对网络应用程序中的逻辑缺陷进行黑匣子检测

詹卡洛-佩莱格里诺
EURECOM, 法国
SAP产品安全研究，法国
giancarlo.pellegrino@eurecom.fr

Davide Balzarotti
EURECOM, 法国
davide.balzarotti@eurecom.fr

***摘要*-**
网络应用在许多关键领域发挥着非常重要的作用，包括网上银行、医疗保健和个人通信。这一点，再加上许多网络开发者的安全培训有限，使得网络应用成为攻击者最常见的目标之一。

在过去，研究人员提出了大量的白盒和黑盒技术来测试网络应用是否存在几类漏洞。然而，传统的方法主要集中在输入验证缺陷的检测上，如SQL注入和跨站脚本。不幸的是，针对特定应用程序的逻辑漏洞仍然不在大多数现有工具的范围内，仍然需要通过人工检查来发现。

在本文中，我们提出了一种新的黑箱技术来检测网络应用的逻辑漏洞。我们的方法是基于自动识别一些行为模式，从用户与某个应用程序互动的少数网络痕迹开始。基于提取的模型，我们按照一些常见的攻击场景生成有针对性的测试案例。

我们将我们的原型应用于七个真实世界的电子商务网络应用，发现了十个非常严重的、以前未知的逻辑漏洞。

I. 简介

网络应用程序在许多关键领域发挥着非常重要的作用，目前有数十亿用户信任它来进行金融交易、存储个人信息和与他们的朋友交流。不幸的是，这使得网络应用程序成为对各种恶意活动感兴趣的攻击者的主要目标之一。

为了减轻现有的威胁，研究人员已经提出了大量的技术来自动测试网络应用程序是否存在几类漏洞。现有的解决方案包括黑匣子模糊器和entesting从静态分析工具到静态分析系统，它们解析应用程序的源代码，寻找定义明确的漏洞模式。然而，传统的方法主要集中在对输入验证缺陷的检测上，如SQL注入和跨站脚本。到目前为止，针对特定应用程序的逻辑的更微妙的漏洞仍然是通过人工检查发现的[33]。

逻辑漏洞仍然缺乏正式的定义，但总的来说，它们往往是网络应用程序的业务流程没有得到充分验证的结果。由此产生的漏洞可能涉及控制面（即不同页面之间的导航）和数据面（即连接不同页面参数的数据流）。在第一种情况下，根本原因是应用程序未能正确执行用户执行的行动顺序。例如，一个应用程序可能不要求用户以管理员身份登录来改变数据库设置（认证绕过），或者它可能不检查购物车结账过程中的所有步骤是否按照正确的顺序执行。涉及应用程序数据流的逻辑错误是由于没有强制要求用户不能篡改在不同HTTP请求之间传播的某些值而引起的。因此，攻击者可以尝试重放过期的认证令牌，或将同一网络应用程序的几个并行会话获得的值混在一起。

描述内部状态的演变和预期用户行为的正式规范几乎从未用于网络应用。这种文档的缺乏使得我们很难发现逻辑漏洞。例如，虽然能够在购物车中多次添加相同的产品是一个常见的功能，但能够多次添加相同的折扣代码则可能是一个逻辑漏洞。人可以很容易地理解这两种情况的区别，但对于没有适当应用模型的自动扫描器来说，很难区分这两种行为。

直到最近，研究界才开始研究自动检测逻辑漏洞的方法[9, 18, 21]。不幸的是，现有的解决方案有严重的可扩展性问题，限制了它们对小型应用的适用性。此外，为了提取适当的模型来指导测试案例的生成，通常需要应用程序的源代码。因此，到目前为止，可用的自动化工具的影响是相当有限的。

作为一种替代方法，研究人员最近重新

NDSS'14，2014年2月23-26日，美国加州圣地亚哥。

http://dx.doi.org/10.14722/OETT.2014.23021

在现实世界的商业应用中[34,
35]，通过人工分析，暴露了一些严重的逻辑缺陷，例如，导致可以免费在线购物。沿着这些前人的工作步骤，在本文中我们表明，从用户"刺激"某种功能的一些网络痕迹开始，有可能自动推断出一个网络应用的近似模型。我们的目标不是自动重建应用程序或其协议的精确模型（在这个方向上已经有了一些工作[14,
15]），而是根据经验表明，即使是应用程序逻辑的简单表示，也足以进行自动推理，并生成可能暴露逻辑漏洞的测试案例。

在本文中，我们提出了一种技术，分析用户与某个应用程序的功能（如购物车）互动的网络痕迹。然后，我们应用一套启发式方法来识别可能与底层应用逻辑有关的行为模式。例如，总是以相同的顺序执行的操作序列，由服务器生成的值，然后在接下来的用户请求中重新使用，或者在同一会话中从未执行过一次的行动。然后通过执行根据一些攻击模式产生的非常具体的测试案例来验证这些候选行为。需要注意的是，我们的方法不是一个模糊器，跟踪分析和测试用例生成步骤都是离线进行的。换句话说，它们不需要探测应用程序或产生任何额外的互动和网络流量。

虽然我们的方法与应用无关，但攻击模式的选择反映了一类特定的逻辑缺陷和应用领域--
在我们的案例中是为电子商务应用定制的。特别是，我们将我们的原型应用于七个大型购物车应用，这些应用被数以百万计的网店所采用。该原型发现了10个以前未知的逻辑缺陷，其中5个允许攻击者少付钱甚至免费购物。

综上所述，本文有以下贡献。

1) 我们介绍了一种新的黑匣子技术来测试应用程序的逻辑漏洞。

2) 我们介绍了基于我们技术的工具的实现，并展示了该工具如何用于测试几个真实的网络应用，即使是在知识非常有限和网络痕迹数量很少的情况下。

3) 我们在著名的、大量部署的网络应用程序中发现了十个以前不为人知的漏洞。这些漏洞中的大多数具有非常高的影响，将允许攻击者从数十万家网店中免费在线购买。

*本文的结构。*第二节介绍了黑盒方法。第三节描述了我们所做的实验，第四节展示了实验的结果。第五节讨论了我们方法的局限性，第六节介绍了检测逻辑漏洞的相关工作。最后，第七节是本文的结论。

