模块度不断迭代 $k$ 次, 所以算法总复杂度为$O(nk)$. 上述 $n$ 表示图中边向量个数. 整个模型复杂度为 3 个阶段的复杂度叠加所得$O(M^2 + Kd\|x\|_0 + nk)$. 作为对比实验使用 SVM、逻辑回归与 XGBoost 在权重生成阶段进行对比, 其时间复杂度对比使用算法运行时间指标来衡量, 具体实验对比如第 4 节所述.

## 3 实验

实验使用 Python 实现, 利用已有的科学计算库对 MITRE ATT & CK 提供的 APT29 以及 APT3 的仿真攻击结果进行检测. 在 ATT & CK 提供的日志数据集[15,16]中, 主要是使用 Sysmon 采集的主机日志数据. 实验的整体流程如图 3 所示.

在实验的阶段 1 中, 从如图 2 中所示的原始日志中将实验涉及的关联事件日志解析为表 2 的日志实体格式. 此时原始日志清洗预处理之后, 所有的日志实体格式统一. 为了阶段 2 的关联特征提取, 经过多次实验确定了表 3 中的实体间关系, 其中具体每条关系的意义如上文所述. 由于特征提取过程中只关注于实体间关系并归一化, 所以提取到的特征已进行了泛化. 解析到的日志实体与提取到的特征关系向量, 组成了实验

阶段 2 的输入数据无向图$G_0$, 其结构如图 1 所示.

实验阶段 2 中主要是解决无向图$G_0$中边权重分配问题. 为了提高社区内的集群聚集度, 减少社区间的联系. 将阶段 1 中提取到的特征向量作为输入, 利用 2.3 节中 XGBoost 算法进行分类. 实验关注 XGBoost 算法对于特征向量的预测值, 以此预测值作为边的权重. 为了提高 XGBoost 的学习效果, 使用 10 折交叉验证的学习方法对训练集进行优化. 作为对比, 实验使用传统的学习算法逻辑回归和 SVM 算法[20] 与之对比, 具体实验效果对比如下文所示. 在实验阶段 2 之后, 无向图$G_0$经过权值分配生成无向加权图$G_1$, 如图 4 所示.

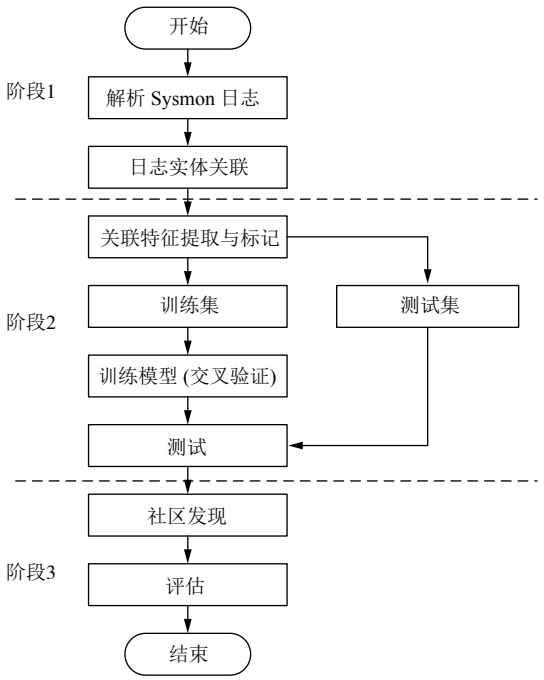



图 3 关键阶段与工作流程图

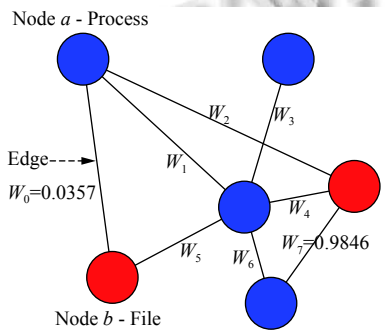



图 4 无向加权图$G_1$示意图

实验阶段 3 中, 将社区检测 Louvain 算法应用在图$G_1$上, 利用 2.2 节中所述的模块度对图中存在的社区进行发现. 找出原始日志中隐藏的恶意日志社区. 阶段 3 的输出为图 5 所示实体名与所属社区的对应表.

```
ret = {
    “Node”：node a;
    “belong”: belong;
}
```

