

Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心



面向攻击溯源的日志处理技术

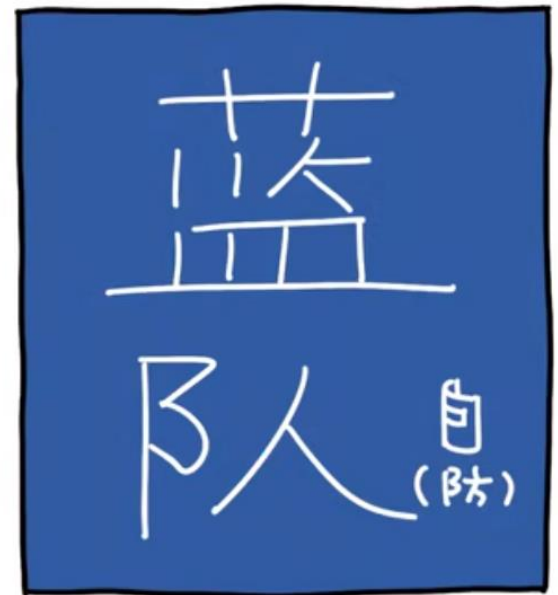
硕士研究生 关迎丹

2022年8月21日

- 背景简介
- 基本概念
- 算法原理
- 总结
- 参考文献

- 预期收获
 - 1.了解攻击溯源场景
 - 2.了解缓解依赖爆炸的日志处理技术原理
 - 3.总结技术缺陷

- 原始日志数据庞大
 - 单台主机上采集的原始数据达5GB/天
 - 实际场景中一次攻击产生的原始数据达PB级
- 攻击溯源场景
 - 安全运维时遭受攻击后溯源
 - 攻防演练时蓝队溯源反制
 -



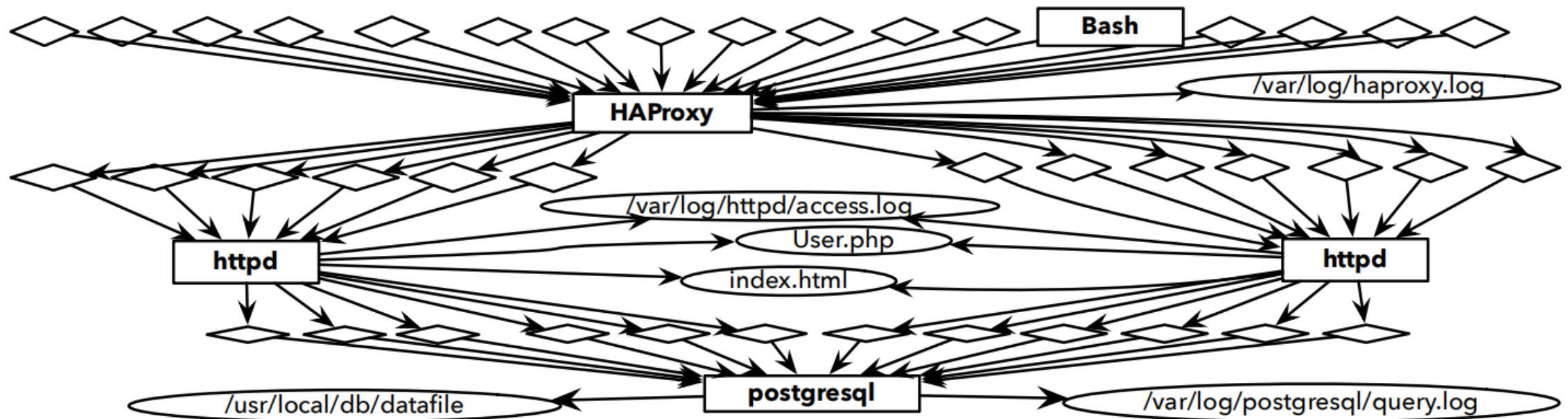


基本概念

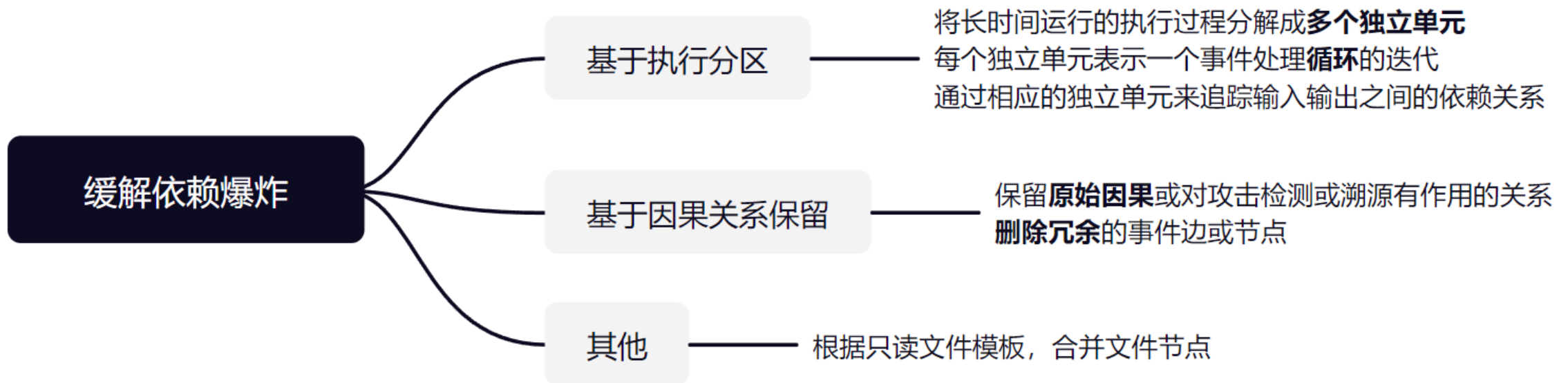
- 系统溯源图
 - 由系统审计日志生成的图，其中顶点表示系统主体（进程）和系统对象（文件等），边表示因果关系，同时包含时间戳或事件类型等信息
- 后向跟踪（backward tracing）
 - 从单个检测点（例如，一个可疑的文件）开始，向后跟踪找到溯源图中**对检测点有因果影响**的所有节点（攻击源）
- 前向跟踪（forward tracing）
 - 从单个检测点开始，前向跟踪找到溯源图中因果**依赖于检测点**的所有节点。
- 攻击溯源
 - 执行前向跟踪和后向跟踪找到**攻击源**与其他受攻击影响事件