II. 办法

OWASP测试指南3.0[33]建议采用四步法来测试黑盒环境下的逻辑缺陷。首先，测试人员通过玩耍和阅读所有可用的文档来研究和了解网络应用。其次，她准备好设计测试所需的信息，包括*预期的工作流程*和*数据流*。然后，她继续设计测试用例，例如，通过重新安排步骤或跳过重要的操作。最后，她通过创建测试账户来设置测试环境，运行测试，并验证结果。

我们的方法旨在通过一个单一的黑箱工具来自动完成前面的步骤。首先，从包含HTTP对话的网络跟踪列表开始，我们的系统推断出一个应用模型，并将与同一工作流程"步骤"相关的资源进行聚类（第二节A）。第二，我们的技术对模型进行分析，并提取一组*行为模式*（第二节B），对应用程序的工作流程和数据流进行建模。第三，我们应用一套*攻击模式*来自动生成测试案例（第二节C）。最后，我们针对网络应用执行这些测试案例（第二节D），并使用一个*神谕*来验证应用的逻辑是否被违反（第二节E）。

在剩下的部分，我们以电子商务网络应用为例，详细描述每个阶段。

A. 模型推理

我们提出的技术是*被动的*和*黑箱的*。我们不需要访问应用程序的源代码（在客户端和服务器端），我们不主动抓取应用程序的页面，也不产生任何流量来探测其内部状态。相反，我们把HTTP对话的列表作为输入。这些痕迹可以由测试人员手动生成，也可以通过记录真实的用户活动收集。

为了简单起见，我们只考虑行使网络应用程序的特定功能的痕迹。例如，如果网络应用是一个购物车，我们就使用用户登录、向购物车添加物品、结账购买产品的痕迹。没有什么能阻止测试人员生成也包含其他功能的痕迹，比如浏览在线目录或发布产品评论。然而，只关注业务逻辑的一个方面有助于我们的系统用最少的输入痕迹找到相关操作。

网络应用程序通常涉及多个方面。例如，电子商务网络应用通常涉及客户、商店和支付服务。然而，他们之间的通信通常是通过客户端进行的，因此，我们把重点放在这一点上，以收集痕迹。此外，收集同一网络应用程序的不同部署的数据是很有用的，这样我们的推理方法就可以识别在某一安装中硬编码的参数值。

第一阶段包括建立应用程序的模型，称为*导航图*。这分两步完成：资源抽象和资源聚类。

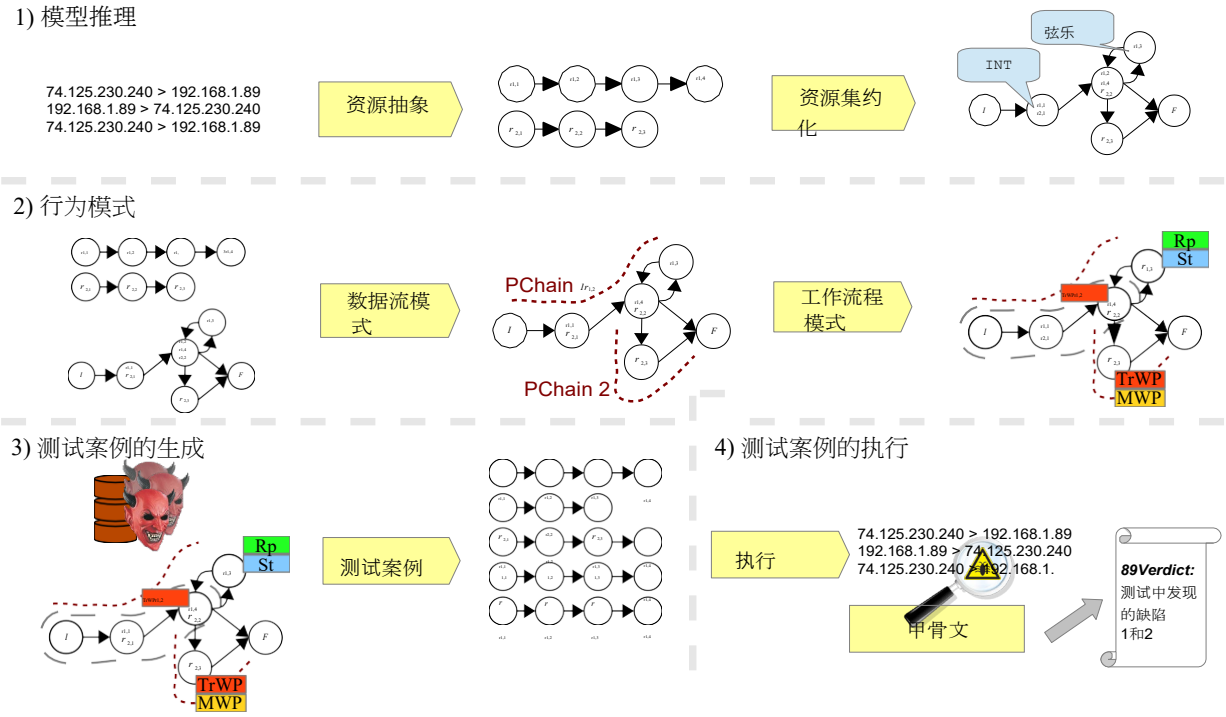

图1：我们方法的架构。

1) 资源抽象

输入轨迹是一对HTTP请求和响应的序列。推理阶段的第一步包括创建一个资源的综合体。我们的方法目前支持JSON数据对象[17]和HTML页面。然而，它可以很容易地扩展到其他类型，如SOAP消息[36]。

我们称*抽象的HTML页面为*（i）其URL的集合。(ii)POST数据，(iii)HTML代码中包含的锚点和表单及其DOM路径，(iv)元刷新标签中的URL，如果有的话，(v)HTTP重定向位置标头。我们称*抽象JSON对象为*(i)其URL的集合，(ii)POST数据，(iii)对象中的值和路径对，以及(iv)如果包含任何HTML代码的HTML链接。例如，图2显示了以下JSON对象的抽象资源。

```
{'项目'。{
   'item1':['价格':19.9,'税收':1.6],
   '项目2':[ ... ]}}
```