图 5 阶段 3 输出格式

## 4 评估

根据在实验过程的最终输出的实体名与所属社区的对应表, 对于分类结果通常有以下几种评价标准, 准确率 (*Accuracy*)、精确率 (*Precision*)、召回率 (*Recall*) 以及 *F*1 分数, 但 *F*1 分数指标综合算法的精确率指标和召回率指标, 评价更为全面. 上述指标的计算公式如式 (12) 到式 (15) 所示:

$$Accuracy = \frac{TP+FN}{TP+FP+TN+FN} \tag{12}$$

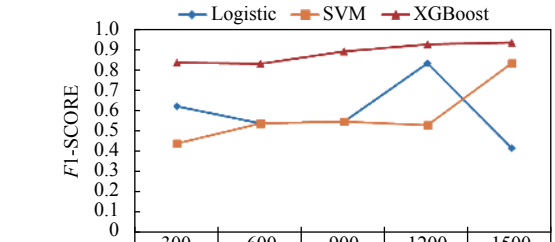
$$Precision = \frac{TP}{TP+FP} \tag{13}$$

$$Recall = \frac{TP}{TP+FN} \tag{14}$$

$$F1 = \frac{2\times Precision\times Recall}{Precision+Recall} \tag{15}$$

实验采用 *F*1 分数对实验结果进行评估, 其中 XGBoost 算法最终社区发现效果最好, *F*1 分数明显高出另外两种算法, 受表 3 中特征提取时间间隔$\Delta$影响不明显, 性能稳定, 可以有效的检测出恶意社区中父子进程及文件操作行为. 另外逻辑回归和 SVM 算法最后的社区检测效果不是很好, 在 APT29 与 APT3 模拟数据集的实验过程中 *F*1 分数受特征提取时间间隔$\Delta$的影响波动较大. 具体数据如图 6, 图 7 所示.

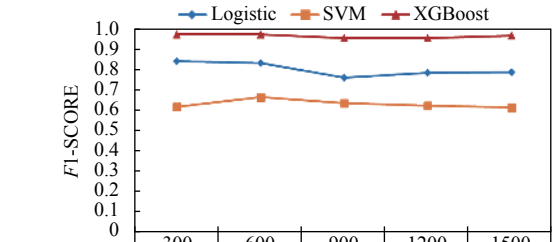
在对 3 种不同的权重分配算法最后的社区检测精度进行评估之后, 考虑到算法复杂度的对比, 实验采用运行时间指标来对比评估本文模型与另外两种算法的复杂度. 控制模型日志预处理与社区发现阶段相同, 所以复杂度的差距在于权重分配阶段. 因此实验在 APT29 数据上对比了不同的特征提取间隔下 3 种权重分配算法的运行时间. 如 2.4 节所述, 由于 XGBoost 算法使用 Block 块对特征项排序优化, 在运行上相比于另外两种算法更高效. 对比数据如图 8 所示, 时间单位为秒.




| | 300 | 600 | 900 | 1200 | 1500 |
|---|---|---|---|---|---|
| Logistic | 0.6282 | 0.5438 | 0.5527 | 0.8391 | 0.4218 |
| SVM | 0.445 | 0.5443 | 0.5534 | 0.5362 | 0.837 |
| XGBoost | 0.8413 | 0.8338 | 0.8962 | 0.9291 | 0.9403 |

时间间隔

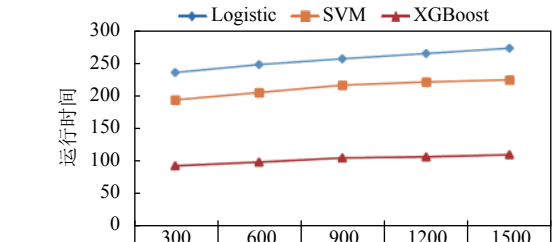
图 6 APT29 社区发现 *F*1 分数图




| | 300 | 600 | 900 | 1200 | 1500 |
|---|---|---|---|---|---|
| Logistic | 0.8484 | 0.839 | 0.7682 | 0.7909 | 0.7953 |
| SVM | 0.6237 | 0.6714 | 0.6408 | 0.6272 | 0.619 |
| XGBoost | 0.9802 | 0.9801 | 0.9612 | 0.9611 | 0.9748 |

时间间隔

图 7 APT3 社区发现 *F*1 分数图




| | 300 | 600 | 900 | 1200 | 1500 |
|---|---|---|---|---|---|
| Logistic | 236.63 | 248.42 | 257.28 | 265.66 | 273.49 |
| SVM | 194.54 | 205.82 | 217.14 | 222.31 | 224.94 |
| XGBoost | 93.88 | 99.98 | 106.05 | 107.6 | 110.69 |

时间间隔

图 8 APT29 权重分配算法运行时间对比图

最后在实验中发现的部分恶意行为序列. 例如, (1) 双击带有恶意负载的 3aka3.doc 文档, 生成 3aka3.scr 恶意进程, 之后派生出 Cmd 与 Powershell 进程建立 C2 通道连接到 192.168.0.5, 完成对受害主机 1 的初始登录. (2) 在受害主机上进行文件收集等工作, 利用 Powershell 执行文件收集与压缩命令对后缀为 xlxs、doc、pdf、png 等的文件进行遍历、加密和压缩生成 Draft.Zip 与 working.zip. (3) 生成新的更高权限的 Powershell 进程, 执行文件与进程的发现任务, 利用 Get-Process 等命令发现主机中的进程, 加载 Netapi32.dll 来探测系统接口. (4) 删除上述过程中生成的 Draft.Zip 与 working.zip 文件, 销毁收集活动痕迹. 之后结合横向移动等一系列步骤完成对主机的侵害.