- 系统日志溯源的局限性

- 系统日志：尽可能记录实体间所有可能的依赖关系
- 依赖爆炸：上下游实体之间的信息依赖随着溯源深度的增加呈现指数级爆炸



- 缓解**依赖爆炸**问题的现有日志处理技术



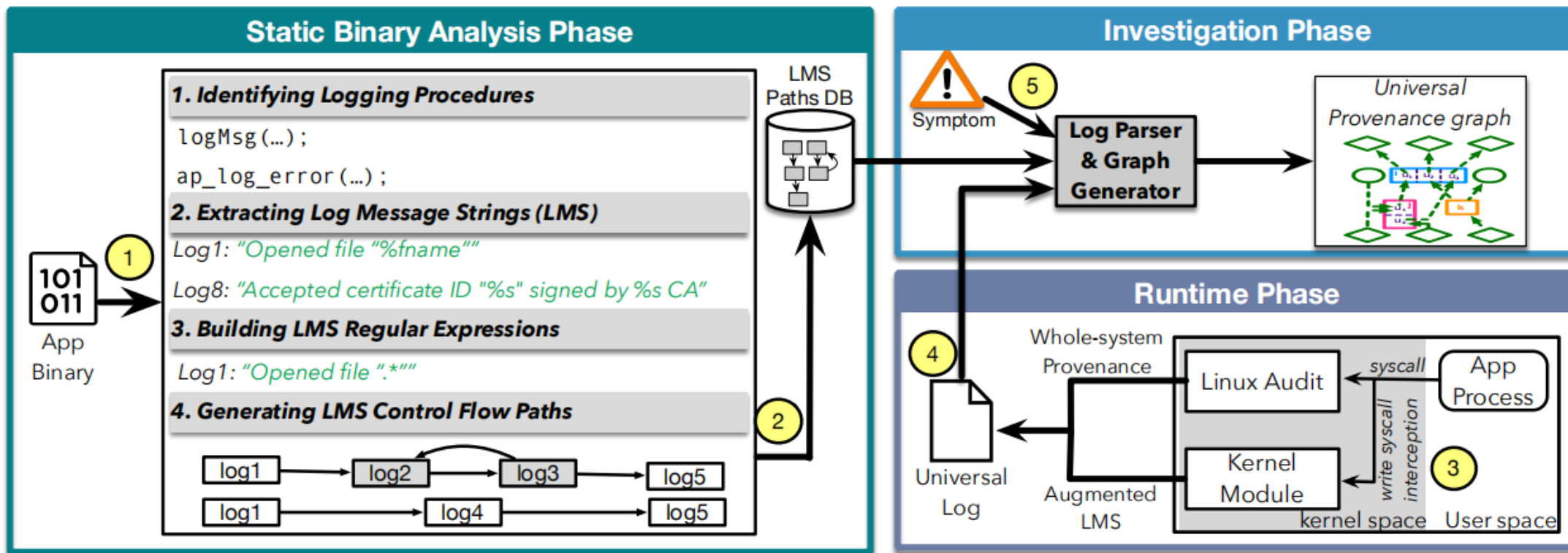


算法原理

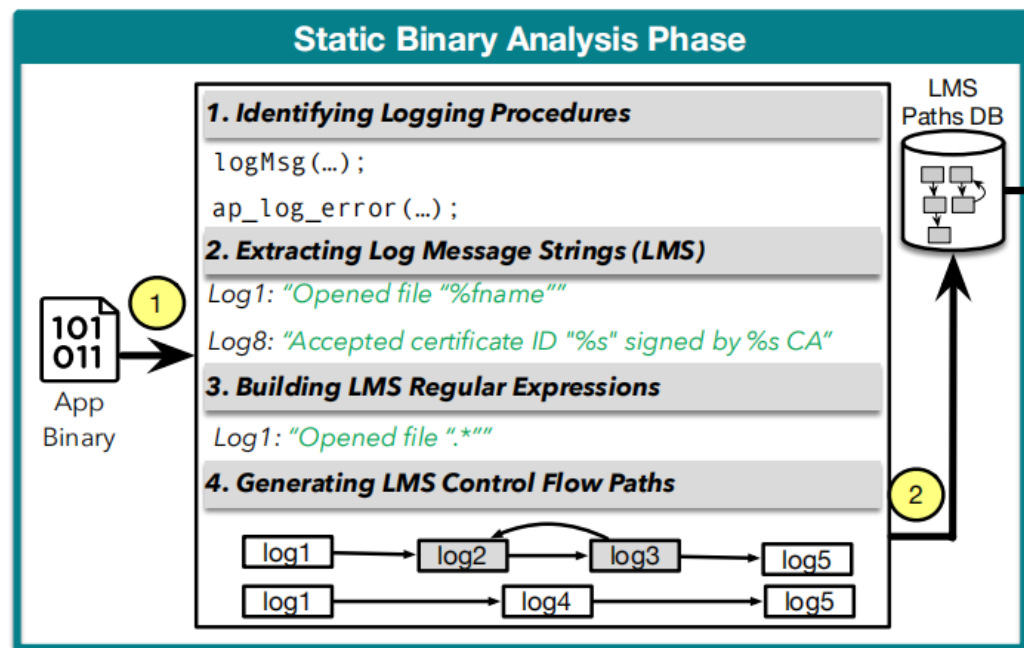
T	结合应用程序日志获取精简全局溯源图，缓解依赖爆炸
I	全局日志文件、应用程序二进制文件
P	1.静态分析应用程序二进制文件，获取 LMS控制流路径 2.使用PID/TID对应审计日志和应用日志获得全局日志 3.利用LMS控制流路径对全局日志进行 执行分区 划分，获得全局溯源图
O	全局溯源图

P	如何利用应用程序日志解决 依赖爆炸 问题
C	应用程序二进制文件可提取控制流图
D	应用程序日志和系统日志的对应
L	2020 NDSS

- OmegaLog原理图
 - 静态二进制分析阶段、运行阶段、调查阶段



- 静态二进制分析阶段
 - 识别日志记录程序
 - 获取常见日志依赖库的日志记录函数
 - 根据路径（例如/var/log/）获取日志记录函数
 - 抽取LMS(Log Message Strings)
 - 使用Angr的FastCFG功能获取简易控制流图
 - 检查各基本块中的日志记录函数
 - 从函数中抽取具体的LMS
 - 构建LMS正则表达式
 - 格式符替换（"%s"替换为".*")
 - LMS控制流路径生成



encounter in OpenSSH is

```
PAM: password from user %.12s accepted.
```

After extraction, that yields the regex

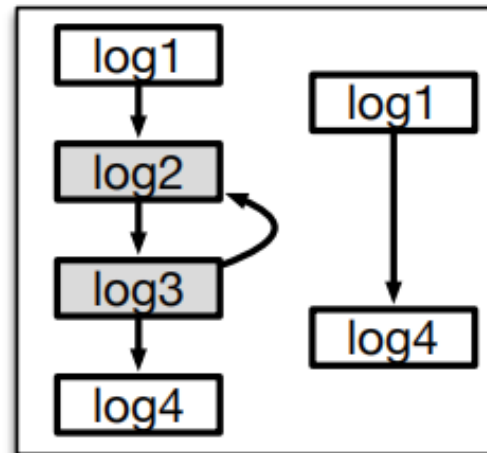
```
PAM: password from user .* accepted.
```

- 静态二进制分析阶段

- LMS控制流路径生成

- 将路径识别任务划分为多个局部遍历功能模块，对每个模块生成子图
 - 遇到一个包含LMS的基本块时，该块将被添加到路径中，并遍历它的传出边
 - 解析call/jump指令关联子图构建完整的LMS控制流路径