我们从每个抽象资源中提取一组元素，对应于所有可能出现在URL、POST数据和所有链接中的参数。每个元素都有一个名称、一个值、一个路径和一个推断的语法类型。我们的方法支持整数类型、十进制类型、URL类型、电子邮件地址类型、单词类型（按字母顺序排列的字符串，如
"添加"、"删除"，...）、字符串类型、列表类型（逗号分隔的值）和*未知*类型（即其他一切）。类型是通过检查元素的值与每个元素相关联的。明显的优先级规则被应用在

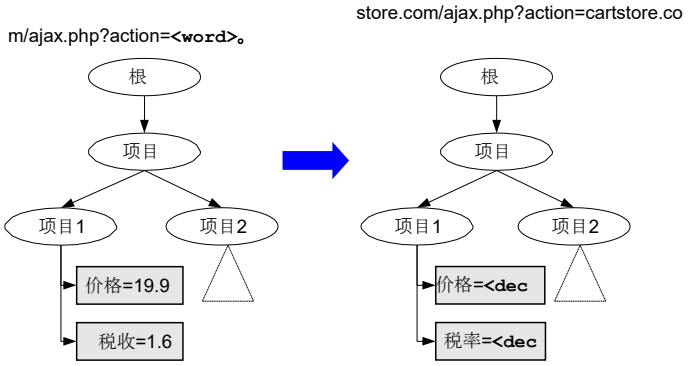

图2：JSON数据对象的资源抽象和句法类型推理

歧义的情况下--
例如，id=20既可以是数字也可以是字符串，但由于前者是后者的子集，它被认为是一个数字。

2) 资源集约化

现代网络应用程序将应用逻辑操作映射到不同的资源。例如，显示购物车的操作可能涉及一个包含网页骨架的初始HTML页面，然后使用一些异步AJAX请求来填充页面上的物品清单、税款、可用优惠券等。我们在三个阶段对这些资源进行分组。首先，我们将异步请求与发起这些请求的资源，即同步资源联系起来。然后，我们对资源进行分组，同时考虑

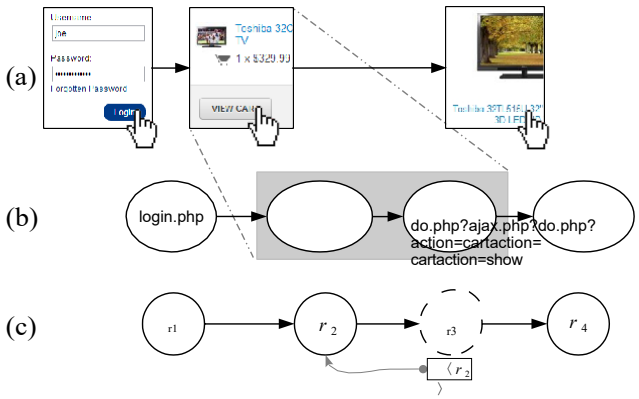

图3：(a)应用层面的行动，(b)请求的URL，和
(c) 附有原创者名单的抽象资源

相似性和发起人。第三，如果一个群组的资源参数编码为一个命令而不是携带一个值，我们就将其拆分。

在第一阶段，我们对输入痕迹进行预处理以识别AJAX请求。这可以通过检查 "X-Requested-With "HTTP请求头[32]或检测JSON响应来完成。之后，我们将每个资源与它的*发起人*联系起来。图3提供了这个第一阶段的一个例子。在图3.c中，我们有HTML页面r_1，后面是页面r_2。然后r_3通过使用$AJAX_2$请求r，$AJAX_2$用新的HTML代码或新的客户端脚本丰富了r。然后，这个例子以r结束4，我们假设它是由r中的一个链接引起的2，或由r添加的3。图3.c还显示了每个资源的发起人列表。r_1、r_2和r_4没有发起人，而r_3是由r发起的2。

在第二个阶段，我们对资源进行分组。一般来说，如果两个资源具有相同的URL域和路径，相同的GET/POST参数名称，如果有的话，还有相同的重定向URL，那么这两个资源就被归入同一个集群。在比较参数时，我们不考虑它们的值，只考虑它们的语法类型。例如，以下三个URL是等同的。

```
store.com/do.php?action=add&id=3
store.com/do.php?action=add&id=7
store.com/do.php?action=show&id=3
```

我们首先比较前面解释的同步资源，然后再比较异步资源。如果两个异步资源具有相同的URL域和路径、GET/POST参数名称、重定向URL和相同的发起人，它们就在同一个集群中。

在最后一个阶段，我们确定那些对命令进行编码的参数，而不是传输一个值。对于每个参数，我们采取与该参数具有相同价值的页面。例如，参数action将图4.a的灰色群组划分为两个子群组，一个是购物车值，一个是显示值。然后我们计算同一子群中的页面和不同子群中的页面之间的页面*相似度*。比较是通过查看HTML表单的DOM路径、其动作属性（URL域和参数名称）以及嵌套输入元素的名称来完成的。该功能是通过计算相似的页面的百分比来应用于子组。

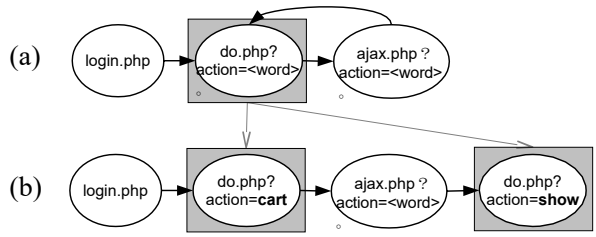

图4：(a)比较所有资源后的集群
(b)确定了编码命令的参数后的集群。

如果同一子群内部的相似度很高（超过55%），而不同子群之间的相似度很低（低于45%），那么我们就假定该参数是用来指定操作的，我们为每个值创建一个不同的节点。否则，我们就不修改该分组。这一阶段的结果如图4.b所示。

导航图是一个有向图G = (I, F , E) 其中 C [{} Gs是集群的集合，I是源节点，F是最终节点，E是边的集合。如果存在一个输入轨迹↑，其中一个资源r ′2 u紧接着一个资源r ″2
v，我们就放置边（u,
v）。然后，对于每个轨迹开始的每个r*j*（即↑ = hr*j*，. .i），我们在r *j*2
u的地方放置边（I，u），对于每个跟踪结束时的每个r, *j*（即↑ = h. . , r *j*i）我们在r *j*2
u的地方放置边（u，F）。