## 5 结论

通过实验评估证明了 XGBoost 与社区检测算法结合在检测主机日志恶意行为中的价值. 关注父子进程链及其文件等操作, 提取出日志实体之间的社区图关系, 通过社区发现算法检测其中恶意社区. 在边权重分配过程中使用 XGBoost 集成学习算法与逻辑回归、SVM 算法对比, 结果表明基于 XGBoost 与社区发现的主机攻击行为检测模型具有良好的检测效果, 对于 Lotl 技术有更好的检测能力. 但是, 在实验过程中日志关系之间的生成与训练过程可以进一步优化, 使得模型运行更加快速和高效.

## 参考文献

1 Symantec. Internet security threat report 2019. https://docs.broadcom.com/doc/istr-24-2019-en. [2020-12-22].

2 Filar B, French D. ProblemChild: Discovering anomalous patterns based on parent-child process relationships. arXiv: 2008.04676, 2020.

3 MITRE ATT & CK. ATT & CK matrix for enterprise. https://attack.mitre.org/. [2021-01-05].

4 Mavroeidis V, Jøsang A. Data-driven threat hunting using sysmon. Proceedings of the 2nd International Conference on Cryptography, Security and Privacy. New York: ACM, 2016. 82–88. [doi: 10.1145/3199478.3199490]

5 Milajerdi SM, Gjomemo R, Eshete B, *et al*. HOLMES: Real-time APT detection through correlation of suspicious information flows. Proceedings of 2019 IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2019. 1137–1152. [doi: 10.1109/SP.2019.00026]

6 Yu H, Li AP, Jiang R. Needle in a haystack: Attack detection from large-scale system audit. Proceedings of 2019 IEEE 19th International Conference on Communication Technology. Xi'an: IEEE, 2019. 1418–1426. [doi: 10.1109/ICCT46805.2019.8947201]

7 Lee KH, Zhang XY, Xu DY. LogGC: Garbage collecting audit log. Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. New York: ACM, 2013. 1005–1016. [doi: 10.1145/2508859.2516731]

8 贲永明, 韩言妮, 安伟, 等. 一种基于污点追踪的系统审计日志压缩方法. 信息安全学报, 2020, 5(5): 30–42. [doi: 10.19363/J.cnki.cn10-1380/tn.2020.09.03]
9 阮琳琦. 保持依赖关系的实时日志压缩系统的设计与实现 [ 硕士学位论文 ]. 杭州: 浙江大学, 2020. [doi: 10.27461/d.cnki.gzjdx.2020.000649]
10 Pei KX, Gu ZS, Saltaformaggio B, *et al*. HERCULE: Attack story reconstruction via community discovery on correlated log graph. Proceedings of the 32nd Annual Conference on Computer Security Applications. New York: ACM, 2016. 583–595. [doi: 10.1145/2991079.2991122]
11 Mark Russinovich, Thomas Garnier. Sysmon-windows sysinternals microsoft docs. https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon. [2020-11-28].
12 李春强, 夏伟. 基于 Windows 日志分析的终端安全研究. 网络空间安全, 2018, 9(9): 70–77. [doi: 10.3969/j.issn.1674-9456.2018.09.014]
13 孙文贺. 面向主机的攻击行为分析研究 [ 硕士学位论文 ]. 北京: 北京交通大学, 2019.
14 徐嘉涔, 王轶骏, 薛质. 网络空间威胁狩猎的研究综述. 通信技术, 2020, 53(1): 1–8. [doi: 10.3969/j.issn.1002-0802.2020.01.001]
15 Mitre-attack, mitre-attack/attack-arsenal-GitHub. https://github.com/mitre-attack/attack-arsenal. [2020-11-15].
16 OTRF. OTRF/mordor-GitHub. https://github.com/OTRF/mordor. [2020-11-15].
17 Blondel VD, Guillaume JL, Lambiotte R, *et al*. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008, 2008(10): P10008. [doi: 10.1088/1742-5468/2008/10/P10008]
18 Newman MEJ. Fast algorithm for detecting community structure in networks. Physical Review E, 2004, 69(6): 066133. [doi: 10.1103/PhysRevE.69.066133]
19 Chen TQ, Guestrin C. XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM, 2016. 785–794. [doi: 10.1145/2939672.2939785]
20 Cortes C, Vapnik V. Support-vector networks. Machine Learning, 1995, 20(3): 273–297. [doi: 10.1007/BF00994018]