

动态分析

- 仪器测试代码
 - 堆内存：净化
 - Perl污染（信息流）
 - Java竞争条件检查
- 黑匣子测试
 - 模糊测试和渗透测试
 - 黑盒Web应用程序安全性分析

净化

- 目标
 - 检测程序以检测运行时内存错误（越界，初始化前使用）和内存泄漏
- 技术
 - 适用于可重定位目标代码
 - 链接到提供跟踪表的已修改malloc
 - 内存访问错误：在每个加载和存储指令之前插入指令序列
 - 内存泄漏：GC算法

污染：例如Perl

- Perl代码的运行时检查
 - Perl用于CGI脚本，对安全性敏感
 - 污秽检查阻止了一些潜在的不安全呼叫
- 污染的弦
 - 用户输入，来自用户输入的值
 - 与未受污染的字符串匹配的结果除外
- 禁止通话
 - 打印\$ form_data {“ email”}。 “ \ n”;
 - 确定，因为可以安全打印 (???)
 - system (“ mail”。 \$ form_data {“ email”}) ;
 - 带用户输入作为参数的标记系统调用

静态分析

- 摘要程序属性和/或查找问题
- 工具来自程序分析
 - 类型推断，数据流分析，定理证明
- 通常在源代码上，可以在字节码或汇编代码上
- 长处
 - 完整的代码覆盖范围（理论上）
 - 潜在地验证是否存在/报告整个类错误的所有实例
 - 捕获与动态分析不同的错误
- 弱点
 - 假阳性率高
 - 许多属性无法轻松建模
 - 建造困难
 - 几乎没有真正系统中的所有源代码（操作系统，共享库，动态加载等）。

两种类型的静态分析

- (而是) 简单的代码分析。
 - 查找已知的代码问题：例如，不安全的字符串函数 `strncpy ()` , `sprintf ()` , `gets ()`
 - 在您的源代码库中寻找不安全的功能
 - 查找重复出现的问题代码 (问题界面 , 错误代码的复制/粘贴等)
- 更深入的分析
 - 需要复杂的代码解析和计算
 - 有些是在诸如Coverity , Fortify , Visual Studio等工具中实现的。
 - 否则必须在解析器 (如LLVM) 之上开发
 - 对于反汇编程序 , 安全专家是静态分析器的最后一部分.....

静态分析：健全性，完整性

属性	定义
健全性	“报告正确性的声音”分析说没有错误 ® 没有错误 或同等 有一个错误 ® 分析发现错误
完整性 “报告正确性完整”	没有错误 ® 分析说没有错误