B. 行为模式

行为模式是工作流和数据流模式，可能与应用程序的逻辑有关。我们将工作流模式分为*跟踪模式*和*模型模式*，前者模拟用户在输入跟踪中通常做什么，后者模拟导航图允许做什么。最后，*数据传播模式*对数据如何在整个导航图中传播进行了建模。

1) 追踪模式

追踪模式是对用户在输入追踪中所执行的行动进行建模。特别是，我们专注于三种模式。

单子节点
如果一个节点从未被任何输入跟踪访问过一次，那么它就是一个单子。有些用户可能会访问这些节点，有些则不会--
但没有人访问它们两次。例如，提交折扣券可能是在一些输入追踪中观察到的操作，但没有人提交两次折扣券。

多步骤操作
多步骤操作是一个连续节点的序列，总是以相同的顺序访问。这在网络应用的许多功能中是非常常见的。例如，支付程序或用户注册通常包括一

个精确的步骤序列，并且所有的

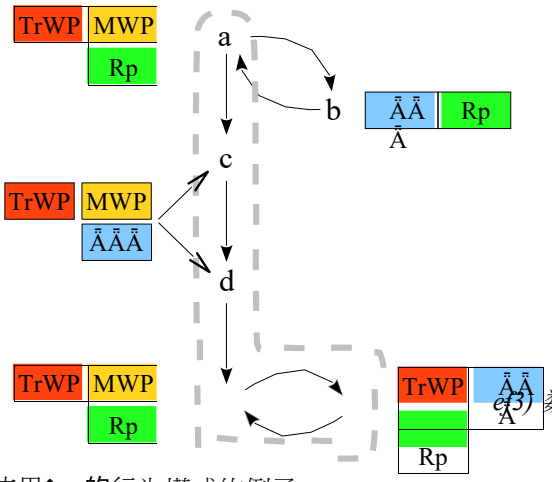

图5：使用↾1=的行为模式的例子。
ha, b, a, c, d, e, f, ei 和 ↾ 2= ha, c, d, e, f, ei

经历这些过程的轨迹总是以相同的顺序执行它们。

追踪航点

我们使用术语*waypoint*来描述在用户和应用程序之间的交互中起重要作用的节点。特别是，跟踪航点是那些出现在所有输入跟踪中的节点。例如，如果我们所有的痕迹都包含一个购买，那么重定向到支付网站（例如PayPal）就是一个痕迹航点。

2) 模型模式

模型模式是根据导航图允许的行动序列的模型。

可重复的操作

在导航图中属于循环的节点与有可能重复多次的操作相关。

模型航点

模型航点是属于导航图中从源节点到最终节点的每个路径的节点。这些节点不仅在所有输入轨迹中被访问，而且在导航图中没有办法绕过它们。根据定义，每个模型航点也是一个航迹航点，反之亦然。

图5显示了一个例子，以更好地描述模型和痕迹模式之间的区别。这个例子显示了从两个输入痕迹↾ 1= a, b, a, c, d, e, f, e和↾ 2= a, c, d, e, f, e中提取的导航图的行为模式。符号St、TrWP、Rp和MWP分别代表。
单子节点、跟踪航点、可重复节点和模型航点。粗的虚线划定了多步骤操作的范围。

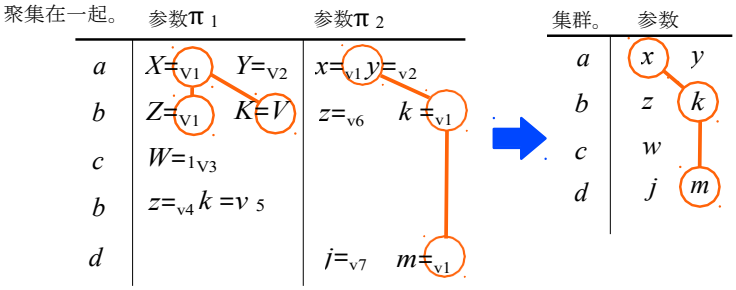

图6：传播链：从痕迹到导航图

e(3) 数据传播模式

传播链是一组具有相同价值的参数，这些参数在HTTP对话期间在客户端和Web应用程序之间来回发送。我们说，如果有一些输入轨迹中的两个参数持有相同的值，并且不存在值不同的轨迹（因为用户在所有的轨迹中并没有执行相同的操作，某个参数可能不会出现在所有的轨迹中），那么这两个参数就有相同的值。如果初始值是由用户选择的，我们就说该链是*客户端生成的*，否则就是*服务器生成的*。Wang等人[34]也使用了类似的分类。然而，他们的概念仅限于单个输入轨迹，而我们的概念则扩展到不同长度的轨迹和导航图。

我们分两步计算传播链。首先，我们确定一个跟踪中每个值的传播链。让我们考虑图6中的例子。这里，在输入轨迹↾1中，参数x的值与z的值相同。
k.在追踪↾2中，参数x仍然等于k，但它是
现在与z不同。此外，相同的值与
其次，通过比较追踪链，我们消除了矛盾，达到了图6右侧所示的结果。

C. 测试案例的生成

在本节中，我们描述了测试案例的生成。这是通过采用一些攻击模式来完成的，这些模式模拟了攻击者如何以非常规的方式使用应用程序。特别是，我们专注于攻击者可能执行的一系列行动：重复操作、跳过操作、颠覆操作顺序以及在用户会话中混合参数值。我们为每个动作设计了一个模式。这些模式呈现在图7中，并以图5的导航图为基础。我们用数字丰富了图7，以显示节点被访问的顺序。为了简单起见，我们省略了源节点I和最终节点F，它们分别与a和e相连。

需要注意的是，虽然本文提出的方法是通用的，但攻击模式的选择需要反映一类特定的逻辑缺陷（在我们的案例中，颠覆应用程序的控制或数据流）。其他类型的逻辑漏洞，如认证绕过，可能需要使用其他模式（例如，随机访问管理页面），这些模式可以添加到我们的系统中。