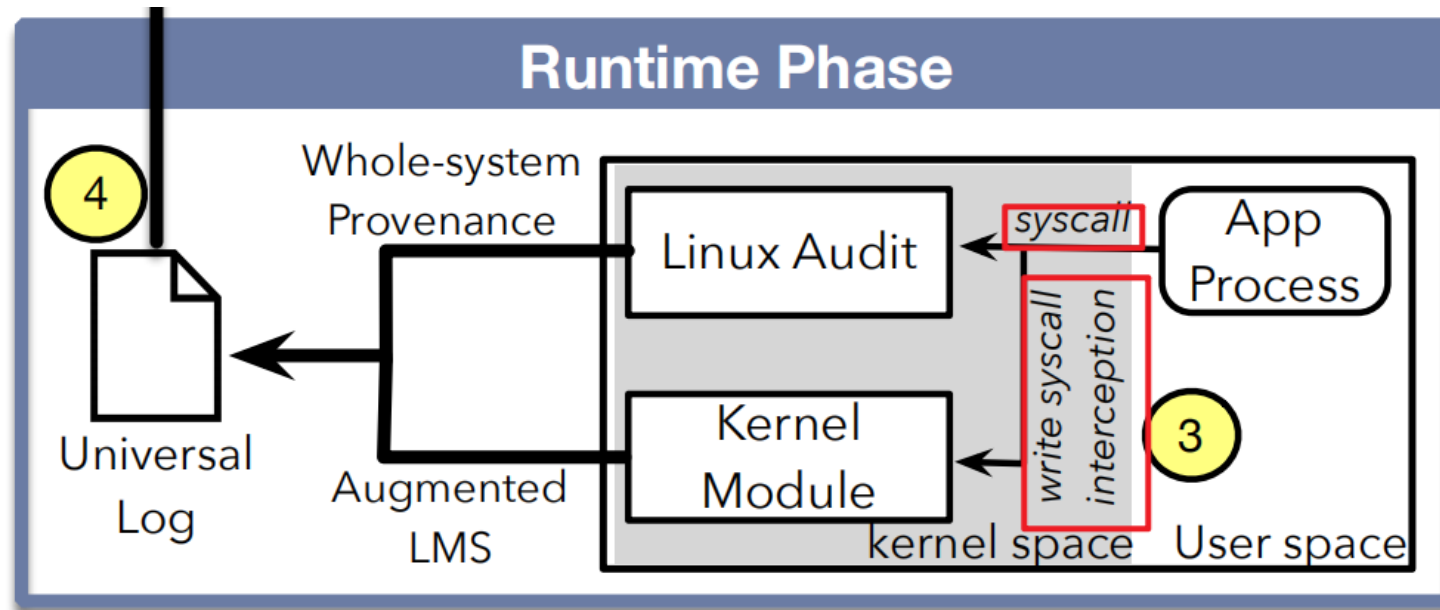
```
log("Server started"); // log1
while(...) {
  log("Accepted Connection"); // log2
  ... /*Handle request here*/
  log("Closed Connection"); // log3
}
log("Server stopped"); // log4
```



log2和log3存在循环结构，不止出现一次

- 运行阶段

- 系统日志和应用程序日志是**如何对应**的? -进程/线程的PID/TID
- **内核模块**将应用程序日志信息写入对应系统日志, 构成全局日志
- 全局日志包含应用程序级事件信息、系统级事件及时间戳等



- 调查阶段

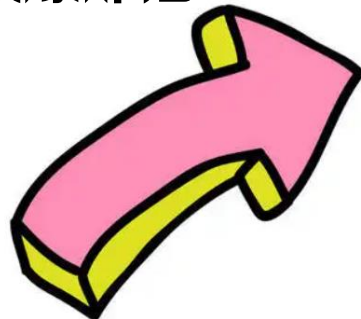
- 现有的输入包括:

- 全局日志文件(L_{uni})
- LMS控制流路径($Paths_{lms}$)

- 给定攻击标记事件(e_s), 获得精简的**全局溯源图** (UPG)

- 精简: 缓解依赖爆炸后

算法原理



初始化执行单元标志 $endUnit == False$

对于 每个全局日志文件 L_{uni} 中 发生在**标记事件** e_s 之间的每个事件 e

如果 e 是应用程序事件:

根据 e 在数据库中匹配一条候选LMS(LMS_{cand})

LMS_{cand} 与数据库中的有效 $Paths_{lms}$ 进行匹配

LMS_{state} 指向 $Paths_{lms}$ 的下一个状态

若 LMS_{cand} 匹配到 $Paths_{lms}$ 的末端:

$endUnit == True$

如果 $endUnit == True$ (表示单元结束):

事件 e 加入单元 $eventUnit[Pide]$

将单元 $eventUnit[Pide]$ 所有事件加入全局溯源图 G

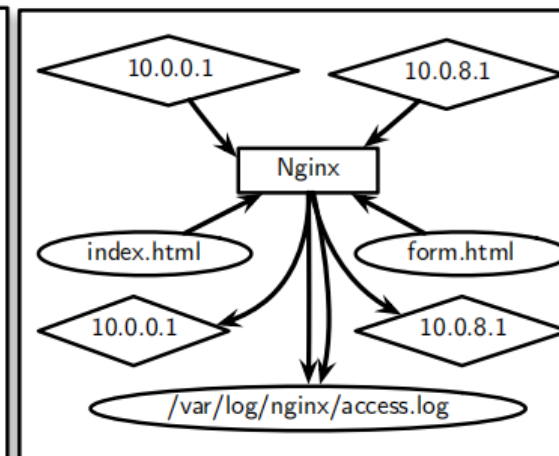
$endUnit == False$ 且清空单元 $eventUnit$

如果不是以上任一情况:

事件 e 加入单元 $eventUnit[Pide]$

• 实验设置

- 实验环境: Intel Core(TM) i7-6700 CPU @ 3.40 GHz 内存 32 GB, Ubuntu 16.04
- 系统溯源日志采集: Linux Audit
- 采集指令: clone, close, creat, execve, exit, fork, open, rename, unlink.....
- 应用程序: PostgesSQL, HAProxy, Nginx, Postfix, wget, Redis.....