召回：A ® B 等于 $(\neg B) \text{ ® } (\neg \text{一种})$

完成

不完整

准确

报告所有错误
无误报

犹豫不决

报告所有错误
可能会报告错误警报

可决定的

不健全

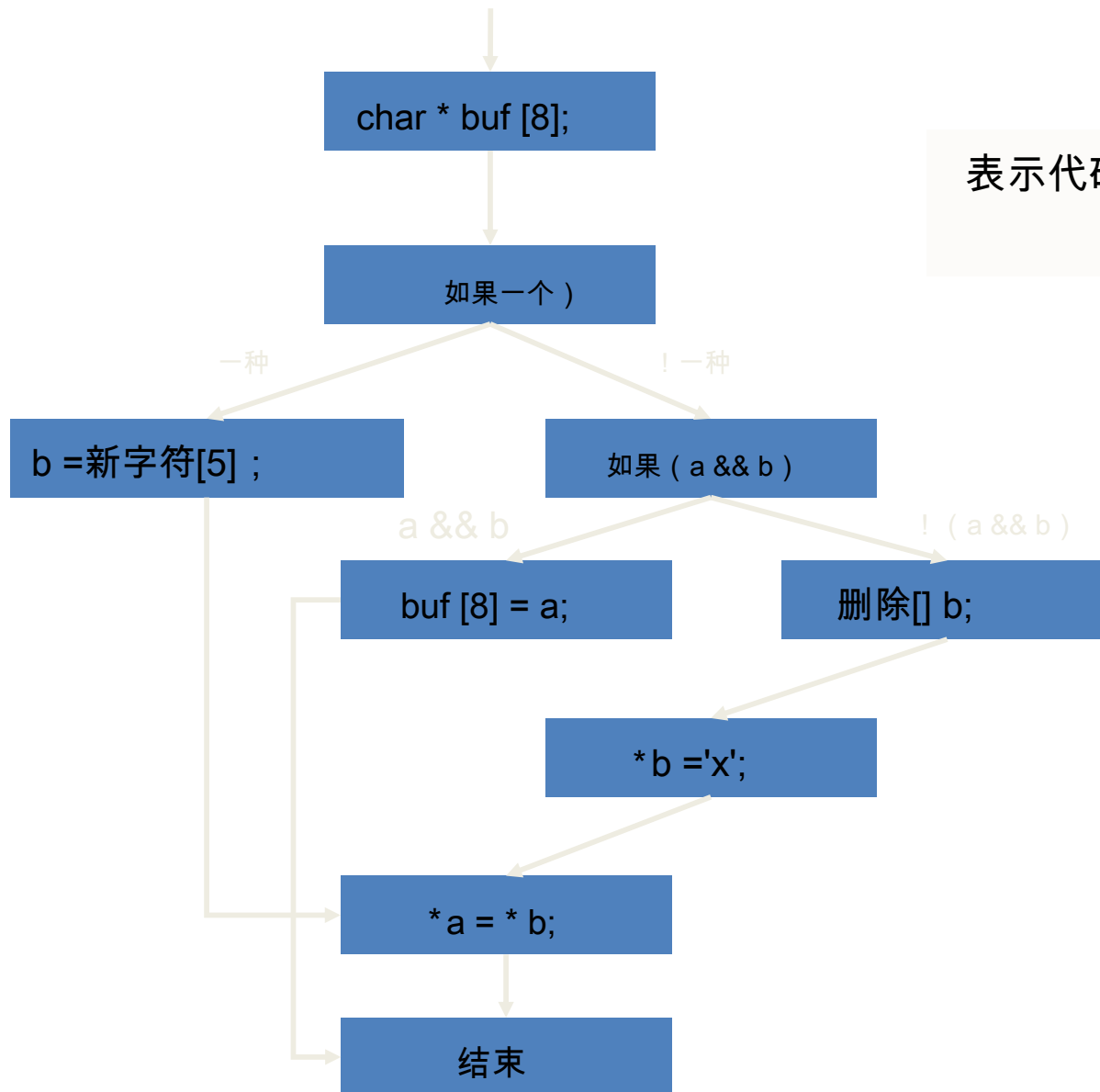
可能不会报告所有错误
报告没有错误警报

可决定的

可能不会报告所有错误
可能会报告错误警报

可决定的

控制流程图

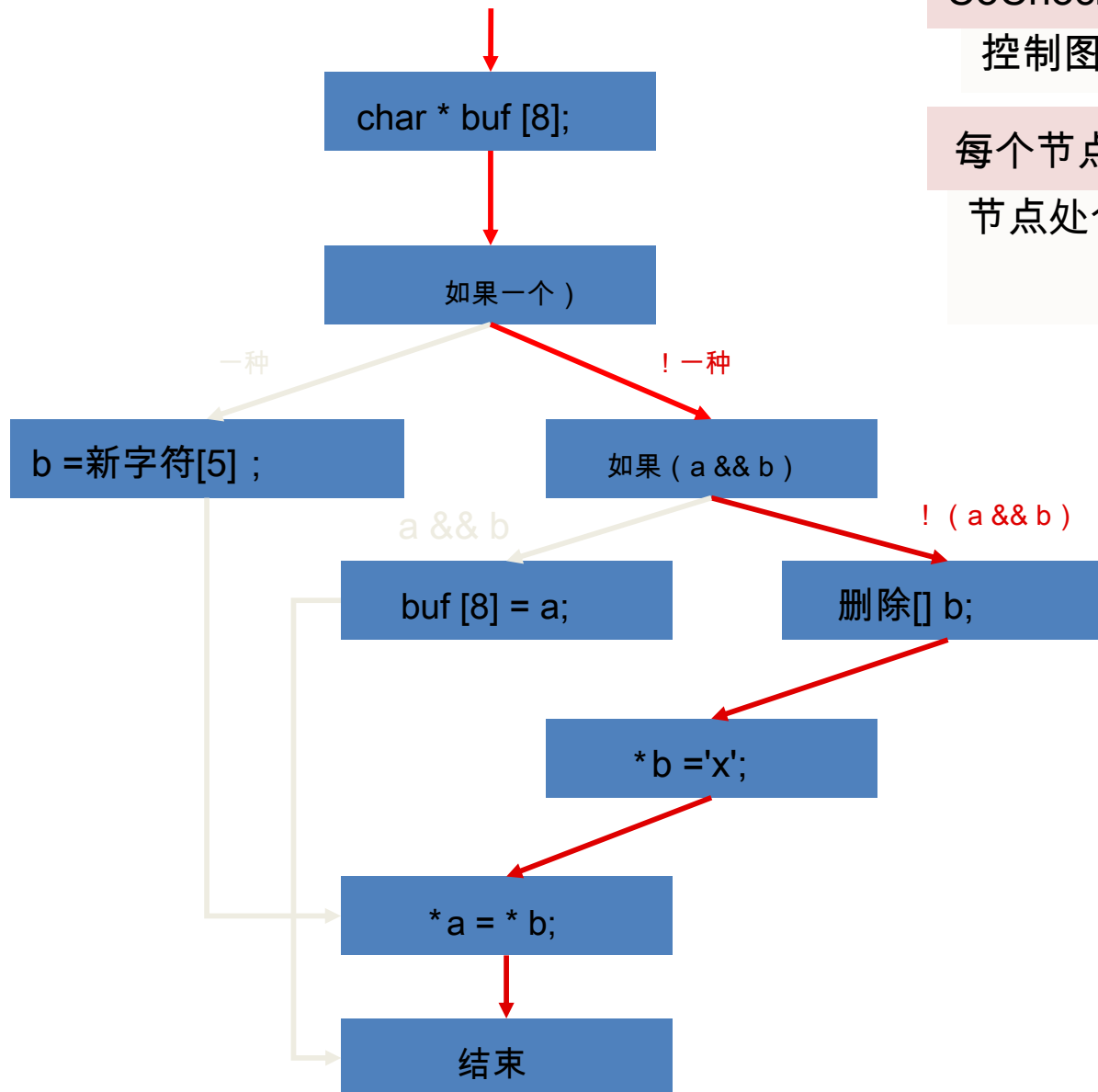


表示代码的逻辑结构
以图形形式

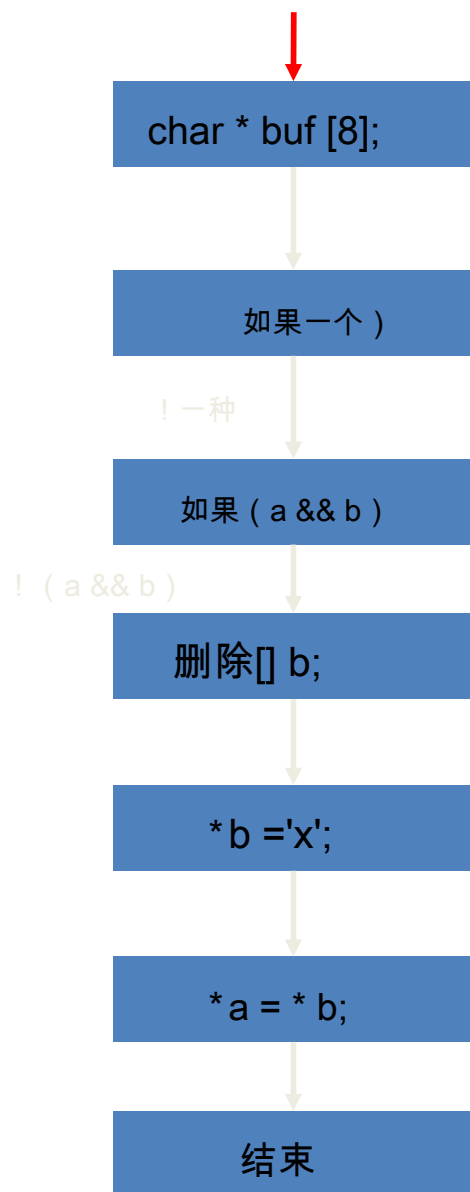
路径遍历

CoCnocnecpetputaulalyly : 分别分析通过
控制图的每条路径

每个节点一次执行一些检查计算 ; 在合并
节点处合并路径



申请检查



查看如何为此路径运行三个检查器

空指针

免费使用

数组超限

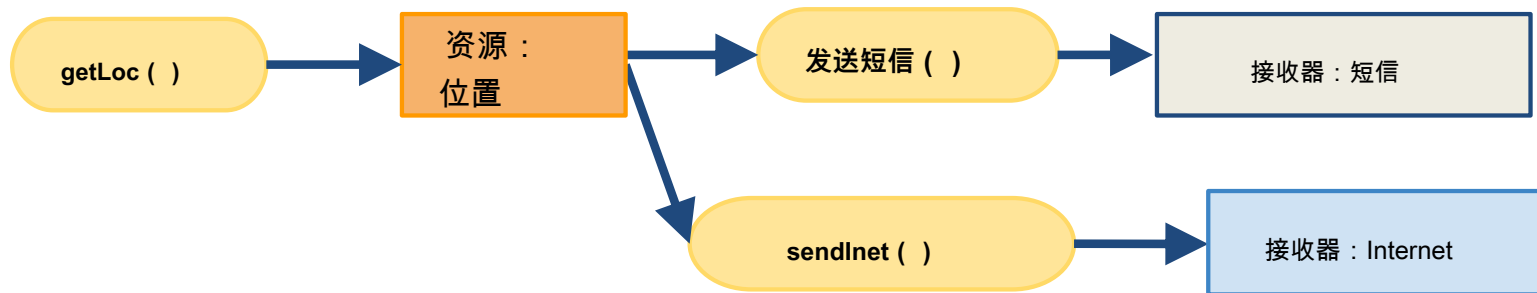
检查器

- 由状态图定义，具有状态转换和错误状态

Run Checker

- 为每个程序变量分配初始状态
- 程序点的状态取决于先前点的状态，程序动作
- 如果达到错误状态则发出错误

数据流分析



- 小号 ∅ 流到汇

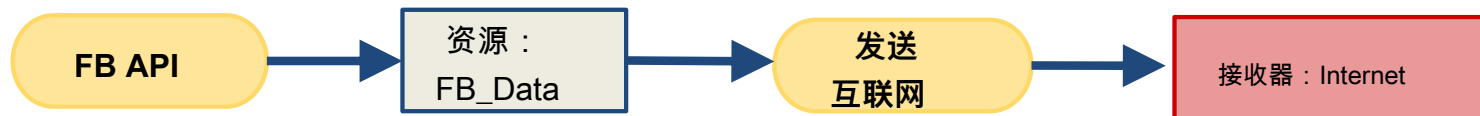
∅ 来源位置Internet, 联系人磁盘等ID等。



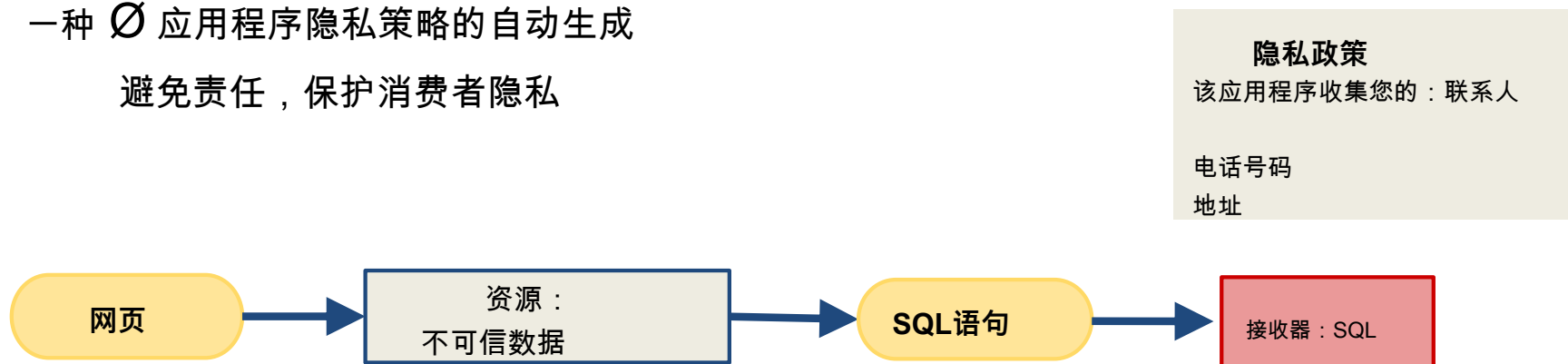
数据流分析的应用

- 漏洞发现
- 中号 Ø 恶意软件/灰色软件分析
数据流摘要可启用特定于企业的策略

- API滥用和数据盗窃检测



- 一种 Ø 应用程序隐私策略的自动生成
避免责任，保护消费者隐私



程序依赖图 (PDG)

q 控制依赖

q 显式+隐式数据依赖

q 特性：

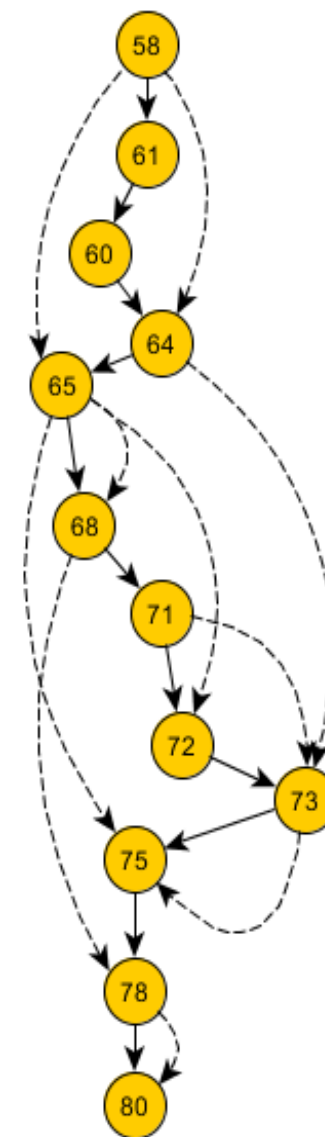
q 路径敏感

q 上下文相关

q 物体敏感

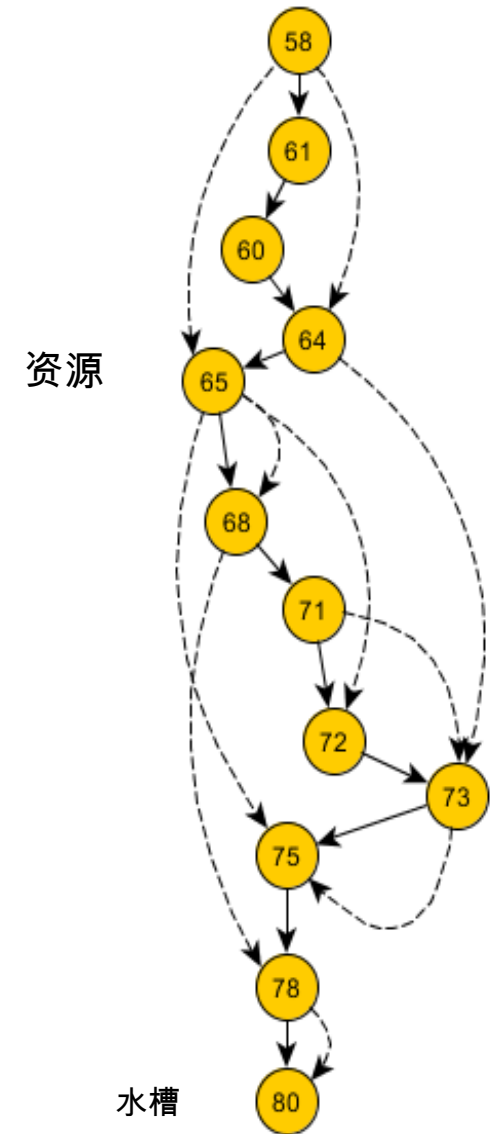
——> 控制依赖

- - - -> 数据依赖



JOANA IFC工具

- 用于信息流分析
- 注释：SINK / SOURCE
- 无干扰：安全级别（高/低）



好处：自动化代码分析

- 例子

- C语言：FindBugs，Fortify，Coverity，MS工具（商业），KLEE（学术）
- Java语言：JOANA，FindBugs，Soot，PMD（学术）

- 目标

- 在所有人都可以使用的工具中捕获专家已知的推荐做法
- 还捕获信息流，无干扰

渗透测试：逆向工程

- 静态分析的一种特殊形式
 - 需要专家用户
 - 用于研究程序的工作原理并查找可以在封闭源代码软件中利用的漏洞
 - 查找后门 (内部攻击.....)
 - 也用于研究恶意软件及其如何利用软件/系统
- 将二进制反转为：
 - 组装形式 (给定的文件格式)
 - 可执行和链接 (ELF) 格式- Linux
 - 便携式可执行 (PE) 格式-Windows
 - 源代码形式 (不常见，例如Java从字节码中反汇编)

渗透测试：逆向工程

- 需要执行拆卸的工具：
 - IDA (Pro) : 世界著名的拆卸工具
 - Ghidra : NSA编写的反汇编程序
 - Objdump (Linux / Mac)
- 可能还需要使用调试器和汇编器来修改汇编代码：
 - Ollydbg (Windows)
 - GDB (Linux / Mac)
 - 美国宇航局

反汇编程序与调试程序

- 调试器旨在运行代码
 - 他们可以反汇编代码 (例如gdb«disas»)
 - 单功能
 - 基于指令指针
 - 一般不要批量拆卸
- 反汇编器不运行代码
 - 输出是反汇编清单
 - 通常相当大的输出
 - 难以导航
 - 比源代码更难理解！
 - 先进的工具还提供带有直观导航的控制流图视图
 - 以及许多其他工具/功能 (重新命名，重新格式化，引入注释，十六进制转储，代码结构分析，库分析)

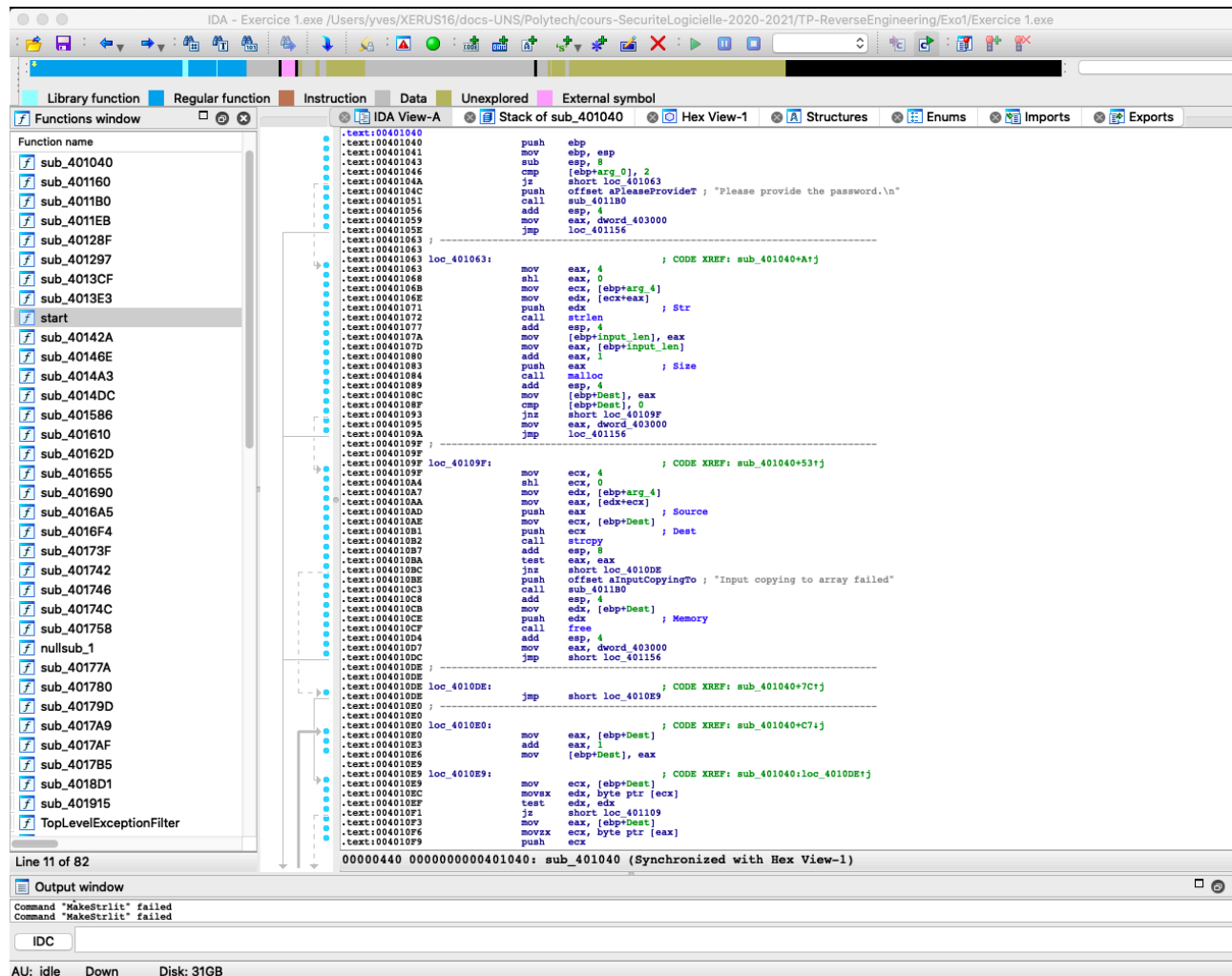
IDA运作

- 加载您感兴趣的二进制文件（例如，拖放）
- IDA分析并表征二进制文件的每个字节
 - 建立数据库（请参阅目录下的文件）
 - 进一步的操作将涉及数据库交互（用于导航的读取，用于重命名的更新等）
- 对代码进行详细分析：
 - 识别函数边界和库调用（甚至包括已知库调用的名称）
 - 识别已知库调用的数据类型识别字符串常量
 -
- 您可以浏览代码和图形（双击）
 - Web浏览器，如历史记录（和ESC =返回）
- 您可以在识别数据和功能（更改名称）时修改内容
 - 当心：许多热键，并且没有撤消！

IDA：拆卸清单

- 主视窗
 - 最初位于入口点
 - 入口点=通常不是主要的，而是开始或_start
- 可以使用空格键切换到图形视图
- 在汇编代码转储左侧的空白处还包含跳转（有条件或无条件）
 - 有助于识别分支和循环结构
 - 有条件的跳跃-破折号
 - 无条件跳-稳固
 - 向后跳-较重的线

