# 缩写

|  |  |  |
| --- | --- | --- |
| 缩写 | 全称 | 描述 |
| APT | Advanced Persistent Threat | 高级持久性威胁 |
| ATT&CK | Adversarial Tactics, Techniques, and Common Knowledge | 对抗性的战术、技术和常识 |
| C&C | Command and Control |  |
| CTI | Cyber threat intelligence | 网络威胁情报 |
| CVSS | Common Vulnerability Scoring System | 常见的漏洞评分系统 |
| DARPA | Defense Advanced Research Projects Agency | 国防高等研究计划署 |
| ETW | Event Tracing for Windows | 窗口的事件跟踪 |
| GPM | Graph Pattern Matching | 图形模式匹配 |
| HSG | High-level Scenario Graphs | 高级方案图 |
| IDS | Intrusion Detection System | 入侵检测系统 |
| IOC | Indicator of compromise | 折衷指标 |
| IPS | Intrusion Prevention System | 入侵预防系统 |
| LoC | Lines of Code | 代码行数 |
| PLC | Programmable Logic Controller | 可编程序逻辑控制器 |
| RAT | Remote Administration Tool | 远程管理工具 |
| SIEM | Security Information and Event Management | 安全信息和事件管理 |
| TC | Transparent Computing Project (by DARPA) | 透明计算项目(由DARPA提供) |
| TTP | Tactics, Techniques, and Procedures | 战术、技术和程序 |

# 概述

## 基本概念

### provenance graphs

通过Audit logs生成provenance graphs（溯源图），再根据溯源图进行以下安全行为：

* APT detection 高效持久威胁
* attack scenario reconstruction 攻击场景重建
* cyber threat-hunting 网络威胁探测

### tag-based approach

用于识别subjects, objects and events

### high-level scenario graphs

高级场景图，大概是对当前活动（campaigns）的总结图

### graph pattern matching problem

图匹配问题，把cyber threat-hunting转化为图匹配问题，而这个图匹配，就是query graph（查询图，源自CTI，网络威胁情报）和provenance graph（溯源图，源自内核Audit Logs）。

## 主要研究内容

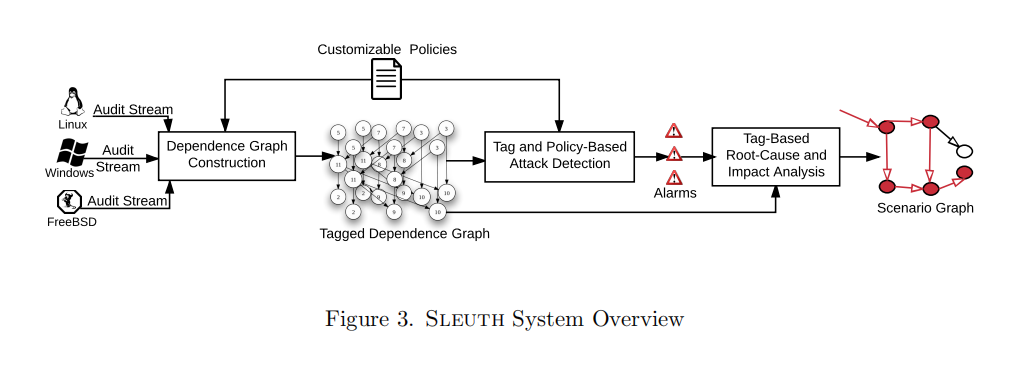
Sleuth，通过内核审计日志构建图（provenance graph？），以实现及时攻击场景重建。

Holmes，在APT检测中，把一堆不可靠告警，转化为强的检测信号。

Poirot，搜索分析溯源图，揭示攻击活动。

# SLEUTH：基于COTS审计数据的实时攻击场景重构

## overview



来自不同操作系统的审计数据被处理成平台独立的图，这个图是攻击检测、关联分析和场景重构的基础。

SLETTH亮点就是全自动，有以下contribution：

1. 将依赖图（紧凑的）存储在内存中，使效率高效。
2. 使用tag-based方法描述Subject、Object和Event。
3. 新的算法，进行root-cause identifification and impact analysis，这里包含backward analysis和forward search算法
4. 可定制的策略框架
5. 提供实验评估（experimental evaluation），更快、更高效、更低的误报率。