<p>- <i>Receives HTTP request</i></p> <p>- <i>Reads index.html</i></p> <p>- <i>Sends HTTP</i></p> <p>- <i>Logs event in access.log</i></p> <p>- <i>Receives HTTP request</i></p> <p>- <i>Reads form.html</i></p> <p>- <i>Sends HTTP</i></p> <p>- <i>Logs event in access.log</i></p>	<ol style="list-style-type: none">1. Socket_Read("10.0.0.1")2. FRead(index.html)3. Socket_Write("10.0.0.1")4. FWrite(access.log) 5. Socket_Read("10.0.8.1")6. FRead(form.html)7. Socket_Write("10.0.8.1")8. FWrite(access.log)	<ol style="list-style-type: none">1. [16/Apr/2019:20:21:56 +0100] "GET / index.html HTTP/1.1" 200 3804 "-" "Mozilla/5.0 (Windows NT 6.0; WOW64; rv:45.0) Gecko/20100101 Firefox/45.0" 2. [16/Apr/2019:20:21:56 +0100] "GET / form.html HTTP/1.1" 200 3804 "-" "Mozilla/5.0 (Windows NT 6.0; WOW64; rv:45.0) Gecko/20100101 Firefox/45.0"	 <pre>graph TD; I1{10.0.0.1} --> N[Nginx]; I2{10.0.8.1} --> N; N --> I3{10.0.0.1}; N --> I4{10.0.8.1}; N --> L([/var/log/nginx/access.log]); N --- I5([index.html]); N --- I6([form.html]);</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

(a) Execution

执行说明

(无实际数据, 仅参考)