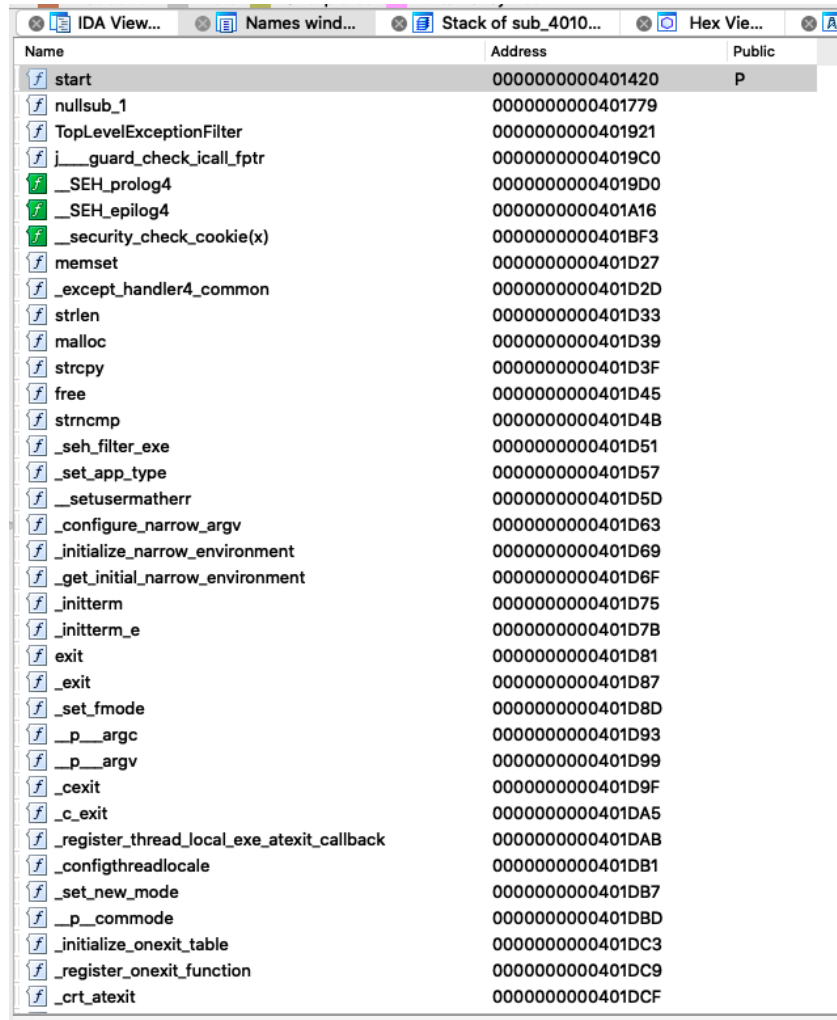
IDA : 拆卸清单



IDA：“名称”窗口

- 基于进口，出口和一些分析
 - F是一个函数
 - L是一个库函数
 - C是代码/指令
 - A是一个字符串
 - D是定义数据
 - 我是导入功能（动态链接）

IDA：“名称”窗口



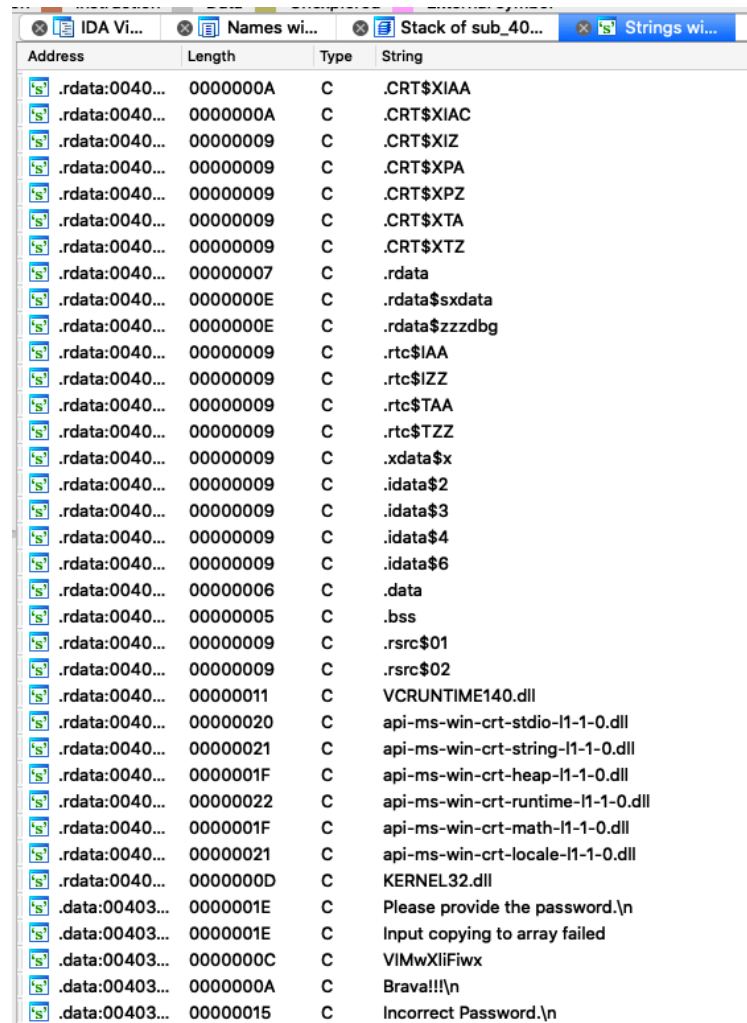
The screenshot shows the IDA Name Window, which displays a list of symbols (functions, variables, etc.) and their corresponding memory addresses. The window has a title bar with tabs for 'IDA View...', 'Names wind...', 'Stack of sub_4010...', 'Hex Vie...', and 'A'. The main content area is a table with three columns: 'Name', 'Address', and 'Public'. The 'start' symbol is highlighted in grey. The list includes various system and application-specific symbols, such as 'memset', 'strlen', 'malloc', 'strcpy', 'free', 'strncpy', 'strncmp', '_seh_filter_exe', '_set_app_type', '_setusermatherr', '_configure_narrow_argv', '_initialize_narrow_environment', '_get_initial_narrow_environment', '_initterm', '_initterm_e', 'exit', '_exit', '_set_fmode', '_p_argv', '_p_argv', '_cexit', '_c_exit', '_register_thread_local_exe_atexit_callback', '_configthreadlocale', '_set_new_mode', '_p_commode', '_initialize_onexit_table', '_register_onexit_function', and '_crt_atexit'.

Name	Address	Public
start	000000000401420	P
nullsub_1	000000000401779	
TopLevelExceptionFilter	000000000401921	
j__guard_check_icall_fptr	0000000004019C0	
__SEH_prolog4	0000000004019D0	
__SEH_epilog4	000000000401A16	
__security_check_cookie(x)	000000000401BF3	
memset	000000000401D27	
_except_handler4_common	000000000401D2D	
strlen	000000000401D33	
malloc	000000000401D39	
strcpy	000000000401D3F	
free	000000000401D45	
strncpy	000000000401D4B	
_seh_filter_exe	000000000401D51	
_set_app_type	000000000401D57	
_setusermatherr	000000000401D5D	
_configure_narrow_argv	000000000401D63	
_initialize_narrow_environment	000000000401D69	
_get_initial_narrow_environment	000000000401D6F	
_initterm	000000000401D75	
_initterm_e	000000000401D7B	
exit	000000000401D81	
_exit	000000000401D87	
_set_fmode	000000000401D8D	
_p_argv	000000000401D93	
_p_argv	000000000401D99	
_cexit	000000000401D9F	
_c_exit	000000000401DA5	
_register_thread_local_exe_atexit_callback	000000000401DAB	
_configthreadlocale	000000000401DB1	
_set_new_mode	000000000401DB7	
_p_commode	000000000401DBD	
_initialize_onexit_table	000000000401DC3	
_register_onexit_function	000000000401DC9	
_crt_atexit	000000000401DCF	

IDA：字符串窗口

- 完整列出程序中嵌入的字符串
- 可配置的
 - 在“字符串”窗口中右键单击并选择设置
 - 可以更改要搜索的最小长度或字符串样式（如果更改设置，IDA将重新扫描字符串）
- 定位有趣的输入/输出或文本数据的出色工具
 - 例如，在代码中检测成功条件

IDA：字符串窗口



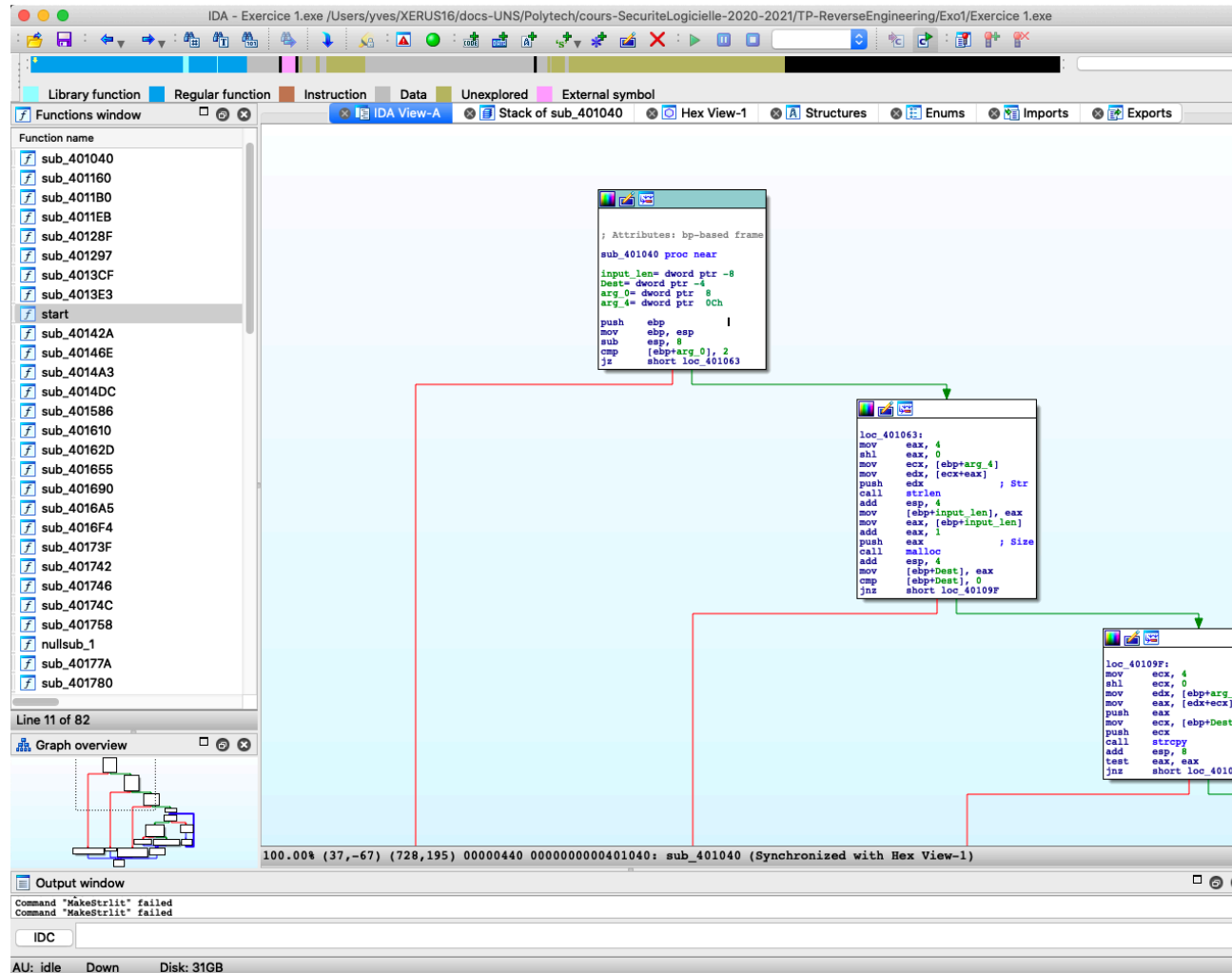
The screenshot shows the IDA Pro interface with the 'Strings window' active. The window displays a list of strings found in the program, organized by address, length, type, and the string itself. The strings are categorized into various sections like .rdata, .data, .bss, and .rsrc, and include system DLLs and user-defined strings.

Address	Length	Type	String
.rdata:0040...	0000000A	C	.CRT\$XIAA
.rdata:0040...	0000000A	C	.CRT\$XIAC
.rdata:0040...	00000009	C	.CRT\$XIZ
.rdata:0040...	00000009	C	.CRT\$XPA
.rdata:0040...	00000009	C	.CRT\$XPZ
.rdata:0040...	00000009	C	.CRT\$XTA
.rdata:0040...	00000009	C	.CRT\$XTZ
.rdata:0040...	00000007	C	.rdata
.rdata:0040...	0000000E	C	.rdata\$sxdata
.rdata:0040...	0000000E	C	.rdata\$zzzdbg
.rdata:0040...	00000009	C	.rtc\$IAA
.rdata:0040...	00000009	C	.rtc\$IZZ
.rdata:0040...	00000009	C	.rtc\$TAA
.rdata:0040...	00000009	C	.rtc\$TZZ
.rdata:0040...	00000009	C	.xdata\$x
.rdata:0040...	00000009	C	.idata\$2
.rdata:0040...	00000009	C	.idata\$3
.rdata:0040...	00000009	C	.idata\$4
.rdata:0040...	00000009	C	.idata\$6
.rdata:0040...	00000006	C	.data
.rdata:0040...	00000005	C	.bss
.rdata:0040...	00000009	C	.rsrc\$01
.rdata:0040...	00000009	C	.rsrc\$02
.rdata:0040...	00000011	C	VCRUNTIME140.dll
.rdata:0040...	00000020	C	api-ms-win-crt-stdio-l1-1-0.dll
.rdata:0040...	00000021	C	api-ms-win-crt-string-l1-1-0.dll
.rdata:0040...	0000001F	C	api-ms-win-crt-heap-l1-1-0.dll
.rdata:0040...	00000022	C	api-ms-win-crt-runtime-l1-1-0.dll
.rdata:0040...	0000001F	C	api-ms-win-crt-math-l1-1-0.dll
.rdata:0040...	00000021	C	api-ms-win-crt-locale-l1-1-0.dll
.rdata:0040...	0000000D	C	KERNEL32.dll
.data:00403...	0000001E	C	Please provide the password.\n
.data:00403...	0000001E	C	Input copying to array failed
.data:00403...	0000000C	C	VIMwXliFiwx
.data:00403...	0000000A	C	Brava!!!\n
.data:00403...	00000015	C	Incorrect Password.\n

IDA：图表

- 可以生成许多图
- 功能流程图
 - 无条件跳-蓝线
 - 如果为真，则有条件跳转-绿线
 - 如果为假，则有条件跳转-红线
 - 将鼠标移到图形上方以获取更多信息
- 程序的函数调用树（森林）
- 一个函数的所有交叉引用（《我该叫谁？》）
- 所有对函数的交叉引用（《谁叫我？》）