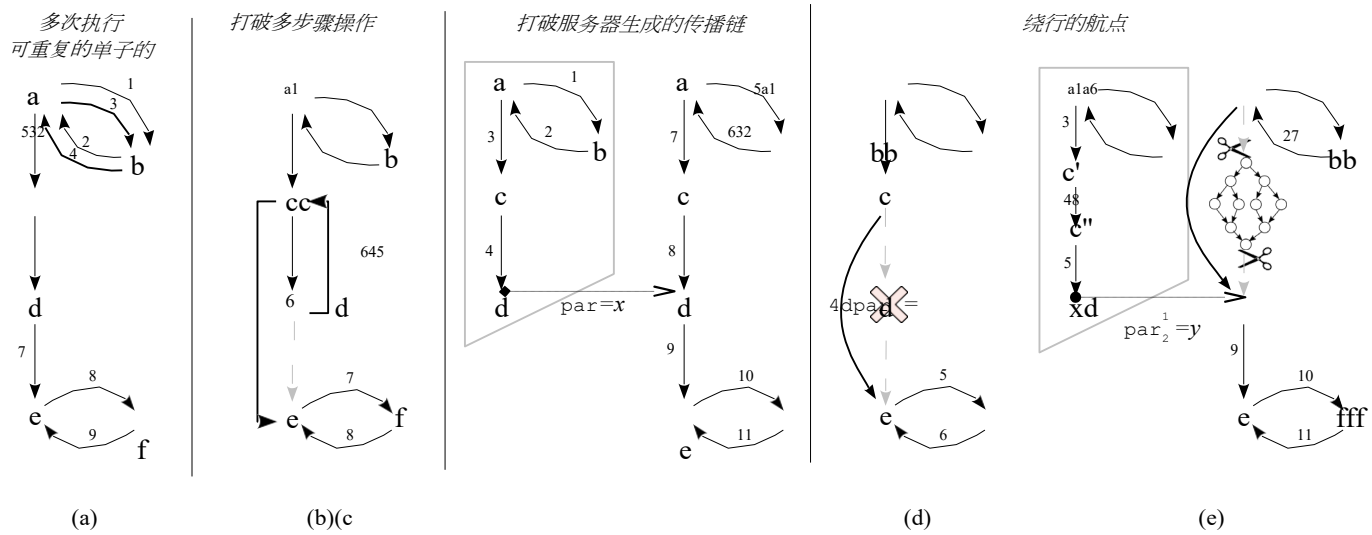

图7：测试用例生成模式

但这不在我们论文的范围之内。然而，使用自定义技术来检测某些漏洞是许多其他工具和方法所共有的--
例如，一个旨在发现SQL注入的技术不能被用来检测其他类型的输入消毒漏洞。

1) 可重复的单子的多重执行

这种模式为试图多次执行一个操作的攻击者建立了模型。如果该模式有一个节点是可重复的和单子的，这意味着即使有一种方法可以多次重复一个操作，但在我们的输入追踪中从未观察到这一点。因此，攻击者试图访问它两次。

图7.a显示了该测试案例的步骤。我们选择一个访问b的输入轨迹（例如，a， b，a， c， d，e， f， e），一个可重复的单子节点。然后，我们在单子节点之后将其分成两部分（例如，a， b和a， c， d， e， f， e）。我们称这两部分为*前缀*和*后缀*。其次，我们找到从单子节点到自身的最短循环（例如，b ， a ， b）。最后，测试案例是前缀、没有第一个节点的循环和后缀的串联。

2) 打破多步骤操作

这种模式模拟了一个破坏多步骤操作的攻击者。例如，一旦到达支付页面，攻击者就会返回并在购物车中添加一个项目。一般来说，有几种方法可以破坏图5的多步骤操作。第一种方法是使用不同的顺序（例如，a， d， f， e ， f）。第二种方法是将其他步骤交错进行。在这个模式中，我们专注于后一种方法，即在测试用例的后面重复已经包含在多步骤中的一个步骤。例如，在图7.b中的测试用例a, c, d, c, e, f 中，我们在d之后重复c。在这个模式中，我们在e和f之后也重复c， 但不在a之后。

3) 打破服务器生成的传播链

这种攻击模式的目标是篡改网络应用程序的数据流。图7.c显示了一个测试案例的例子。测试的第一部分与应用程序交互，并捕获服务器生成的传播链的值x。在第二部分，我们启动另一个会话，通过用x 替 换 par的值来中断传播链。

由于网络应用包含许多服务器生成的传播链（例如，所有的项目或消息ID），这种攻击模式可能产生非常多的测试案例。因此，我们只关注两种类型的传播链：含有独特值的传播链（即在所有输入轨迹中都不同，因此与会话有关）和含有安装特定值的传播链（即仅在同一安装中不变的值）。

测试用例的生成过程如下。首先，我们选择出现在HTTP请求中的属于链的参数。这些参数被称为*注入点*，是攻击者可以替换服务器生成的值的地方。例如，在图7.c中，节点d的参数par是一个注入点。其次，我们从访问注入点的节点的不同用户会话中选择两个痕迹。第一条在注入点处被截断，第二条被附加到第一条上。参照图7.c，这两个部分分别位于左手边和右手边。

4) 绕行的航点

Waypoints是指所有的输入跟踪都会执行的操作，如支付或提供运输数据。当这些操作在每个输入跟踪中只发生一次时，它们似乎表明网络应用程序的业务流程执行中的某种里程碑。在航点迂回模式中，攻击者试图通过使用两种可能的技术之一来跳过这些类型的操作。如果该航点节点不是传播链的一部分，我们就简单地尝试跳过它。

否则，我们会尝试删除两个航点之间的导航图部分，通过从另一个用户会话中获取丢失的数据值来重建传播链。

图7.d是这种模式的一个例子。在这第二种情况下，如果节点d的URL取决于出现在a和d之间的段中的一个值，我们通过选择一个输入轨迹并在d处中断它来准备另一个用户会话，这部分的生成与打破传播链相似。然后，第一个用户会话是a, b, a, c′, c″, d。之后，我们准备第二个用户会话，跳过a和d之间的序列。在这个例子中 h有两种可能性i：跳过b, a, c′, c″或者c′, c″。图7.e只显示了后者。在这种情况下，测试案例是a, b, a, c′, c″, d和a, b, a, d, e, f, e的串联。对于这种情况，我们也支持第一用户会话在节点d处不被打断的变体。