(b) System Log

系统日志

(c) Application Log

应用程序日志

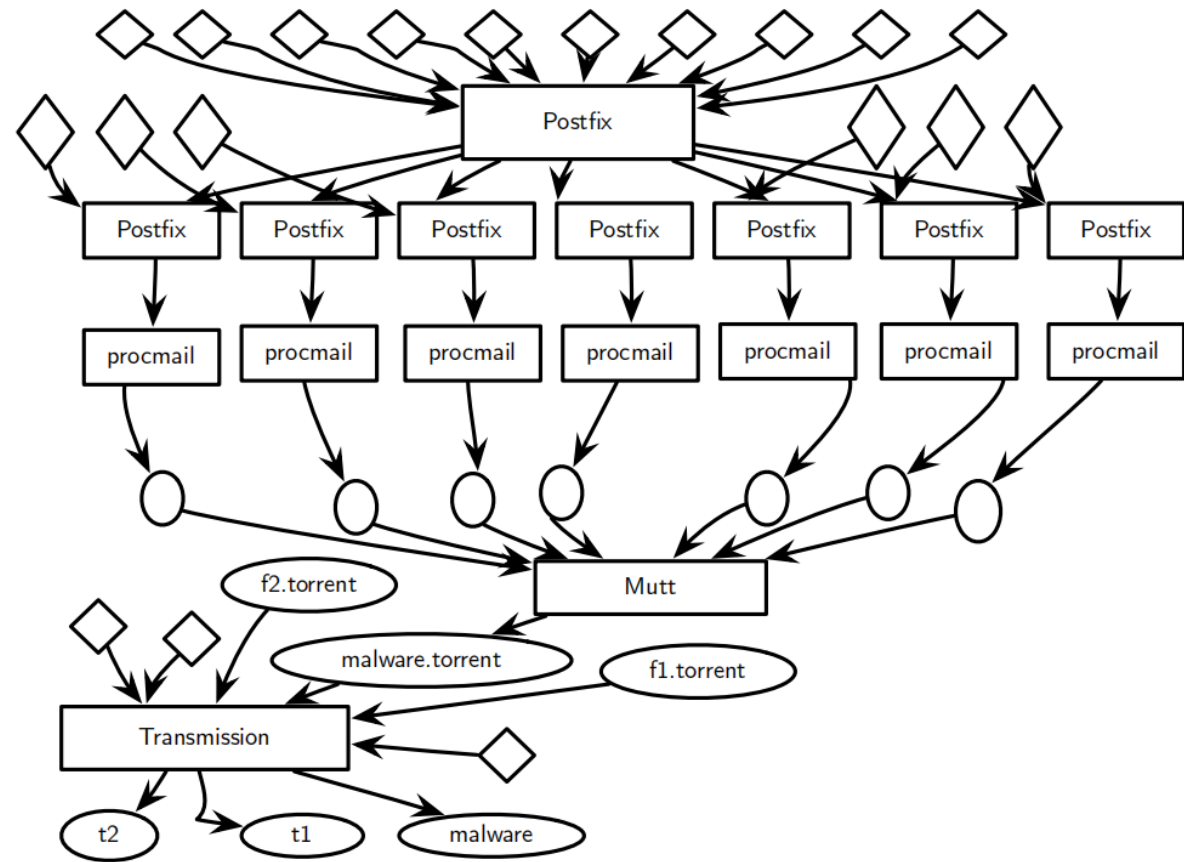
(d) System Provenance Graph

系统溯源图

以Nginx应用为例: Web服务收到两个HTTP请求

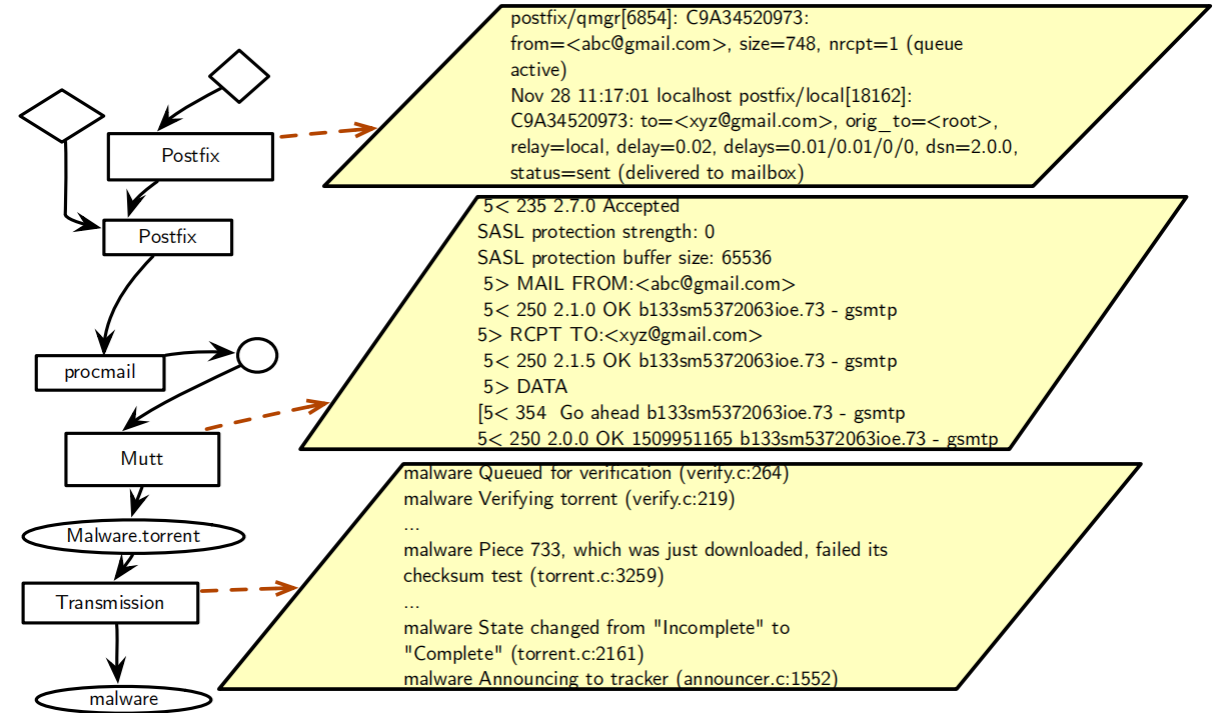
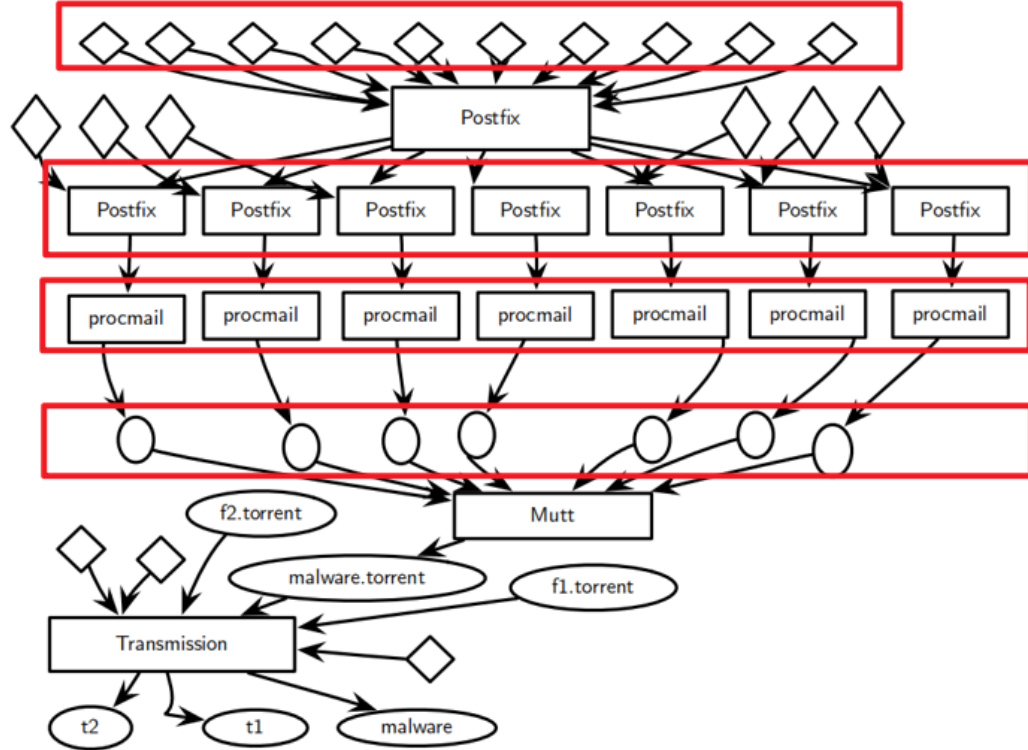
- 全局溯源图获取

- 溯源场景
- 员工Mutt客户端收到一封**钓鱼邮件**，其中提供了下载电影的种子.torrent
- 员工打开电子邮件，下载附加的.torrent文件，利用下载工具下载该电影
- 员工打开下载的电影文件（实际上是**恶意软件**），主机被植入后门
- 管理员发现异常运行，从.torrent文件（**攻击标志事件**）出发溯源攻击



Linux Audit 采集的系统日志构成的**系统溯源图**

- 全局溯源图获取
 - Postfix、Transmission执行分区



- 静态分析程序的计算代价和LMS的覆盖率
 - LMS生成需要12秒到1小时不等，LMSPs生成大多控制在1~8秒
 - 18个程序中的12个应用程序可达95%的覆盖率

Program	Binary Size (kB)	Log Level inside EHL	Avg. Time (sec)		Number of		Completeness	
			LMS	LMSPs	LMS	LMSPs	Callsites	Cov. %
Squid	64,250	IN+DE	831	46	64	157,829	70	91
PostgreSQL	22,299	IN+DE	3,880	258	3,530	4,713,072	5,529	64
Redis	8,296	INFO	495	7	375	34,690	394	95
HAProxy	4,095	IN+DE	144	4	53	13,113	56	95
ntpd	3,503	INFO	2,602	4	490	10,314	518	95
OpenSSH	2,959	IN+DE	734	4	845	11,422	869	97
NGINX	2,044	IN+DE	775	11	923	8,463	925	100
Httpd	1,473	IN+DE	99	2	211	3,910	211	100
Proftpd	1,392	IN+DE	201	4	717	9,899	718	100
Lighttpd	1,212	INFO	1,906	2	349	5,304	358	97
CUPSD	1,210	DEBUG	1,426	3	531	4,927	531	100
yafc	1,007	IN+DE	88	2	57	3,183	60	95
Transmission	930	IN+DE	102	2	178	5,560	227	78
Postfix	900	INFO	97	3	96	2,636	98	98
memcached	673	IN+DE	193	7	64	19,510	69	93
wget	559	INFO	200	3	84	3,923	275	31
thttpd	105	N/A	157	8	4	14,847	5	80
skod	47	N/A	12	0	25	115	25	100

- 静态分析程序的计算代价和LMS的覆盖率
 - postgresSQL、Transmission和wget（覆盖率分别为64%、78%和31%）
 - 三种程序使用了GNU的gettext（”_”操作符调用），导致解析失败

Program	Binary Size (kB)	Log Level inside EHL	Avg. Time (sec)		Number of		Completeness	
			LMS	LMSPs	LMS	LMSPs	Callsites	Cov. %
Squid	64,250	IN+DE	831	46	64	157,829	70	91
PostgreSQL	22,299	IN+DE	3,880	258	3,530	4,713,072	5,529	64
Transmission	930	IN+DE	102	2	178	5,560	227	78
Postfix	900	INFO	97	3	96	2,636	98	98
memcached	673	IN+DE	193	7	64	19,510	69	93
wget	559	INFO	200	3	84	3,923	275	31
thttpd	105	N/A	157	8	4	14,847	5	80
skod	47	N/A	12	0	25	115	25	100