IDA : 功能流程图



x86汇编基础知识：说明

- 内存操作：
 - Mov <dst> , <src>-地址可以通过«[地址值]»描述
 - 推送/弹出<注册表>
 - Xcgh <注册表1> , <注册表2>
- 算术运算符：
 - Add / Dec / Mul / Div <注册表> , <操作数2>
 - Inc / Dec <注册表>
 - Neg <注册表>-补码
- 位级操作：
 - 和/或/异或<registry1> , <registry2>
 - 不是<registry>
 - Shl <注册表> , <添加位>
 - Shr <注册表>
 - 角色/角色<registry>

x86汇编基础知识：说明

- 测验
 - Cmp <注册表1> , <注册表2>
- 跳
 - JMP <代码位置>
 - Je <代码位置> (如果先前的测试相等)
 - Jne <代码位置> (如果先前的测试不相等)
 - Jz <代码位置> (如果先前的操作为零)
 - Jnz <代码位置> (如果先前的操作不为零)
- 子程序调用
 - 呼叫<代码位置>
 - t

安全需求工程：两个方法

ñ 认证安全

- ñ 最佳做法，安全准则
- ñ 信息流控制
- ñ 静态和动态分析

ñ 设计安全

- ñ 安全目标
- ñ 威胁分析

现有方法

- UML提案

- SecureUML , 模型驱动的体系结构[Basin等]
- UMLsec [Juriens]
- 虐待案件[McDermott & Fox]
- 滥用案例[Sindre & Opdhal]

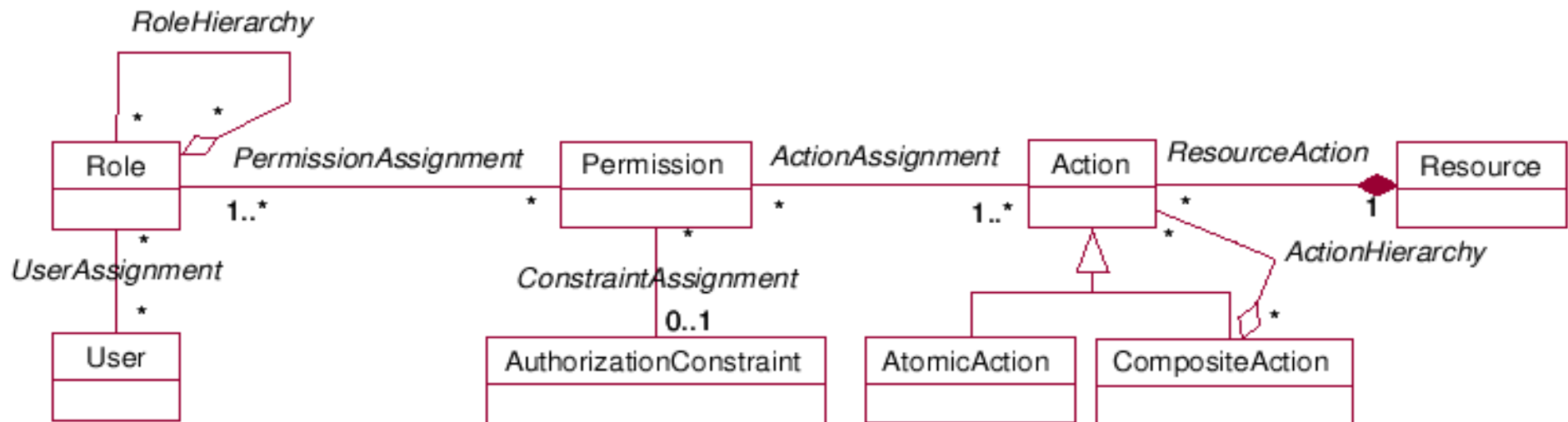
- 早期需求建议

- 反需求[van Lamsweerde等 , Crook等] ,
- 问题框架 , 滥用框架[Hall等 , Lin等]
- 安全模式[Giorgini和Mouratidis]
- 隐私建模[Liu等 , Anton等 ,]

SecureUML

- D.盆地 , J. Doser和T. Lodderstedt , 2003年
- 它们为指定访问控制策略提供支持
- RBAC的概念表示为元模型类型
 - 用户 , 角色 , 权限 , 操作是类型
 - UserAssignment , PermissionAssignment , RoleHierarchy是关系
 - AuthorizationConstraint是关联ConstraintAssigment附加到权限的谓词
 - 一阶逻辑表示的授权约束
 - 用于建立权限的有效性

SecureUML元模型



D.盆地, J. Doser和T. Lodderstedt. 面向过程的系统的模型驱动的安全性。在过程中。SACMAT '03的第100-109页。ACM出版社, 2003年。

SecureUML语义

- 访问控制配置是用户和角色权限的分配
- SecureUML根据访问控制配置和特定系统状态下授权约束的有效性来做出访问控制决策
 - 验证是否允许用户在特定时间针对访问控制配置在系统状态下执行操作

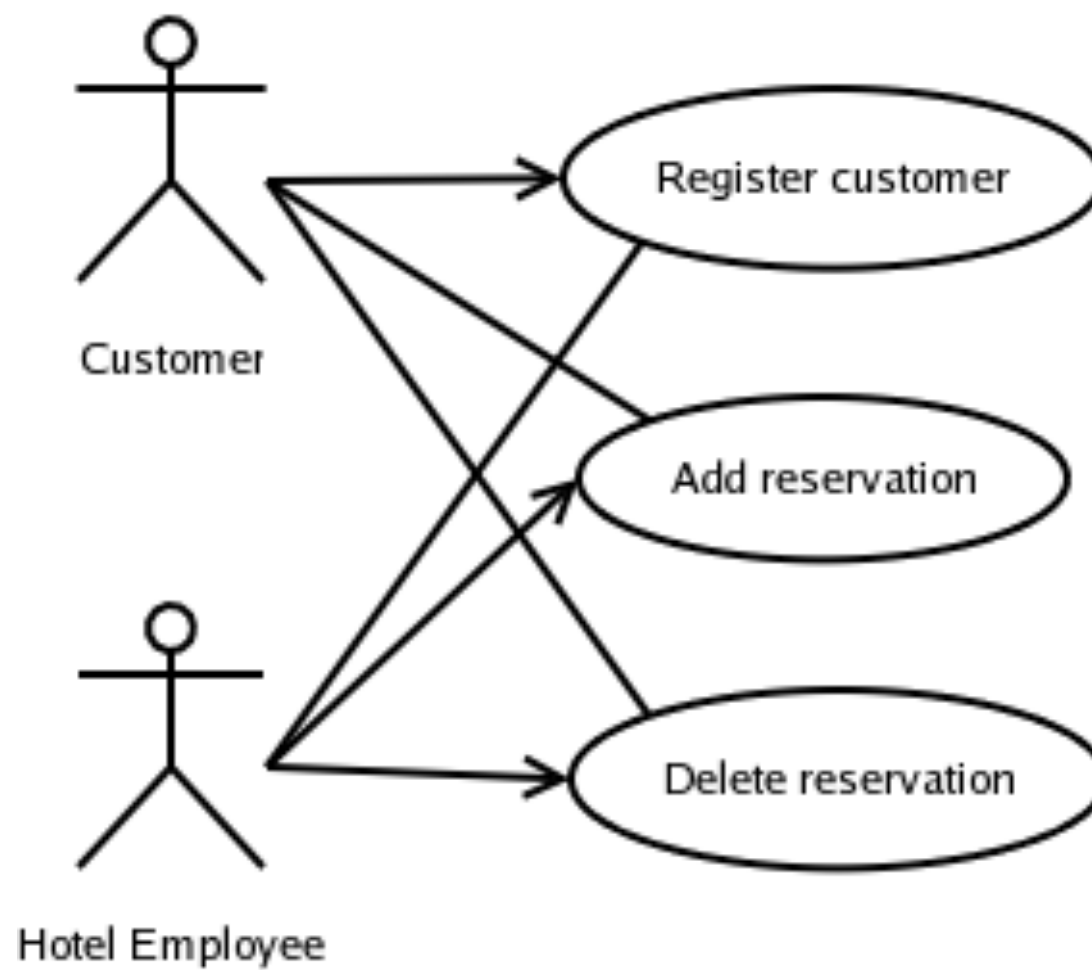
SecureUML的局限性

- 没有分析软件系统将在其中运行的组织环境中的安全要求
- 需要先知道冲突的角色
 - 没有从系统的需求模型中检测到冲突

用例图

- 建立系统的第一个草图模型
- 表征系统的使用方式
- 提供用于描述系统功能的符号
 - 参与者：与系统交互的外部代理的抽象
 - 用例：规范系统与代理之间的交互类型
 - 关联线：将代理与其参与的用例联系起来

预约系统



高斯

- 研究项目
 - 用于将目标形式化为需求
 - 得出系统行为的描述
 - 通过获取和形式化功能需求和非功能需求来分析系统结构
-
- GRAIL工具支持

KAOS概念

- 代理商
 - 系统的活动组件
 - 发挥作用
- 目标
 - 有关系统意图的说明性陈述
 - 功能目标：要提供的服务
 - 非功能性目标：服务质量
- 域属性
 - 关于环境的描述性陈述（例如，物理定律，规范）

目标

- 按照AND / OR层次结构进行组织
 - 目标细化用于构建细化抽象层次
 - 高水平的目标是战略性的
 - 粗粒状，带有许多代理
 - 低级目标是技术性的
 - 细粒，涉及较少的代理商
- 要求：一位代理商的最终目标

障碍物

- 识别违反目标的情况
- 某些目标的障碍是无法满足目标的条件
- 据说障碍物O会妨碍域Dom iff中的目标G

$\{O, \text{Dom}\} \models \neg G$

梗阻

$\text{Dom} \models \neg O$

域一致性

障碍物分析

- 障碍分析在于对目标持悲观态度
- 确定解决目标的尽可能多的方法，以解决每种情况
- 提供了用于障碍物生成和逻辑 (AND / OR) 精化的形式技术
- 一旦产生障碍，就需要解决
 - 解决技术：目标替代，代理替代，目标削弱，目标恢复，障碍预防和障碍缓解
- 障碍分析是一个递归过程
 - 它可能会产生新的目标，为此可能会产生并解决新的障碍

安全目标

- 被认为是元类
- 高层次的抽象
 - 保密
 - 廉洁
 - 可用性
 - 隐私
 - 真实性
 - 不可否认
- 每个安全目标都必须实例化为特定于应用程序的安全目标

保密

目标保密

目标 ~~避免~~ [SensitiveInfoKnownByUnauthorizedAgent]

正式规格 $\forall ag : \text{代理}, ob : \text{对象} \neg \text{授权}(ag, ob.Info) \Rightarrow$
 $\neg \text{知道}(ob.Info)$

果阿 \Rightarrow 升 (~~避免~~ [PaymentMediumKnownBy3rdParty]

正式规格 $\forall p : \text{代理人}, acc : \text{帐户} \neg (\text{拥有者}(p, acc$
 $c) \vee \text{管理}(p, acc))$

如果代理 p 不是帐户的所有者 acc 而且他
不应该管理它，他不知道

数和销帐户的

障碍物分析

- $NE \wedge p$: 代理商, 访问: v 帐户

\forall 门控一个目标

$(NG) \neg (\text{拥有者}(p, acc) \text{ 管理}(p, acc))$

- 如 \forall 求和域理论包含以下属性:

$(D1) \forall (p \text{ 知道 } acc, acc \# \text{ 帐户所有者}(acc, PIN)) \wedge \text{知道}(p.name) \Rightarrow \text{知道}(acc, acc \#)$

$(D2) \forall acc : \text{帐户已知}(acc, \text{帐户编号}) \Rightarrow \text{知道}(\text{根据PIN码})$

- 我们可以形成

$(O) \quad p : \text{代理商}, acc : \text{帐户}$

$\neg (\text{拥有者}(p, acc) \text{ 管理}(p, acc)) \wedge \text{知道}(p.name)$

盟友会带来以下潜在障碍:

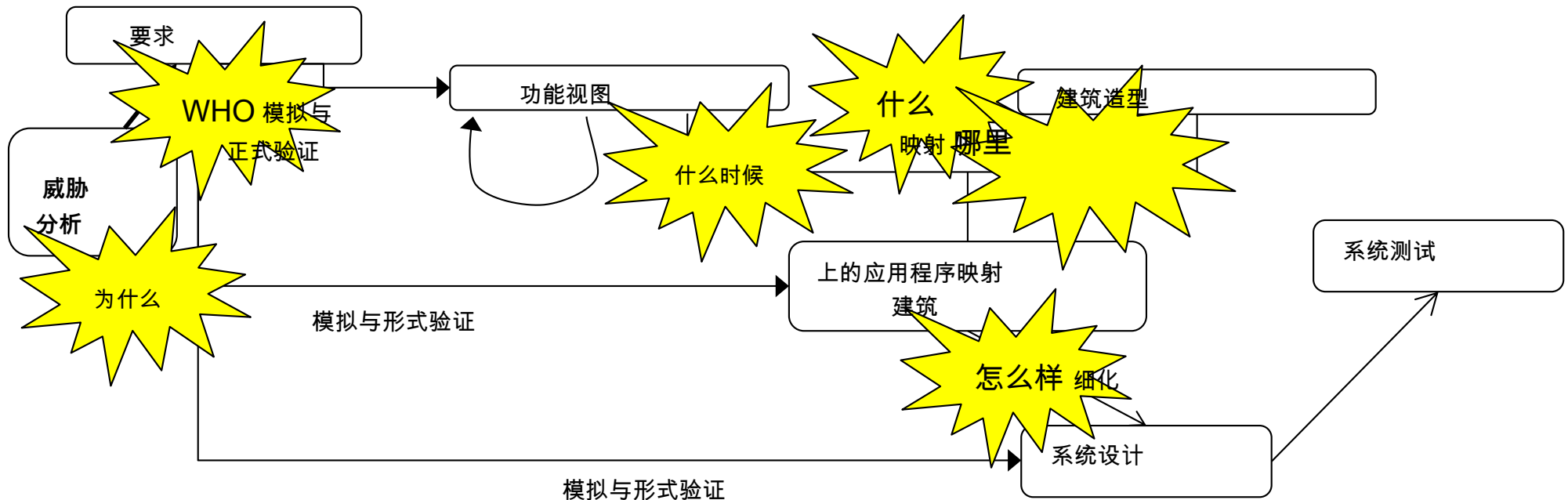
超越纯IT的安全性要求

安全

- SE关键问题：
 - 增量设计？
 - 在整个SDLC中是否连续使用需求？
 - 指定安全要求而不是安全机制？
- 系统关键问题：
 - 捕获系统的观点（硬件/软件分区）？
 - 捕获环境约束（例如实时约束）？
 - 满足功能和安全要求？
- 作为整体方法的模型驱动工程：SysML-Sec
 - 在架构中部署软件组件
 - 架构= CPU +内存+总线+软件
 - 召集系统工程师和安全专家