D. 测试案例的执行

II-C节中描述的测试用例是抽象的，仍然缺少正确执行的细节。例如，一些参数的值不能从模型中确定，需要在测试用例执行期间收集。此外，重要的是，在每次测试后，应用程序被重置为其初始状态，以避免连续执行之间的干扰。例如，一个测试可能会在购物车中留下一些物品，从而影响后续的实验。一般来说，在每次测试结束时，删除cookies并清空购物车通常就足够了。

执行引擎对测试案例的每个节点进行迭代，具体化POST/GET参数，并提交HTTP请求。响应根据传播链进行解析，以提取服务器生成的参数，用于后面的请求。如果执行引擎不能正确地重建一个链（例如，因为本应产生其值的页面返回了一个错误），执行引擎将中止执行，并报告说测试*没有被执行*；否则它将报告*已执行*。

E. 测试Oracle

我们在本文中提出的方法是完全独立于网络应用程序的业务逻辑的。我们的技术可以自动识别行为模式，然后生成测试用例，以多种不同方式打破这些模式。该系统还可以确定一个给定的测试是否被正确执行，但这是与应用无关的方法所能做到的。例如，在一个支付工作流程中替换安全令牌的值可能会使整个过程失败。不幸的是，在不了解底层业务逻辑的情况下，测试判决只能说该模式是否被成功应用，但它不能对可能的影响得出任何结论。因此，如果我们希望我们的工具能够报告可能违反应用逻辑的情况，我们需要提取测试期间发生的事件序列案例的执行，并将它们与我们想要违反的*逻辑属性*进行比较。

一个简单的表达购物车逻辑属性的方法可以是：*如果一个用户的订单被批准，那么该用户必须完成了相应金额的支付*。在这个表述中，有两个事件起着核心作用：一个是订单被下达的事实，另一个是用户支付了一定金额的事实。这个属性的另一个重要方面是这两个事件之间的时间依赖性。由于命题逻辑只能表达真理而不考虑时间，在我们的方法中，我们将逻辑属性建模为线性时态逻辑（LTL）公式[30, 23]。LTL在传统的逻辑运算符（如（和）、和=（暗示））上增加了时间连接词，如0（过去的一次）。这使我们能够验证一个事件最终会在未来发生还是已经在过去发生。

例如，上述逻辑属性可以用LTL写成如下。

$$\mathrm{ord}_{placed} \wedge \mathrm{onStore}(S) \Rightarrow \mathbf{O}(\mathrm{paid}(U, I)) \quad (1)$$

其中ord_{placed}, onStore(S), and paid(U, I)分别是*下的订单*，*在商店S上进行的操作*，以及*用户U支付了物品I的价格*。现在，识别违反逻辑属性的问题被重构为检查LTL公式是否被给定的测试案例所满足的问题。

在我们的方法中，*测试神谕*是一个组件，它给定一个测试案例的执行，如果违反了某个预定义的逻辑属性，则返回*真*，否则返回*假*。神谕由两部分组成：一个*事件提取器*和一个LTL公式检查器。提取器从已执行的测试中收集部分有序的事件集（事件可以按顺序或平行发生），并按用户会话分组。第二部分验证所有序列是否满足所提供的LTL公式。

值得注意的是，事件和LTL公式都取决于被测应用程序的类型和我们感兴趣的漏洞类型。例如，为了找到认证绕过的漏洞，观察与用户登录和访问私人页面有关的事件将是很有趣的。然而，在本文中，由于我们专注于电子商务应用的测试，我们更感兴趣的是监测资金转移和购买物品的价值，这在下一节有更详细的描述。

III. 实验

我们可以用我们的工具来测试网店（如亚马逊）。然而，我们的测试需要尝试畸形的操作并完成大量的结账过程。这既是不道德的，因为应用程序可能会因为我们的测试而发生故障，也是非常昂贵的，因为它要求为每个测试案例至少购买一个产品。因此，我们选择在七个著名的可用于离线测试的开放源代码应用程序上进行测试，如表所示

I. 该表还显示了对其受欢迎程度的估计，其衡量标准是通过执行以下的搜索结果

网络应用程序。	
安装数量 OpenCart	9,710,000
Magento	3,130,000
PrestaShop	650,000
CS-Cart	260,000
番茄车	119,000
osCommerce	80,500
AbanteCart	21,200
共计	13,970,700

表一:受欢迎程度指数

数的*googledorks*[1]。每个谷歌查询都是通过结合URL结构（例如，结账终端的路径）和从网页中提取的一些静态HTML内容（例如，页脚的 "powered by. . ."）建立的。因此，表格中报告的数字只是互联网上可公开获取的安装数量的下限。这种保守的测量表明，这七个应用程序被近1400万个电子商务装置所使用。作为比较，Wang等人[34]测试的两个应用程序使用类似的Google dorks返回不到40,000次点击。

A. 一般设置

我们为每个网络应用程序安装了两个实例（以下简称商店A和商店B）。除了AbanteCart和PrestaShop，所有的安装都被配置为使用PayPal Express Checkout [3]和PayPal Payments Standard [4]方法。我们总共准备了12种配置[1]。

所有应用程序都被配置为*沙盒*模式。在这种配置下，每个应用程序通过使用PayPal沙盒支付网关进行交易。这些付款不涉及真正的金钱，因为它们是在卖方和买方的测试账户之间进行的。

B. 测试Oracle

在他们的实验中，Wang等人[35]使用了以下购物车属性。

*"当且仅当(i)S拥有*I*；(ii)在CaaS中保证将付款从*U*的账户转到*S*的账户；(iii)付款是为了购买*I*，并且只对一件*I*有效。*
*(iv) 此项付款的数额与*I*的价格相等。"*

然而，这一属性在黑箱环境中并不是完全可以验证的。例如，我们不可能测试 "S拥有I"这一谓语的真实性，也不可能检查到期的金额是否已经转到商家的账户上。因此，我们通过删除不可验证的条款来简化上述不变式。可用于自动黑盒测试的新属性变成了。

[1]当我们做实验时，AbanteCart和PrestaShop分别只提供PayPal Payments Standard和PayPal Express Checkout。