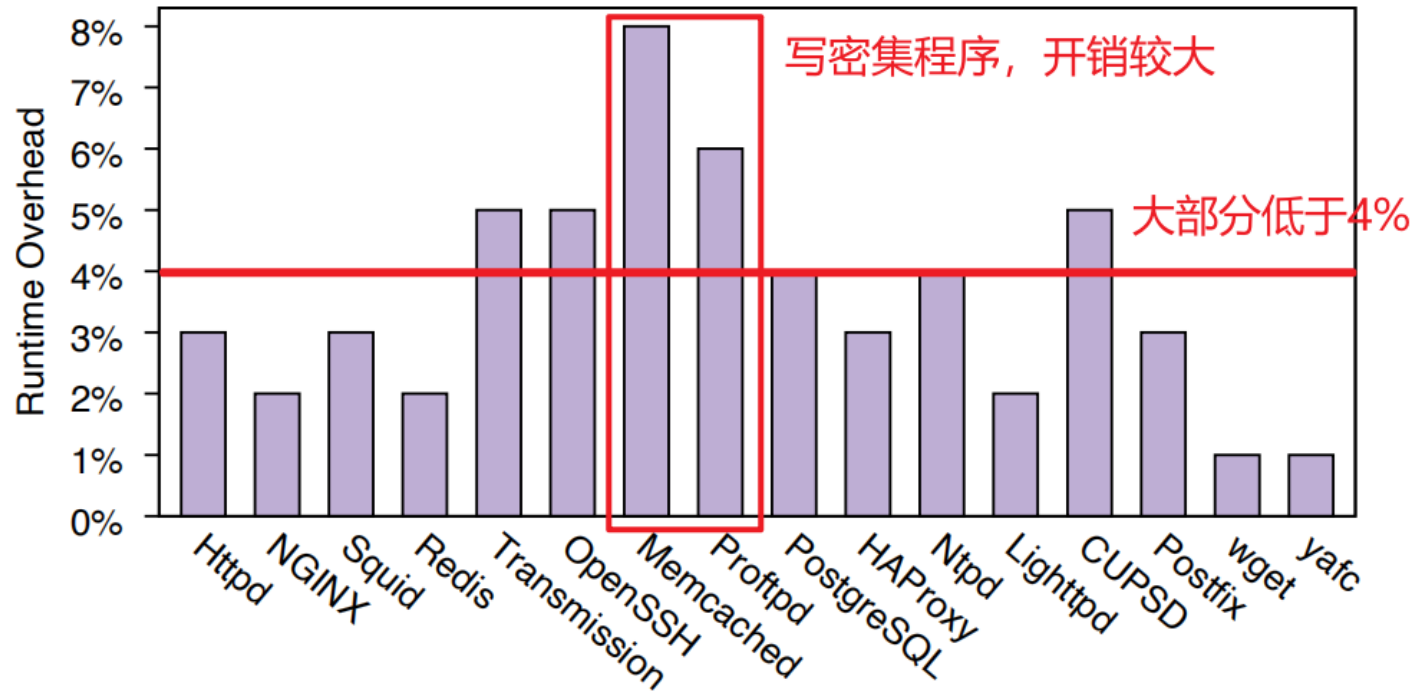
```
/* Below code from Transmission: /libtransmission/rpc-server.c */
tr_logAddNamedError(MY_NAME, _("Couldn't find settings key \"%s\""), str);
/* Below code from wget: /src/convert.c */
logprintf (LOG_VERBOSE, _("Converting links in %s... "), file);
/* Below code from PostGreSQL: /src/backend/commands/tablecmds.c */
default:
    /* shouldn't get here, add all necessary cases above */
    msg = _("\\""%s\" is of the wrong type");
    break; }
```

- 时间和空间开销

- 时间开销：除写密集程序外，平均运行时开销为4%

- 空间开销：

- 假设NGINX服务器每天接收100万个请求，并且每个请求生成一个事件
- 原始事件日志将为860 MB，本文方法则约12 MB，缩减率（约66x）



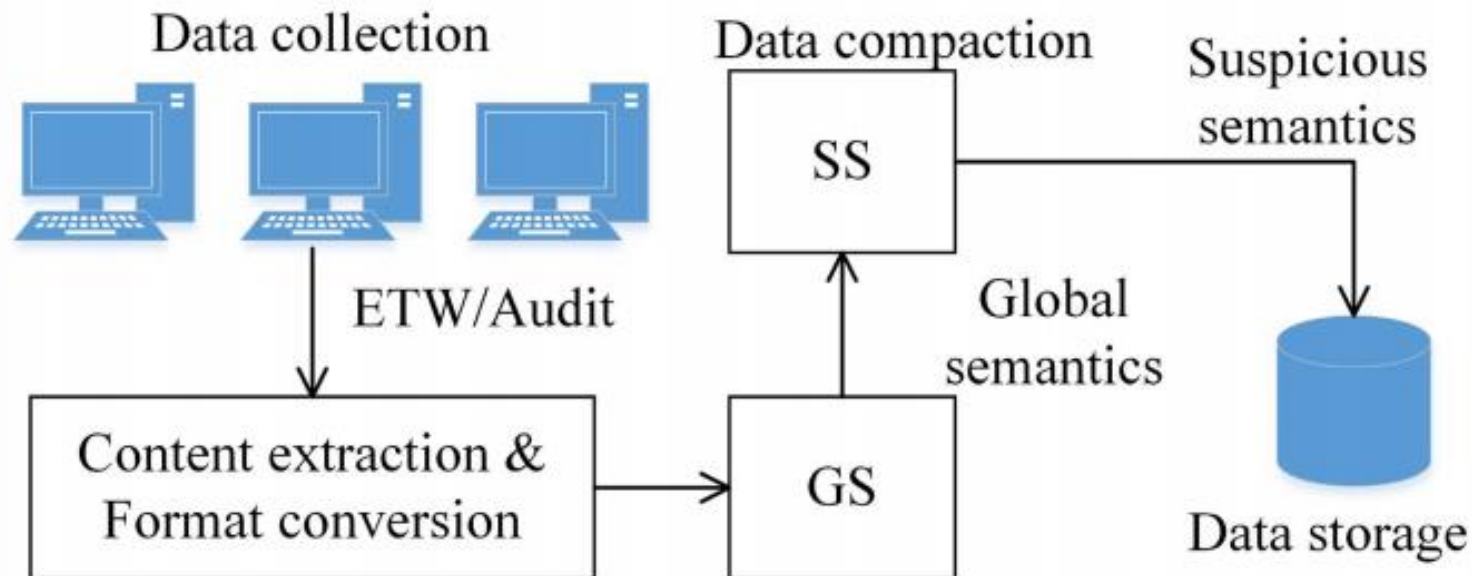


算法原理

T	缓解系统日志的 依赖爆炸 问题， 压缩 日志规模
I	系统日志
P	1.删除不影响全局语义的事件 2.执行上下文分析，仅保留可疑语义事件
O	压缩后的系统日志

P	如何解决 依赖爆炸 问题
C	日志中包含大量读写操作事件或网络事件
D	可疑语义事件定义
L	2021 SCI 1区

- 系统原理图
 - 数据收集、数据压缩、数据存储
 - GS:删除不影响**全局语义**的事件
 - SS:仅保留**可疑语义**事件



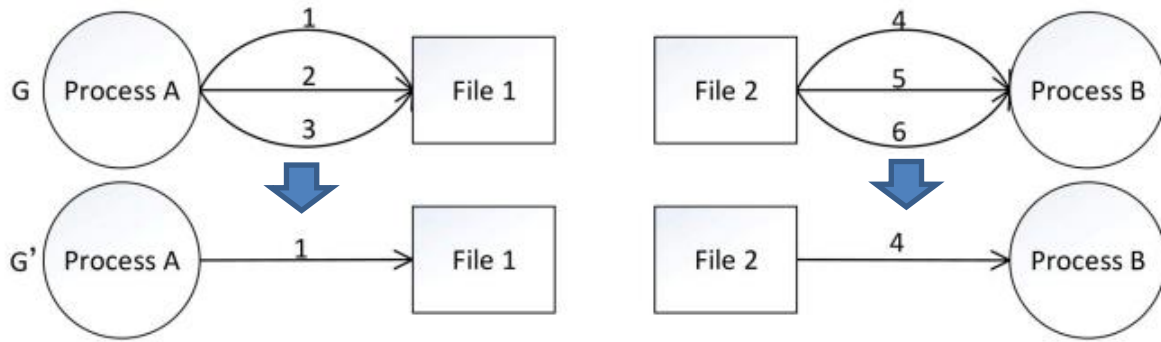
- GS:删除不影响**全局语义**的事件

- 全局语义:

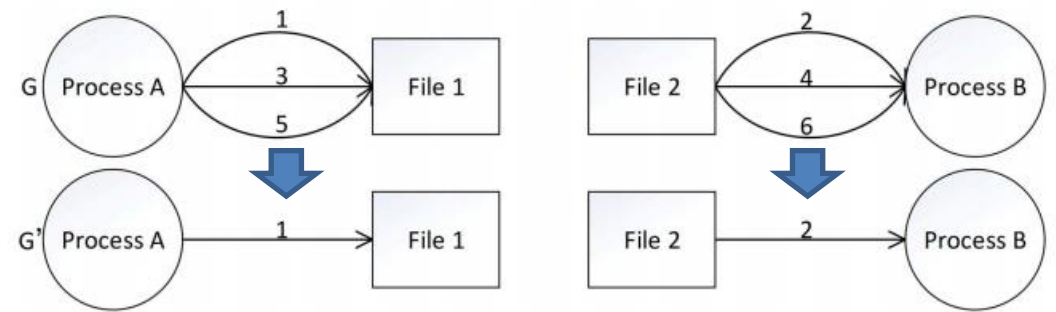
- 受一个源实体W影响的实体集Sem(W)

- 一般认为**“写”**操作会修改目标实体语义，**“读”**操作修改源实体语义

- 在源实体语义不变的情况下，源实体到目标实体的**信息流事件等价**，等价事件在检测、溯源时**语义重复**，可作为冗余事件删除



多个进程独立、**连续**地读写多个文件



多个进程独立地、**交替**地读写多个文件

• GS-对读/写事件的压缩

初始化缓存结构:

进程写过的文件集 `_process_write_cache:PID->{File}`

读取文件的进程集 `_file_read_cache:File->{PID}`

if 事件类型为**写文件**:

if PID 不在 `_process_write_cache` 键集合:

向 `_process_write_cache` 插入 `PID->{File}`

else if 文件不在 `_process_write_cache[PID]` 集合中:

向 `_process_write_cache[PID]` 插入 `file`

if 事件满足以上任一情况:

清空 `_file_read_cache[file]` // 文件语义改变

保存事件

else:

删除事件

if 事件类型为**读文件**:

if file不在 `_file_read_cache` 键集合:

向 `_file_read_cache` 插入 `file->{PID}`

else PID文件不在 `_file_read_cache[file]` 集合中:

向 `_file_read_cache[file]` 插入 `PID`

if 事件满足以上任一情况:

清空 `_process_write_cache[PID]` // 进程语义改变

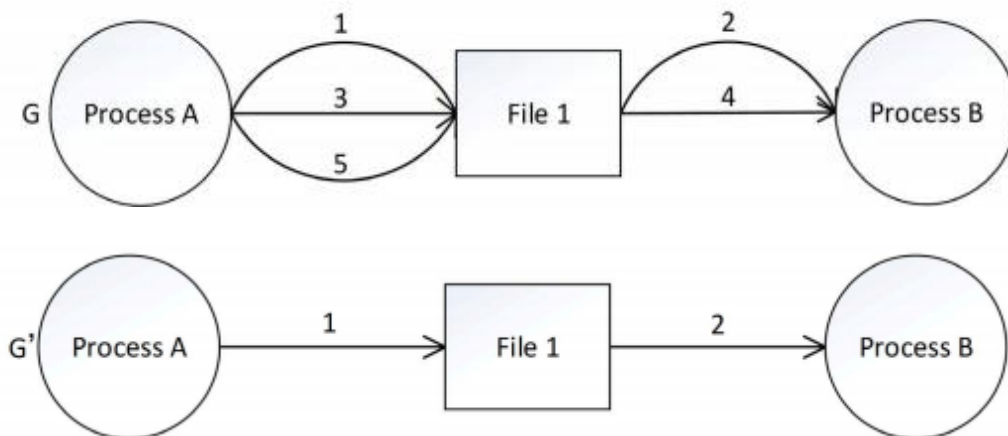
保存事件

else:

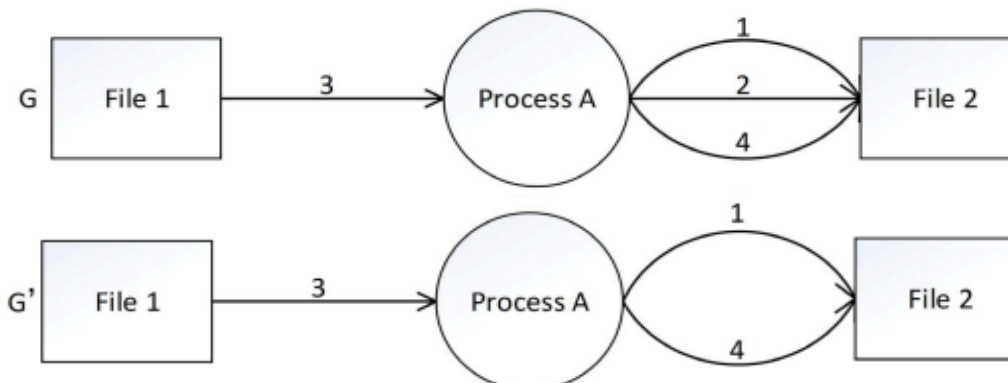
删除事件

- GS-对读/写事件的压缩

- 多个进程交替地读取和写入到同一个文件中



- 同个进程交替地读取和写入到多个文件中

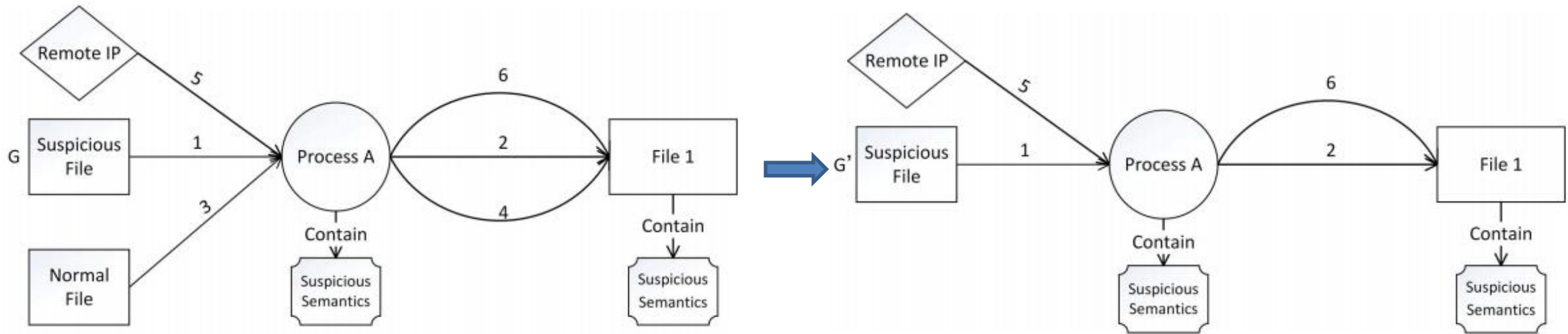


- **SS: 仅保留可疑语义事件**
 - 同一类型的事件由于不同的关联实体而具有不同的可疑程度
 - 通过实体上下文，可以确定事件是否可疑（即与攻击关联），删除攻击无关事件
 - 例如：进程从网络中下载数据、进程访问可疑文件等导致进程可疑

ID	Source entity	Event	Destination entity	Description
1	Network	Receive	Process	A process accesses data from the network, the process becomes suspicious
2	Process	Send	Network	A suspicious process sends data to the network
3	File	Read	Process	A process accesses a suspicious file, the process becomes suspicious
4	Process	Write	File	A suspicious process writes data to a file, the file contains suspicious semantics
5	Image	Load	Process	A process loads an image without a valid certificate and becomes suspicious
6	Process	Start/Fork	Process	One process is started by a suspicious process, and becomes suspicious

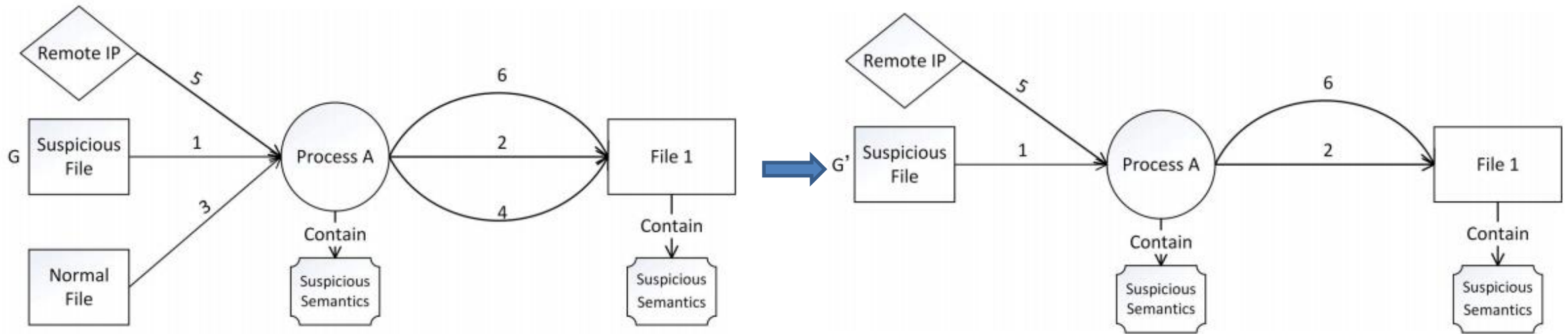
- SS: 仅保留可疑语义事件

- 同一类型的事件由于不同的关联实体而具有不同的可疑程度
- 通过实体上下文，可以确定事件是否可疑（即与攻击关联），删除攻击无关事件
- 例如：进程从网络中下载数据、进程访问可疑文件等导致进程可疑



- SS: 仅保留可疑语义事件

- 同一类型的事件由于不同的关联实体而具有不同的可疑程度
- 通过实体上下文，可以确定事件是否可疑（即与攻击关联），删除攻击无关事件
- 例如：进程从网络中下载数据、进程访问可疑文件等导致进程可疑



- 实验设置

- 实验环境：两台Windows(Win7)设备，两台Linux(Ubuntu 16.04)设备
- 红队模拟从2小时到2天不等攻击行动和良性背景日志

- 实验数据

- W-1和W-2为Windows设备日志，L-1和L-2为Linux设备日志，
- 注意：大部分为文件读写和网络操作

Dataset	Duration (hh:mm:ss)	File read	File write	Process/Thread	Network	Others	Total number of events	Attack
W-1	48:18:34	34.30%	6.01%	0.74%	55.64%	3.31%	1580927	No
W-2	2:41:46	44.59%	24.96%	1.56%	24.43%	4.46%	563981	Yes
L-1	2:31:12	49.49%	22.36%	0.02%	25.08%	3.05%	290326	No
L-2	3:53:21	58.20%	22.26%	0.17%	9.30%	10.07%	178917	Yes

- GS压缩效果

- 压缩比: 原事件数/压缩后事件数

Dataset	Overall compaction ratio	File read	File write	Process/Thread	Network
W-1	13.18×	12.03×	14.08×	15.42×	58.54×
W-2	6.99×	11.90×	16.76×	13.46×	5.44×
L-1	12.09×	19.65×	8.69×	18.00×	179.80×
L-2	4.36×	8.89×	3.74×	3.96×	25.68×

- SS压缩效果(GS效果基础上)

- 整体提升约6~7x

Dataset	Overall compaction ratio	File read	File write	Process/Thread	Network	Others	
						Image	File delete/rename
W-1	19.80×	25.74×	15.49×	26.37×	59.71×	159.70×	1.24×
W-2	12.42×	31.42×	20.45×	16.96×	5.60×	53.10×	4.04×
L-1	26.99×	67.87×	32.74×	18.00×	350.08×	1.25×	1.00×
L-2	7.86×	19.57×	10.42×	3.96×	35.56×	1.27×	1.00×



总结

- 基于因果保留的方法
 - 为保留因果定义严格的事件缩减条件，但适应场景有限
 - 例：如果一个进程交替读写同一个文件，那么GS将保留所有这些读写事件(进程和文件语义不断改变)
- 基于执行分区的方法
 - 依赖于应用系统日志识别执行分区
 - 例：如果应用程序在事件循环中无日志或者仅输出一条日志，OmegaLog将无法划分执行分区，合并循环中的冗余重复事件

- [1] Hassan W U , Nouredine M A , Datta P , et al. **OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis**[C]// **Network and Distributed System Security Symposium. 2020.**
- [2] Zhu T , Wang J , Ruan L , et al. **General, Efficient, and Real-Time Data Compaction Strategy for APT Forensic Analysis**[J]. **IEEE Transactions on Information Forensics and Security, 2021, PP(99):1-1.**
- [3] Li Z,Chen Q A,Yang R,et al.**Threat Detection and Investigation with System-level Provenance Graphs: A Survey**[J]. **Computers & Security, 2021(2):102282.**

知人者智，自知者明。
胜人者有力，自胜者强。
知足者富，强行者有志。
不失其所者久，死而不亡者，寿。

谢谢！