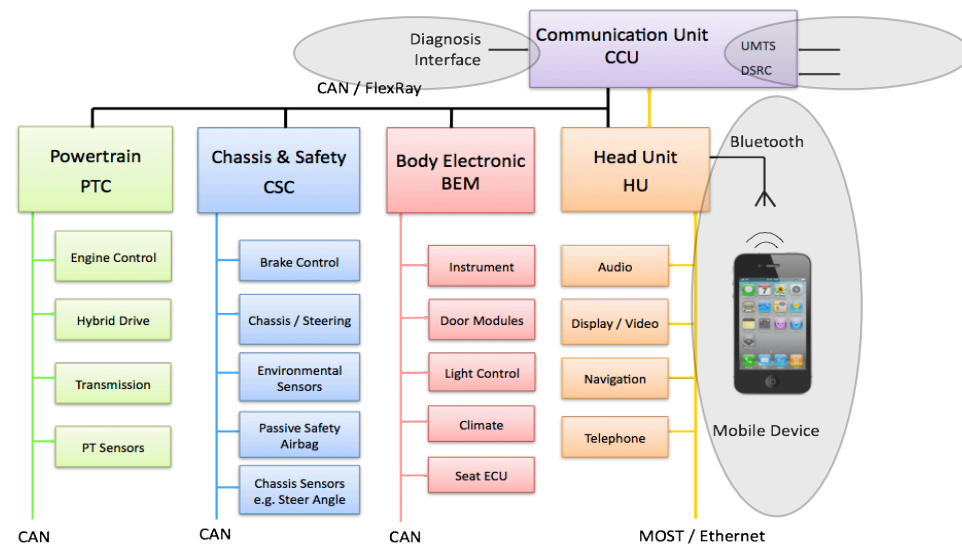
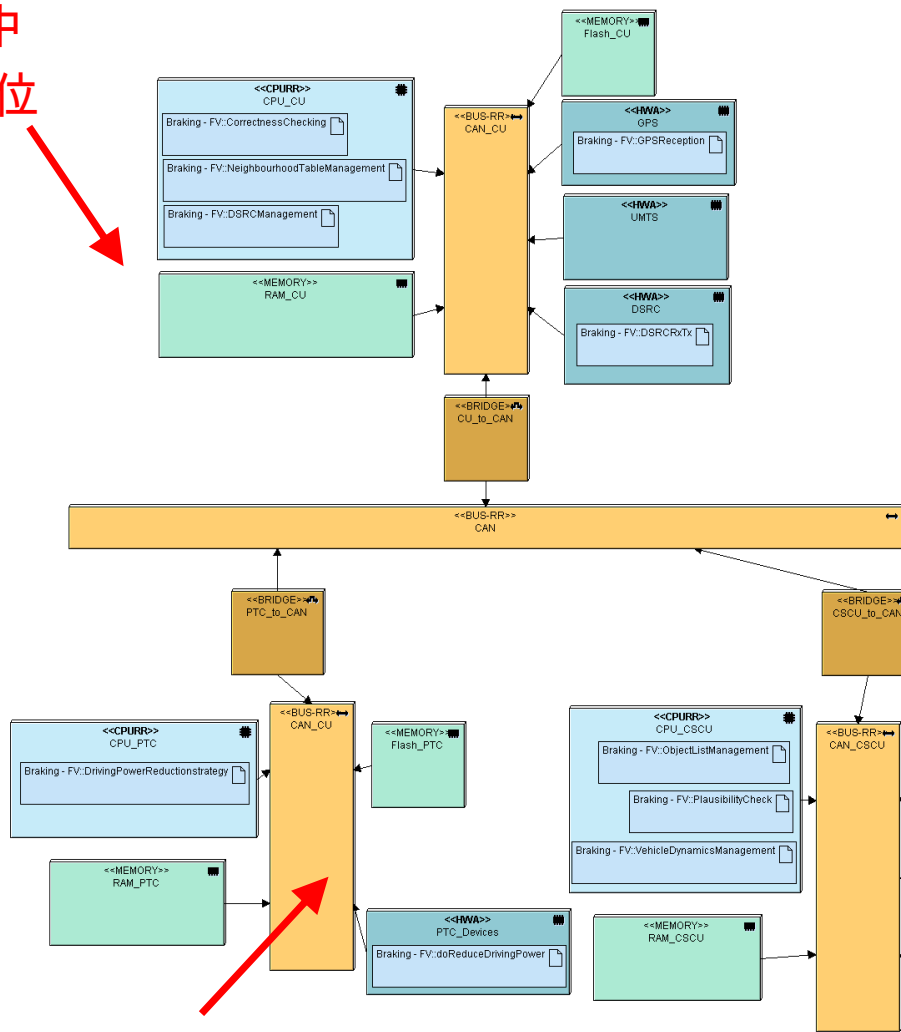
SysML-Sec：重新审视了Y-Chart

- 安全问题：
 - 什么：资产 被保护
 - 什么时候：操作 顺序 涉及这些资产的职能
 - 哪里：体系结构映射 这些资产的功能
 - 为什么：攻击/威胁 设想可以激发安全对策
 - WHO：利益相关者+攻击者和能力 (风险分析)
 - 怎么样：安全目标 由于架构 (例如，网络拓扑，进程隔离等)

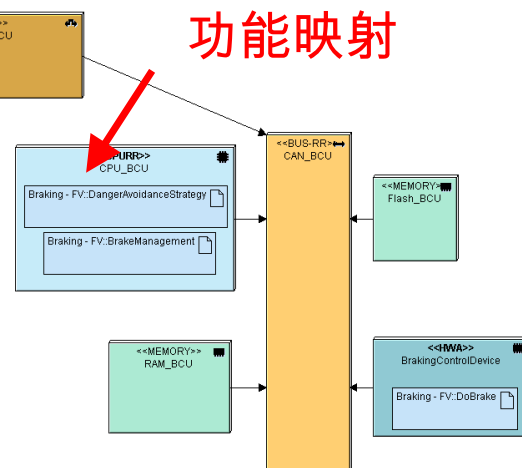


(实际拓扑的部署图)