*当商店S确认用户*U 已 经*下了订单，那么在过去，*U*向S支付了相当于*I*的价格的金额，*U*同意从S购买*I。

我们使用在每个测试案例执行过程中提取的以下事件对逻辑属性进行建模。

- 当商店确认订单已经下达时，就会*发出*指令。
- onStore(S)，当对商店S进行了操作后。
- paid(U, I)，当用户U授权支付网关支付I的价格。
- toStore(S)，当付款是针对商店的时候 S;
- ack(I)，当用户确认购买I时，逻辑属性被表

述为：。

$$\text{ord}_{placed} \wedge \text{onStore}(S) \Rightarrow \Diamond(\text{paid}(U, I) \wedge \text{toStore}(S) \wedge \Diamond(\text{ack}(U, I) \wedge \text{onStore}(S)) \quad (2)$$

C. 输入痕迹

为了产生输入痕迹，我们创建了两个用户账户，U_1和U_2，每个账户控制一个PayPal买家测试账户。对于每个网络应用，我们总共捕获了六个HTTP对话，每个商店有三个：一个是U_1购买一件商品，一个是U_2购买另一件商品，还有一个是U_1购买两件不同的商品。所有的输入痕迹都满足逻辑属性2。这些输入痕迹足以刺激购物车的主要功能，但未来可以使用更好的训练来暴露更微妙的特征，或检测不同类型的逻辑缺陷。

D. 测试案例的生成

表二显示了按攻击模式分组的测试案例。测试用例的生成产生了大约3100个测试用例，平均每个应用有262个。表二还显示了测试执行的结果。当测试用例使应用程序处于无法继续的状态（例如，由于中间步骤的错误页面）时，执行失败。这是一个常见的结果，因为根据定义，我们的测试强调应用程序暴露出一些意外的行为。违反LTL公式的测试案例的数量在表三中报告。如前所述，有些事件对神谕是不可见的。因此，对LTL公式的违反并不总是对应于一个漏洞。事实上，有可能在应用程序的后端进行的进一步检查会发现并阻止攻击。为了区分逻辑漏洞和其他错误（例如，错误地将失败的交易报告给用户为成功），我们手动检查了商家的资产负债表、订单列表和它们的状态。每当结果没有被我们的人工确认时

		测试案例的生成				测试案例的执行			
网络应用。		时间 hh:ss	(a)(b)(c(d), (e)			时间 hh:ss	执行者。	不执行。	共计
	阿班	☒ 00:01				04:51	74	159	233
特卡丁车		00:02	95121152			16:23	240	103	343
	淘宝网		10825246						
	准则	00:02	14627303			14:50	210	176	386
	淘宝网	00:01	1077383			02:34	140	33	173
	准则	00:01	15382260			02:08	71	64	135
	创业者	☒ 00:01				03:22	117	48	165
		00:01	4136142			03:42	128	97	225
	准则		86310144						
	兴发	☒ 00:01				02:42	85	52	137
兴发		00:02	12223100			04:54	238	64	302
	Tom atoCartExp		96810215						
	准则	00:02	173237138			04:36	115	109	224
	CS-CartExp	00:05	8246562			12:02	347	253	600
	准则	00:02	165415137			05:29	127	95	222
共计			132	586	145	2282		1892	1253	3145

表二：每个应用程序在测试用例生成和测试用例执行阶段的统计数据。列（a）、（b）、（c）、（d）和（e）是图7中的攻击模式，而列Exec.和Not Exec.指的是测试执行引擎的两种可能结果。

网络应用。		没有。小提琴。	引起的原因是 ulns。	BugsV
阿班特卡丁车	准则	17	16	1
Magento	解释	65	65	-
	准则	126	126	-
淘宝网	解释	58	46	12
	准则	30	30	-
奥斯曼	解释	42	22	20
	准则	35	34	1
淘宝网	解释	-	-	-
番茄车	解释	90	65	25
	准则	24	24	-
CS-Cart	解释	313	313	-
	准则	109	108	1
共计		909	849	60
		100%	93.4%	6.6%

表三：违反属性2的测试案例的数量和根本原因。

检查，我们将其归类为正常的错误。其余的案例则对应于与真正的软件漏洞有关的异常行为，这一点在下一节中解释。值得注意的是，由我们的方法产生的测试案例中，超过28.9%的案例使应用程序处于违反LTL公式的状态，52个测试中的1个暴露了一个以前未知的逻辑漏洞。

测试用例的生成不需要太多的资源，而执行阶段可能

IV. 结果

相当耗时（Magento为16小时）。这主要是由于在我们的实验中缺乏并行化，以及PayPal沙盒比它的实时对应物慢得多的事实。模型推理

- 从图二中省略--每个应用平均需要9分钟来建立导航图，平均包含34个节点和48条边。

表三报告了违反安全属性2的总次数。换句话说，根据我们的攻击模式，通过篡改工作流程或数据流，我们的系统能够在909个案例中使网络应用处于故障状态。所有这些情况都对应于测试，这些测试一直执行到最后一页，在这一页中，商店祝贺顾客成功购买（这导致了事件的产生，*ordplaced* onStore(S)），尽管支付的金额不正确。虽然这些违规行为都是应用程序代码中的bug造成的后果，但并不是所有的都能被攻击者利用。

这是重要的一点，也是黑盒方法的一个基本限制。我们的工具只能"从外部"观察应用程序的状态，因此它不能区分表现形式的错误（网页上显示的信息是错误的，但应用程序的内部状态是正确的）和更严重的漏洞（内部状态也被破坏了）。

为了区分这两种类型的bug，我们手动检查了后台数据库的状态：结果是表三中总结的无害的表现bug（93.4%）和真正的漏洞（6.6%）之间的区别。虽然这些结果表明我们的工具的*真实*阳性率是6.6%，但也有剩余的93.4%的违规行为对应于应用程序中的真实漏洞，需要由开发人员来修复。一旦所有的表现问题都被解决了，我们的工具发出的警报就只对应于可利用的漏洞。

A. 脆弱性

表三显示，我们的测试案例中有60个（占总数的1.9%）暴露了目标应用程序中的逻辑漏洞。我们发现了以下的缺陷。

在osCommerce 2.3.1, CS-Cart 3.0.4, and Abante-Cart 1.0.4 with PayPal Payments Standard a malicious