## 主存依赖图

图作为数据结构存储依赖关系，一种可行的方法是使用图数据库，例如Neo4J、Titan, 但是除非内存足够大，否则这些数据库在运行许多图算法时会受到限制，而本文提出了一种更省内存的依赖图设计。

图的构成为实体（entity，对应图数据库中的顶点）和事件（event，对应图数据库中的边,）,其中：

实体包含Subject和Objects两种：

1. Subjects

进程，属性值包括：process id（pid）、命令行、所有者（owner）以及代码和数据的标签

1. Objects

实体，例如文件、pipes、网络连接。属性值包括：名称、类型（文件、pipe、socket等）、所有者和标签

事件，用于subjects和objects之间或者两个subjects之间用read､connect､execve来表示。

## 标签和攻击检测

使用标签来评估objects和subjects的trustworthiness和sensitivity。主要是从以下3个方面：

1. 起源：object和subject直接祖先的标记。
2. 系统先验知识：就是对一些应用程序的预了解。
3. 行为：观察sujects行为，并与预期行为比较。

SLEUTH有一个默认的保守策略，把所有的Subject、Object和Event都进行前向、后向分析，该策略可能会导致一些良性事件被错误地识别为恶意事件（over-tainting），但绝不会漏掉攻击。、

### 标签设计

标签分两种，可信度标签和机密性标签。

**trustworthiness tags** （t-tags，可信度标签）有如下定义，可信度依次降低：

1. Benign authentic tag：良性可靠标签，为数据和代码分配该标签，其来源（source）为良性，且可靠性可被验证的。
2. Benign tag：良性标签，来源（source）为良性，但可靠性未被验证的。
3. Unknown tag：未知标签，来源未知，默认策略。

对于t-tags，代码和数据是分开的，比如code t-tags和data t-tags。

***confifidentiality tags***（c-tags，机密性标签）有如下定义，机密性依次降低：

1. Secret：高度敏感信息，例如登陆凭证、私钥。
2. Sensitive：敏感信息，例如系统漏洞。
3. Private：涉及隐私，但构不成安全威胁。
4. Public：可以被公开的数据。

对于c-tags，只有代码，即data c-tags。

针对系统已经存在的subject和object，使用tag initialization policies初始化标签，后续发起的subject和object使用tag propagation policies定义标签，攻击的检测使用detection policies.

### 基于标签的攻击检测

大概意思关注点不应该在应用程序的行为，而是大部分攻击行为的高级目标，比如后门植入、数据窃取。结合了攻击者的motive（动机）和means（手段），如果Audit data中的某条事件可能帮助攻击者实现高级目标，这将为攻击行为提供理由。

在大多数高级攻击活动中的典型步骤：

1. 部署、调用攻击者代码。
2. 替换、修改重要文件，比如/etc/passwd或ssh keys。
3. 窃取敏感数据

攻击检测策略定义：

1. Untrusted code execution（不受信任的代码执行）：当一个高级别code t-tag的subject，执行或调用低级别t-tags的object时，发出警报。
2. Modifification by subjects with lower code t-tag（低级别code t-tag的subject的修改行为），当低级别code t-tag的subject修改一个高级别t-tag的object时，发出警报。修改可能涉及文件的内容或属性。
3. Confifidential data leak（机密数据泄露）：当untrusted subject窃取敏感数据时，发出警报，也就是具有sensitive (data) c-tag 和 unkonwn code t-tag的subject在网络中执行写操作时。
4. Preparation of untrusted data for execution（为执行准备不可信数据）：即具有unknown code t-tag的subject，操作使对象可执行，比如chmod、mprotect。

需要注意的是，攻击者的means并不会因为数据或代码经过了多个中间媒介之后而被“稀释”。比如：对于不受信任的代码执行策略来说，来自未知网站上的数据并不仅仅直接执行才触发警报，而是当未知数据经历下载、提取、解压缩、编译之后再加载执行的，都会触发警报。随后可通过往后的溯源分析，找到漏洞的第一步动作。

第三方detector input可以容易的嵌入SLEUTH中，具体做法是第三方detector input把可疑的subject设置unknown code t-tag，然后后续的检测策略就可以继承使用了。

在静态环境中，不会发生不可信代码执行，因为静态环境不允许新代码产生。我们希望可以连续地更新和升级，又不希望用户下载未知代码，因此后续会描述如何支持标准化的软件更新机制。

## 策略框架