处理中
单位

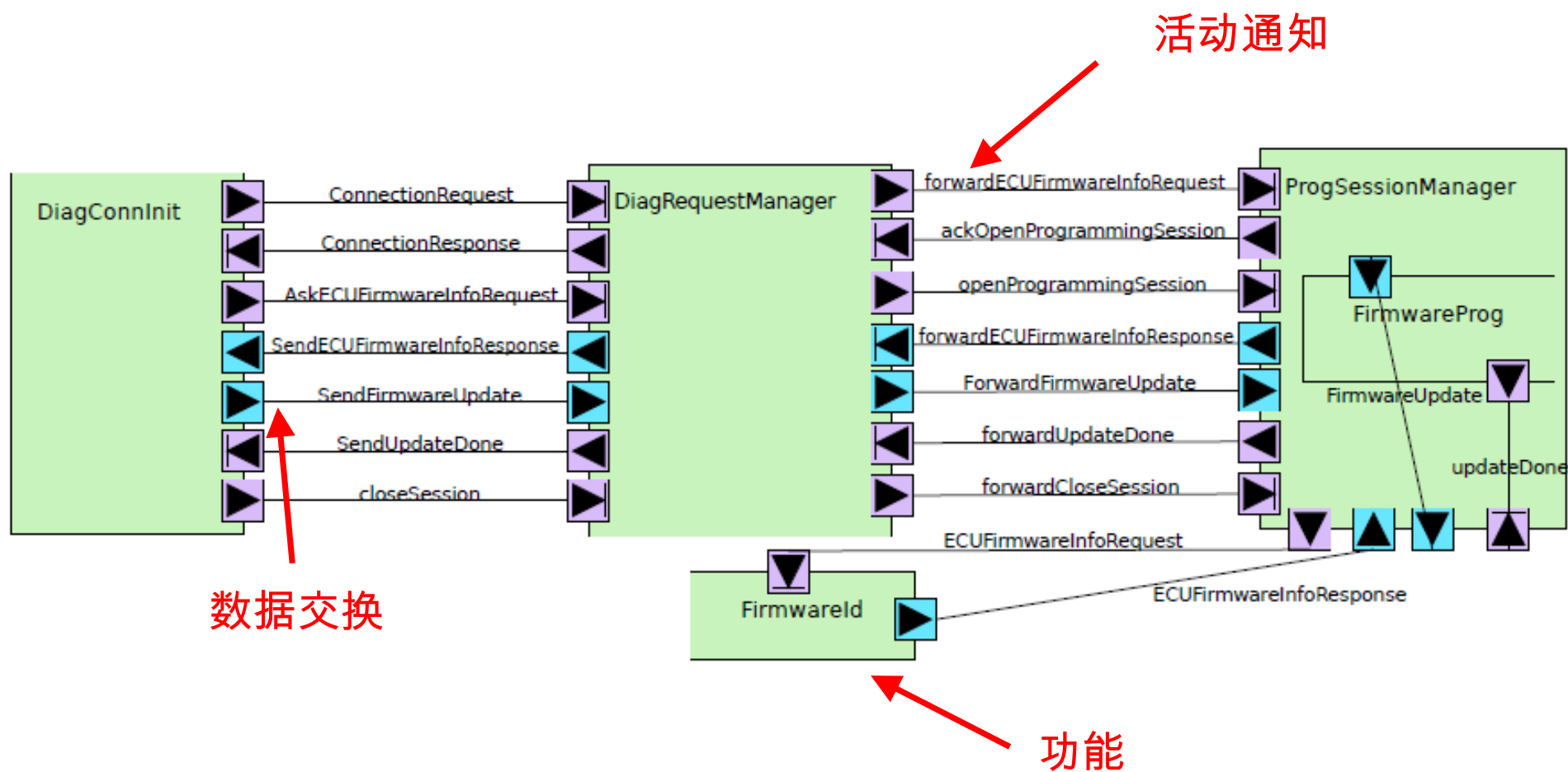


功能映射



事件/数据流映射

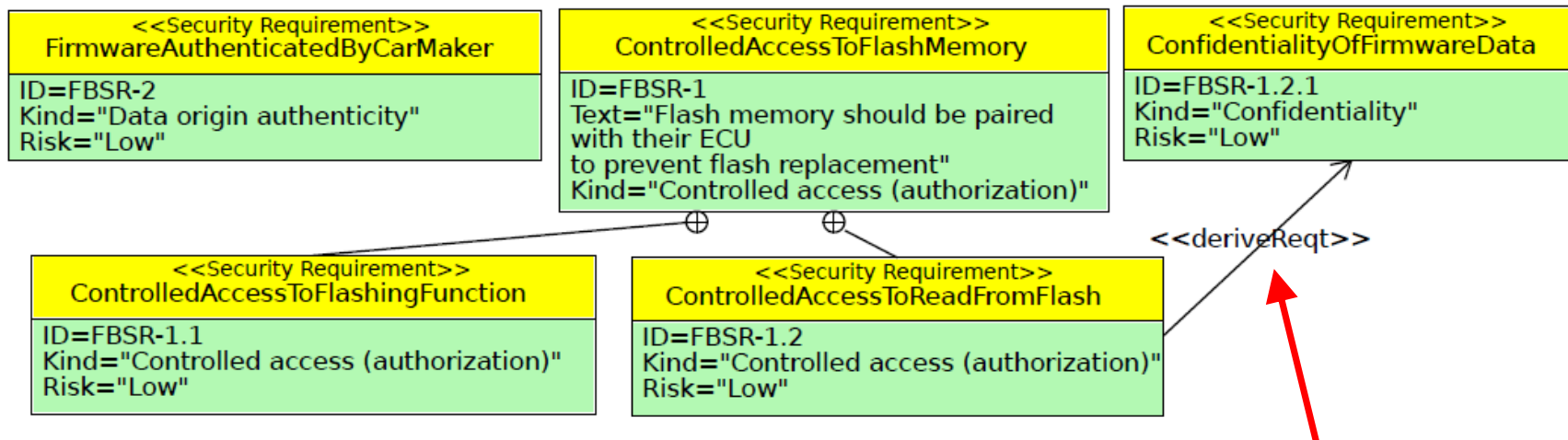
功能视图：指定信息流 内部框图



安全性和对策类型 需求图

安全属性：例如，机密性，真实性，完整性，新鲜度，可用性.....适用于某些资产

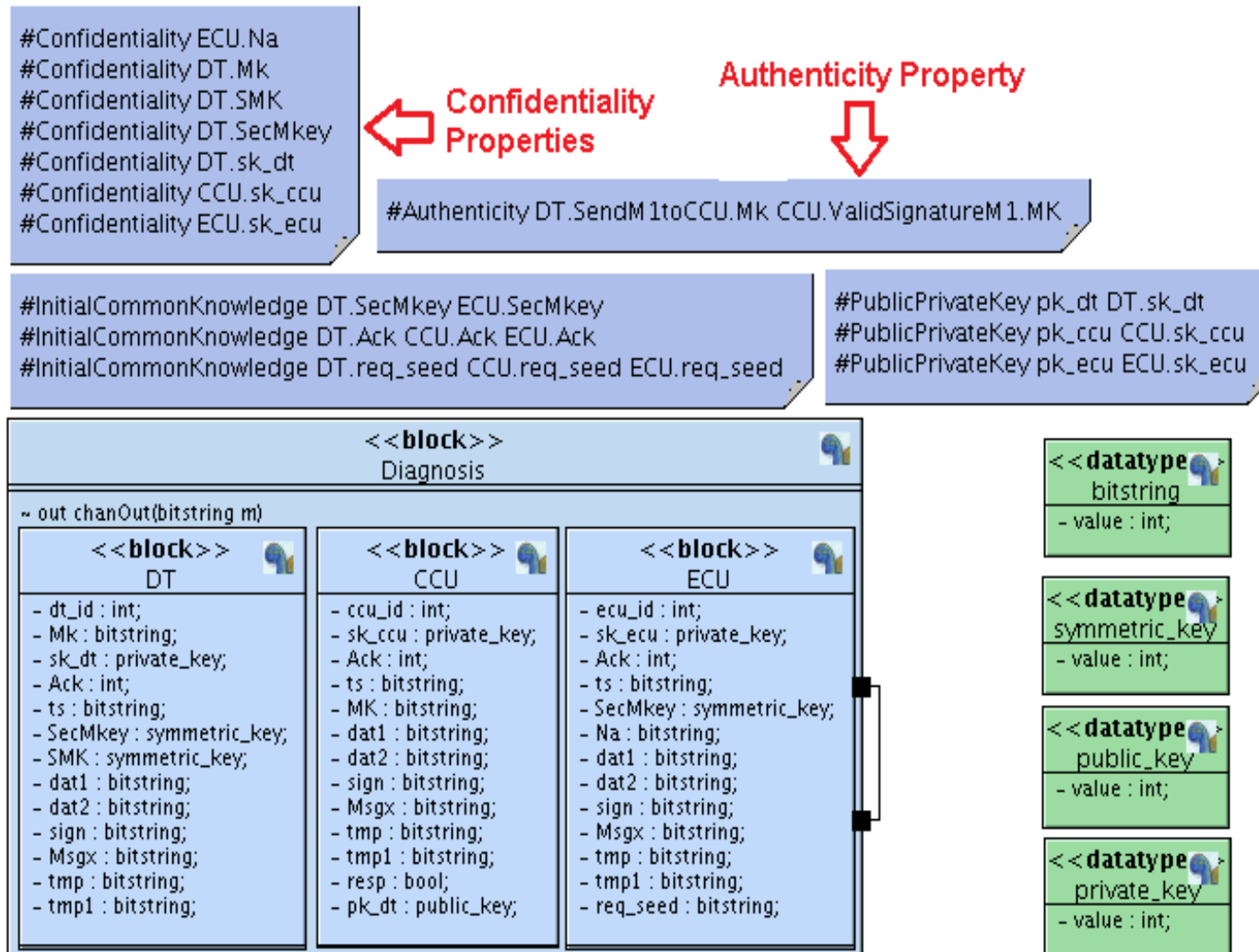
(硬件/软件或更重要的是数据或信息流)



跟踪细化和依赖性

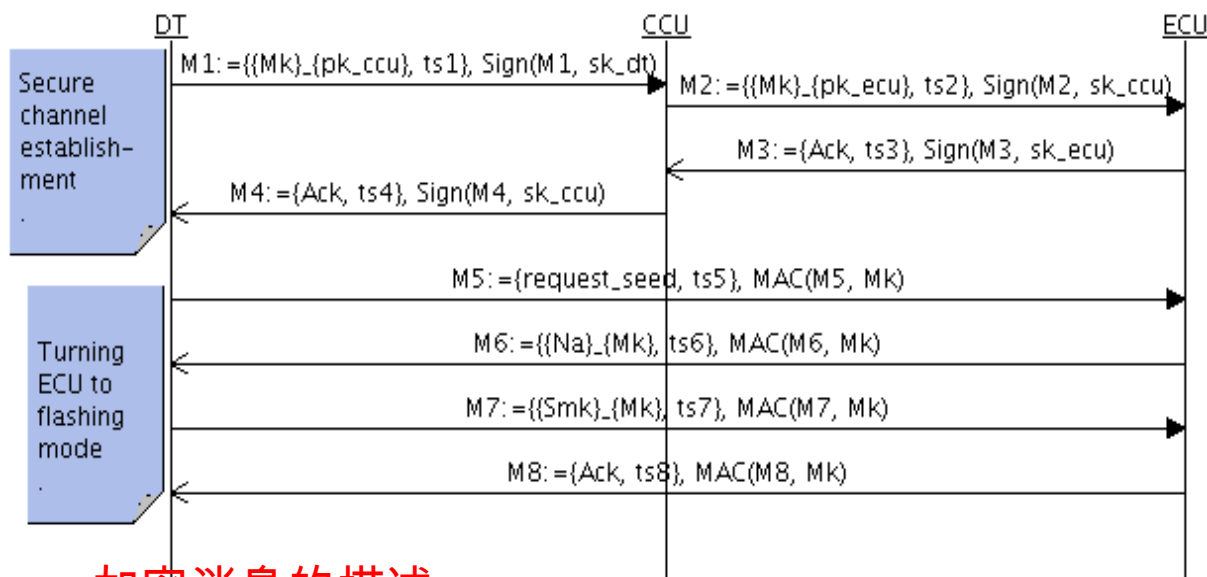
SysML块定义图：加密协议

环境支持

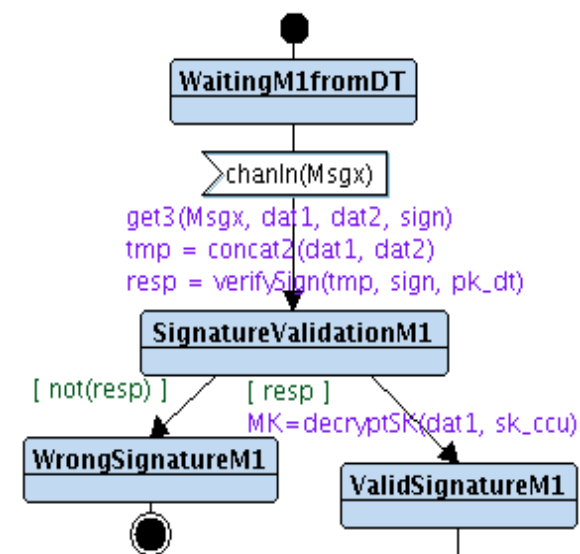


加密协议消息：内容和

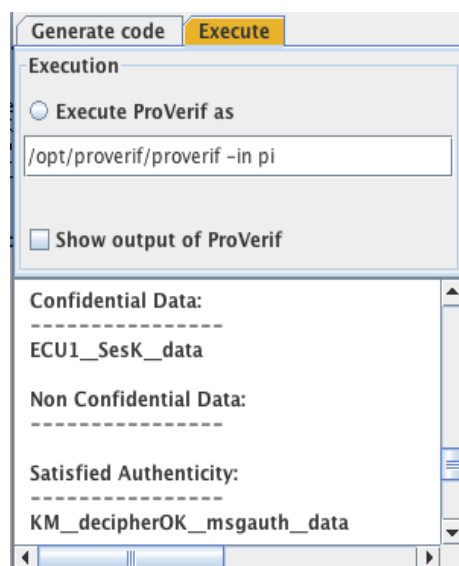
处理方式



加密消息的描述

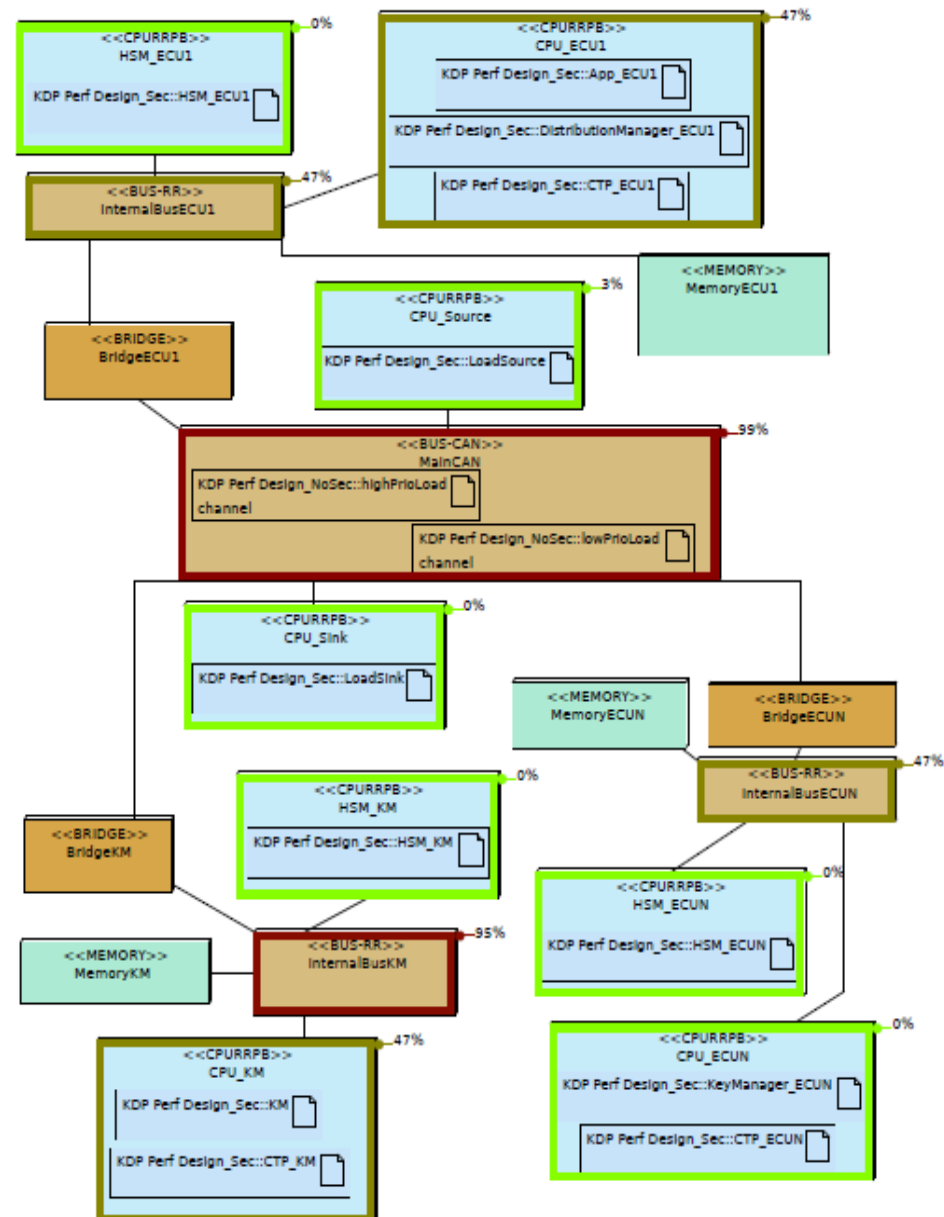


在CCU处理M1



ProVerif中的正式验证 (Dolev-Yao攻击者)

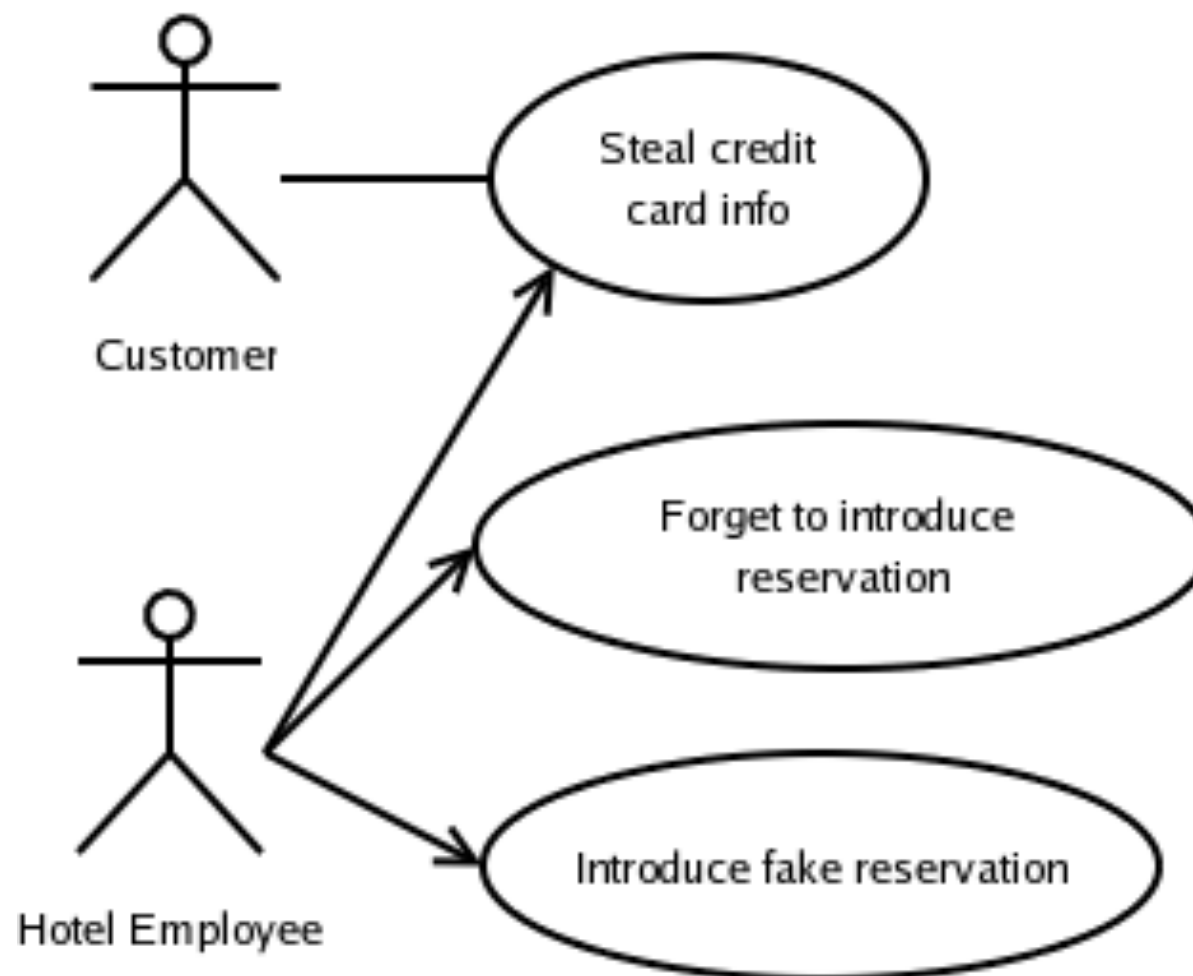
模拟：分析安全性的影响



滥用案例

- 麦克德莫特和福克斯，1999年
- 否定用例，用于对安全需求进行建模
- 指定系统与一个或多个参与者之间的交互，其中交互的结果对系统或系统中的参与者之一有害
- 参与用例的参与者是相同的

滥用案例：预订系统



滥用案件的范围

- 与功能需求分开对安全需求进行建模
 - 滥用案例图仅显示滥用，不与正常使用一起显示滥用
 - 他们不调查使用和滥用之间的关系

滥用案例

- Guttorm Sindre和Andreas Opdahl , 2000年
- 扩展用例以建模安全需求
- 指定系统应避免的行为
- 指定滥用者如何损坏系统

滥用案例：概念

- 误用
 - 敌对演员
 - 与用例中的参与者类似的表示法，只是滥用者的黑色“头”而不是白色
- 滥用案例
 - 损害利益相关者或系统本身的行为过程
 - 系统中不需要的行为
 - 黑色圆圈表示
- 用例
 - 系统功能
 - 防止滥用的对策
- 关系
 - “包含”和“扩展”
 - “预防”：用例可防止滥用案例的激活
 - “检测”：用例检测到滥用案例的激活

使用/滥用案例图

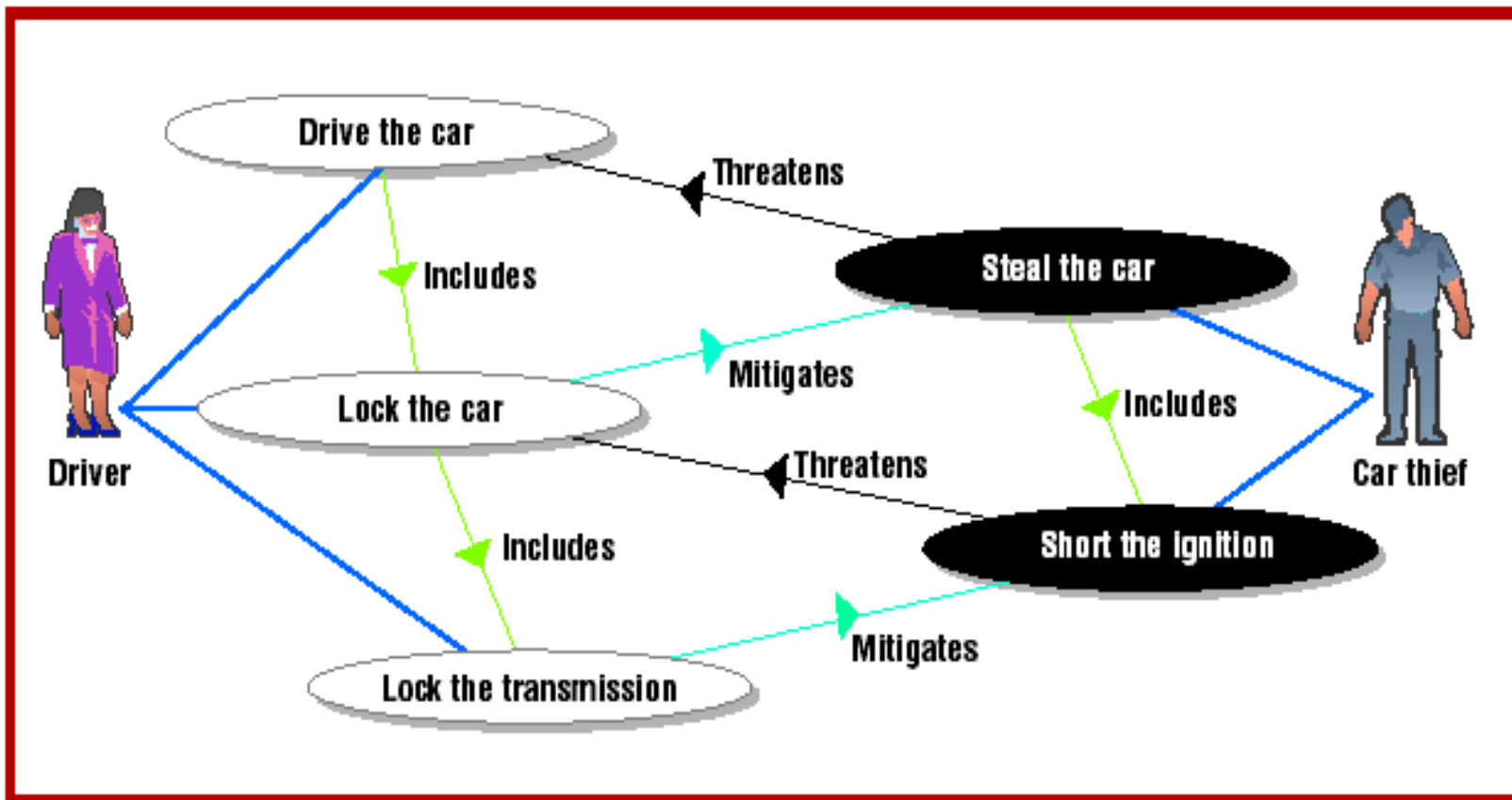
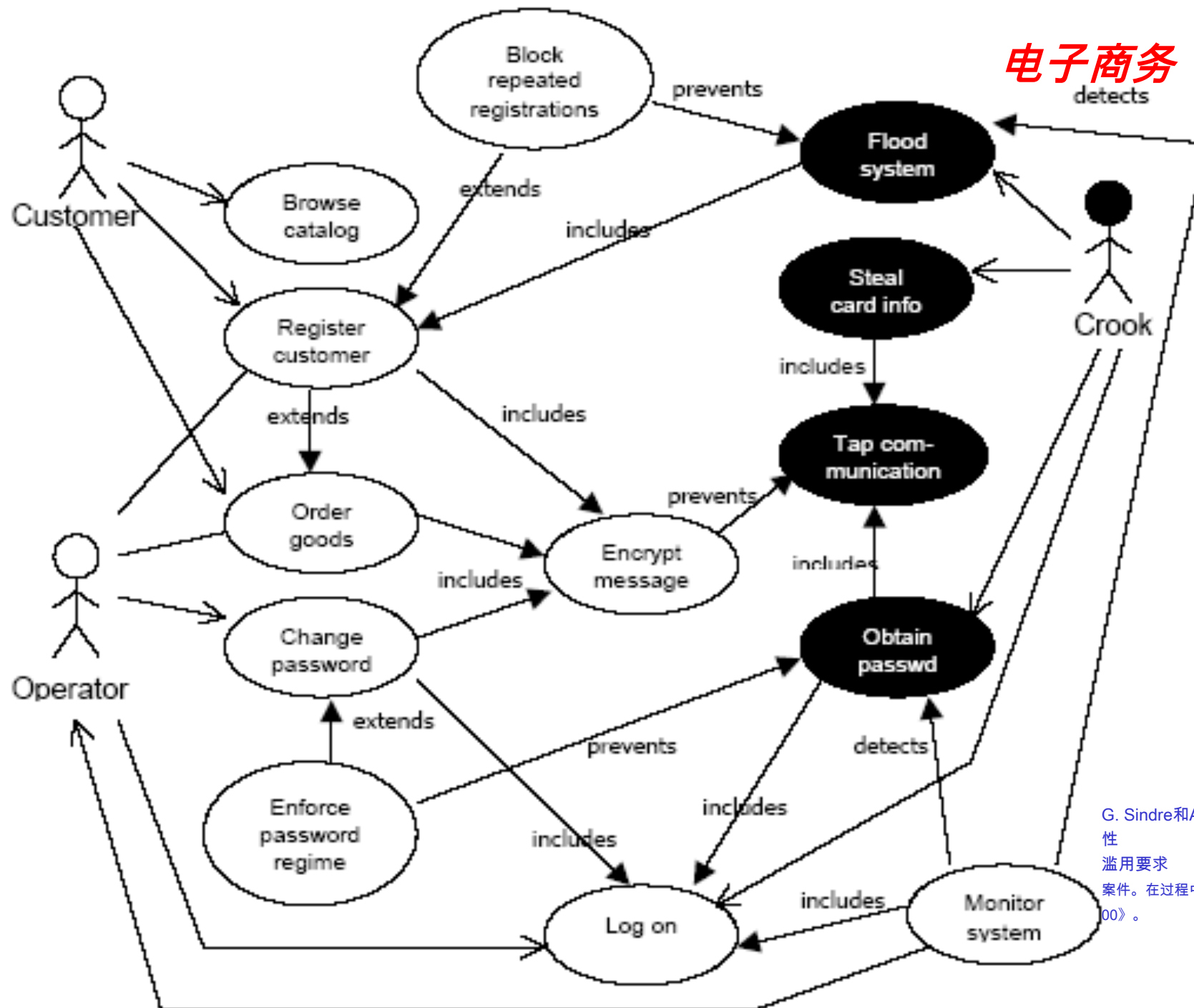


Figure 1. Use/misuse-case diagram of car security requirements. Use-case elements appear on the left; the misuse cases are on the right.

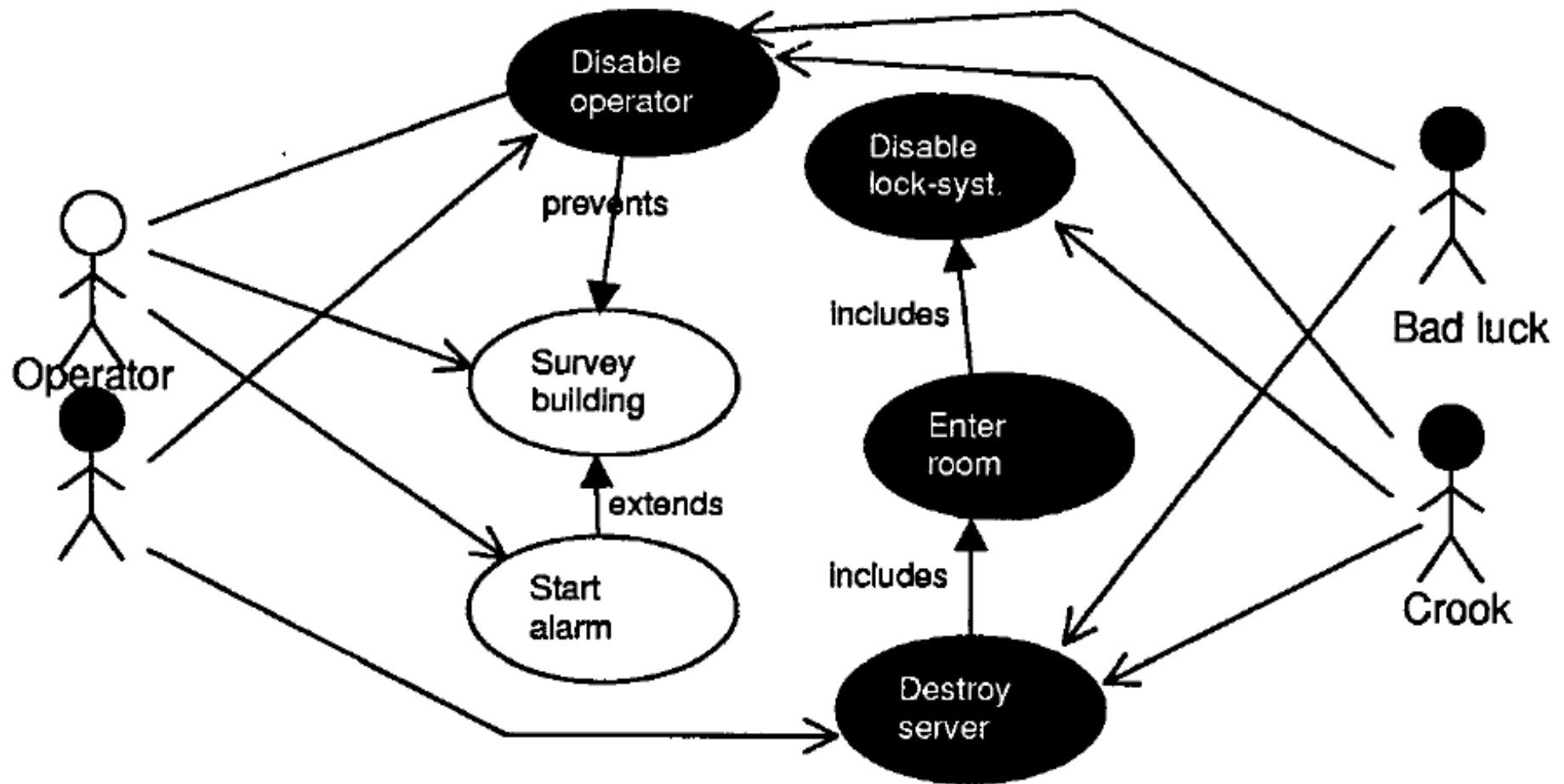
[亚历山大2003]

电子商务



G. Sindre和A. Opdahl。提升安全性
滥用要求
案件。在过程中。《TOOLS Pacific 20
00》。

特殊失误



优点

- 在软件开发过程的早期阶段关注安全性
- 增加发现原本可以忽略的威胁的机会
- 帮助跟踪和组织需求规范
- 帮助评估需求
 - 实施用例的实际成本包括减轻对它的所有严重威胁所需的保护
- 易于在新开发项目中重用

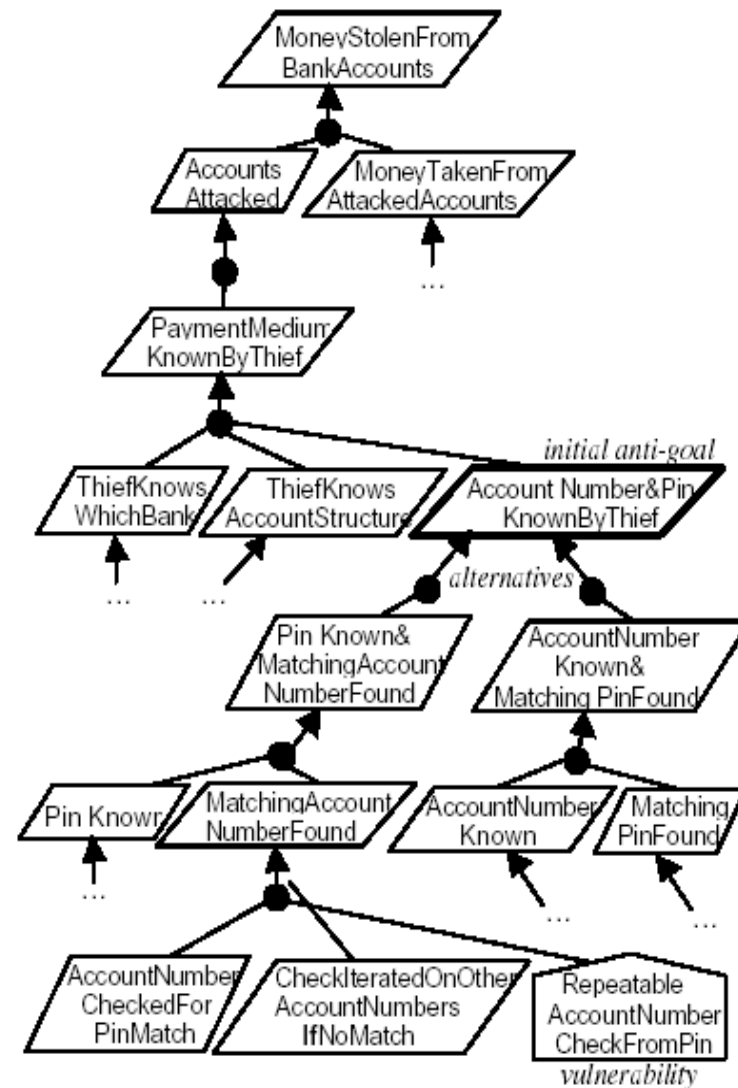
缺点

- 使用/滥用案例是非正式的
 - 没有明确的语义
 - （因此）没有形式分析
- 不知道怎么写 *高品质* 滥用案例
- 重点仅在于系统
- 不适用于各种威胁
- 并非总是有可识别的滥用者，滥用案例可能并不总是由可识别的动作序列组成

KAOS : 反目标

- 足以建模和解决非故意障碍 (偶然障碍) 的障碍
- 建模和解决故意障碍 (恶意障碍) 的限制
- 主动攻击者还可以与自己的目标 , 能力和可以监控或控制的漏洞一起建模 (反模型)
- 反目标是 安全目标的故意障碍

反目标 (KAOS)



[Van Lamsweerde 2004]

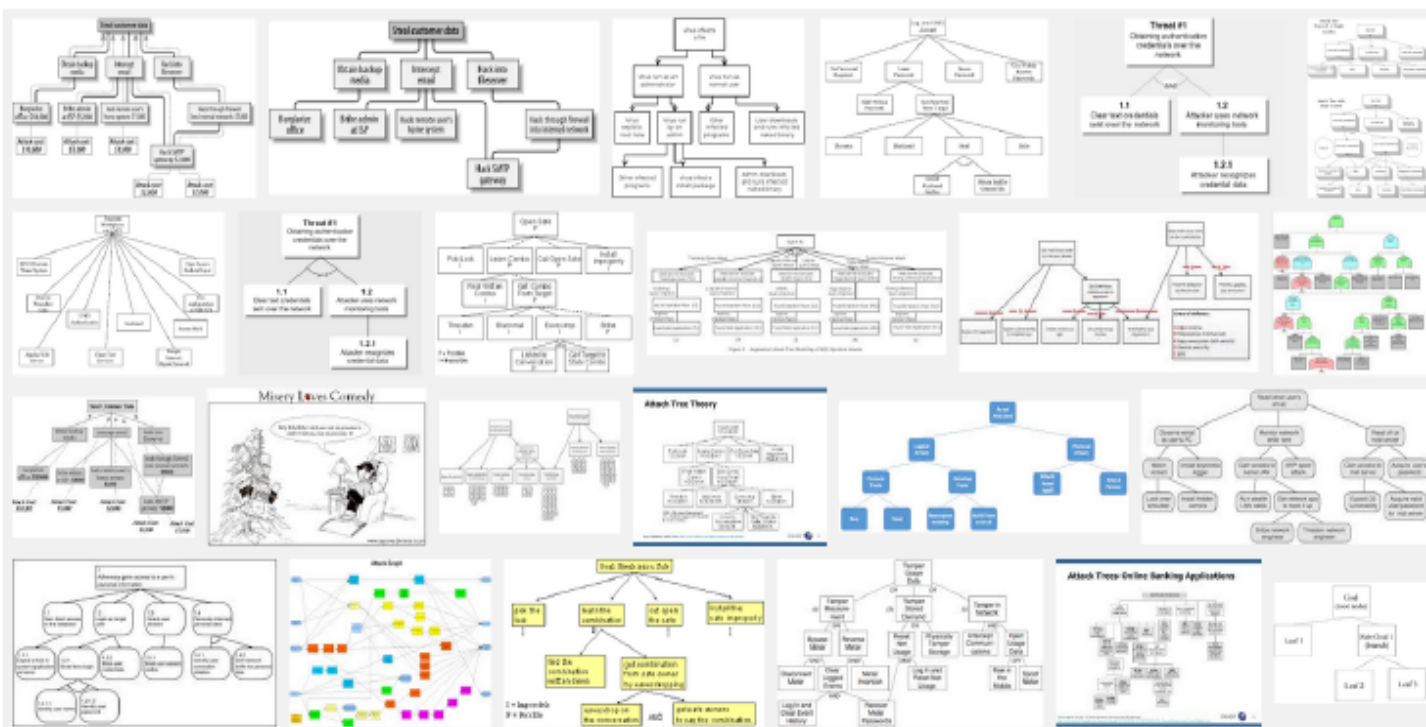
建立反模型

- 通过否定安全目标来获得根本的反目标。针对每个反目标，确定潜在的
- 攻击者 (WHO)
- 对于每个反目标和相应的攻击者，都会确定更高级别的反目标 (WHY)
- 对于每个反目标和相应的攻击者，确定较低级别的反目标 (HOW)
- 与/或改进目标的过程
 - 由攻击者实现 (反要求)
 - 被攻击者可以实现 (漏洞)
- 反模型源自反目标公式
- 反要求是根据相应攻击者的能力定义的

反目标的限制

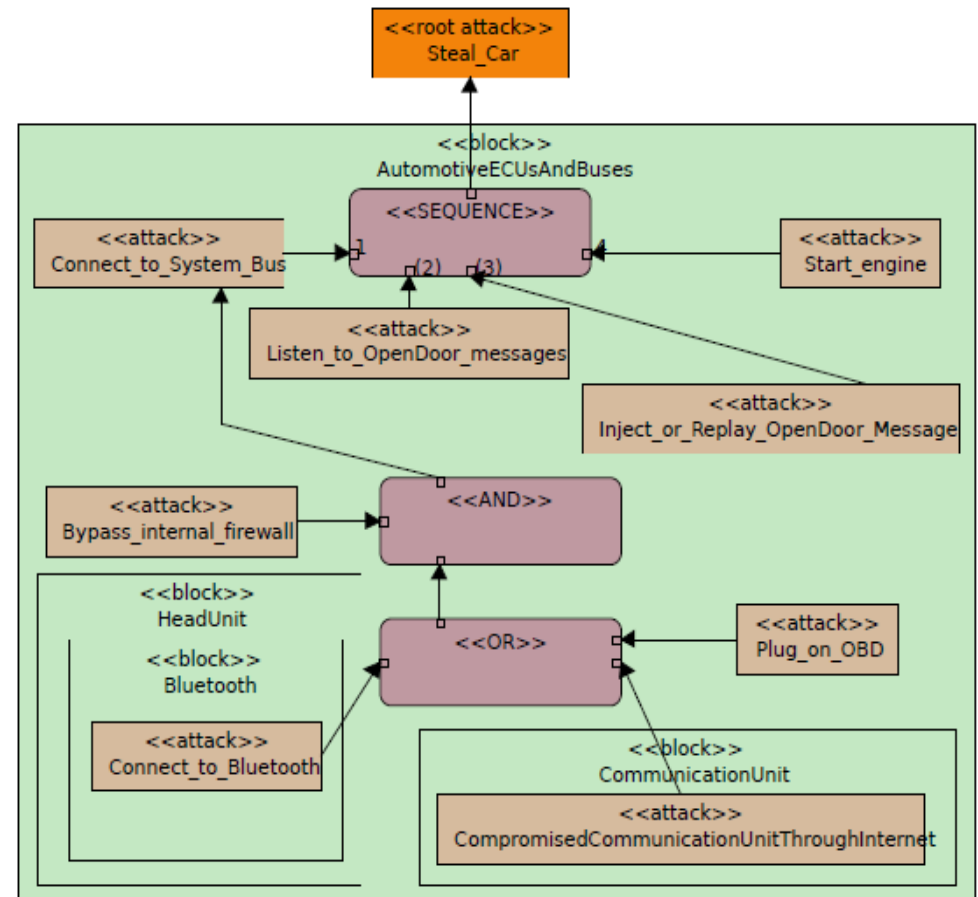
- 建模攻击者很困难
- 我们必须考虑所有可能的障碍，甚至是未知的障碍
 - 经过多年设计，许多安全协议被证明是错误的
- 许多系统漏洞取决于特定的实现
- 尚不完全了解软件漏洞



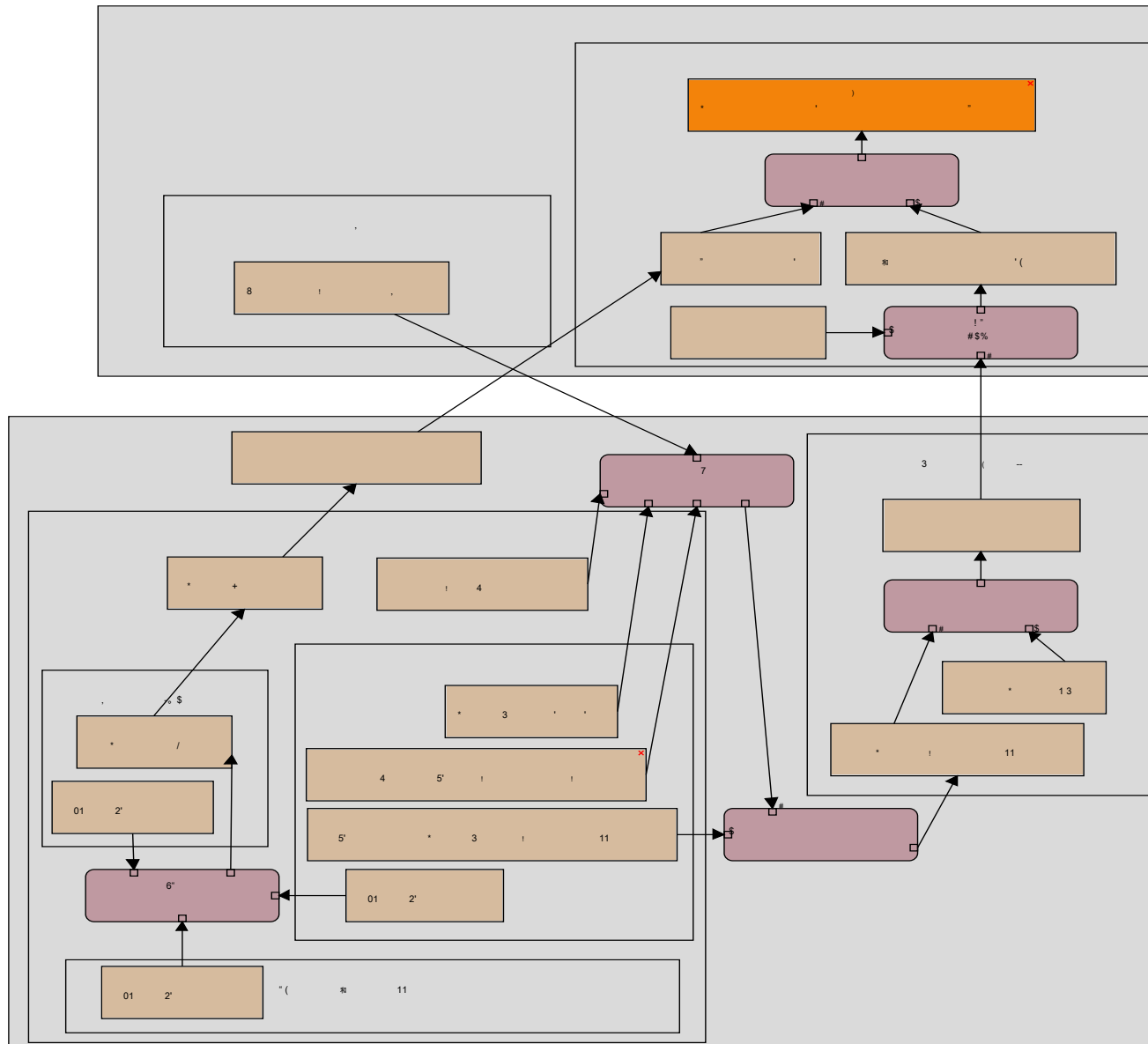


SysML-Sec : 从攻击树到攻击图

- 攻击之间的关系=约束
 - 逻辑 (AND , OR , XOR)
 - 排序 (顺序 , 之前 , 之后)
- 硬件/软件映射非常重要
 - 攻击记录和匹配对策
 - 架构攻击范围的形式分析
- 重用观点
 - 例如 , 更好的CVE文档



SysML-Sec : Zeus / Zitmo恶意软件攻击



攻击者系统

攻击目标
(Windows主机 ,
浏览器
移动电话)