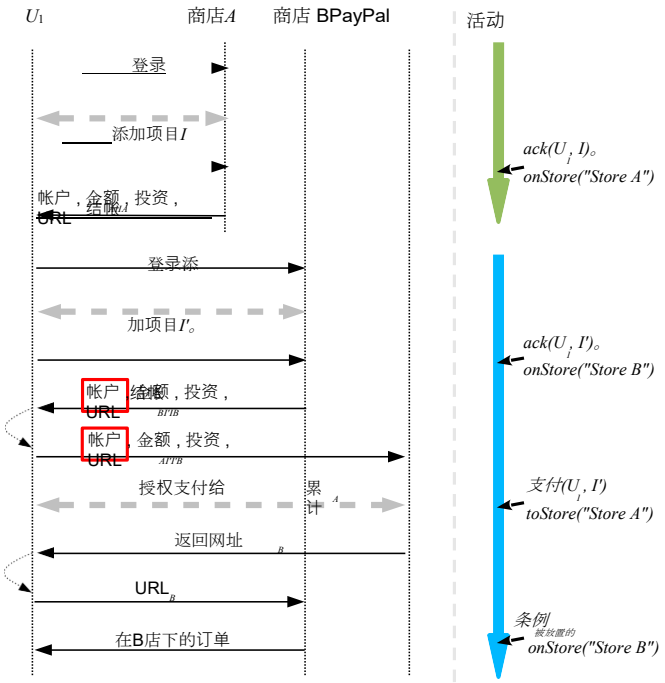

图 8: 使用osCommerce 2.3.1和Aban- teCart 1.0.4免费购物

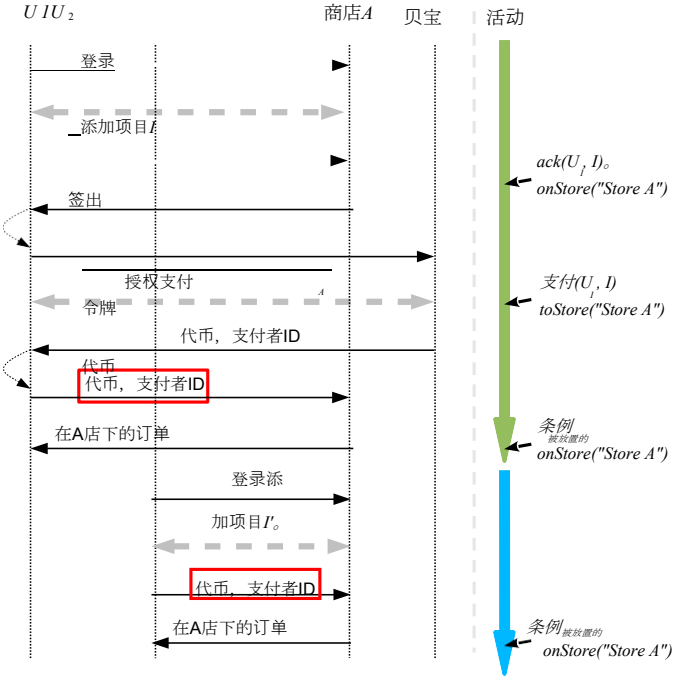

图 9: 使用OpenCart 1.5.3.1和Tomato-支付的费用较少。购物车1.1.7

顾客可以免费购物（可利用）。

- 在OpenCart 1.5.3.1和TomatoCart 1.1.7与PayPal Express Checkout中，恶意的客户可以少付一些钱（可利用）。
- 在TomatoCart 1.1.7与PayPal Express Checkout中，一个恶意的客户可以免费购物（可利用）。
- OpenCart 1.5.3.1, TomatoCart 1.1.7, and osCommerce 2.3.1 with PayPal Express Checkout a customer can pay a amount different from what she authorized (not exploitable)
- TomatoCart 1.1.7带PayPal Express Checkout，一个客户支付另一个客户的购物车（不可用）。

所有可被利用的缺陷都已被负责任地披露。当开发者在我们通知的两周内没有回答时，我们也向美国Cert报告了这些漏洞[2]。在下文中，我们描述了我们在实验中发现的每一类漏洞。

1) osCommerce, CS-Cart, and AbanteCart with PayPal Payments Standard - Shopping for Free

这些缺陷是通过测试发现的，这些测试中断了服务器生成的传播链，传输了商户的PayPal账户。图8中显示了一个例子。图的左边是消息序列图，右边是按用户会话分组的事件。每个用户会话从一个登录信息开始。这些事件显示了甲骨文是如何发现违规行为的。在

在执行结束时，$ordplaced$ ^ onStore("Store B")被满足，因为其中的所有事件都被观察到。然而，公式(1)的左侧没有得到满足，因为其中的事件都没有被观察到。

人工检查验证了（i）没有向B店付款，（ii）B店后台的订单状态是"已完成"，以及（iii）发票已支付。把上述测试变成一次真正的攻击是很简单的。事实上，当重定向到PayPal时，攻击者可以用她控制的另一个PayPal账户替换卖家的PayPal账户。在这种情况下，攻击者可以为她在网上商店购买的物品向自己付款。

2) 带PayPal Express Checkout的OpenCart和TomatoCart--少付费

在OpenCart和TomatoCart的PayPal Express Checkout中，攻击者可以支付低于物品价值的费用。该漏洞是通过使用航点迂回模式检测出来的。测试案例生成器为OpenCart和TomatoCart分别生成了11个测试案例，其中用户U_2跳过重定向到PayPal付款的节点，用U的用户会话的值重建URL_2。
一个有代表性的测试案例显示在图9。在第二个用户会话中，$ordplaced$ onStore("Store A")被满足。然而，该公式的其他条款没有得到满足，因为既没有观察到用户的确认，也没有观察到付款。

人工检查发现，在订单清单中，有两个不同的订单，一个是I的，另一个是I的[0]。两张订单都处于"已付"状态，并准备发货。然而，I的资产负债表

[2]见http://www.kb.cert.org/vuls，ID为459446、207540和583564。

商家只包含I的交易，而没有任何被记录为$\mathbf{I}^{0}$.

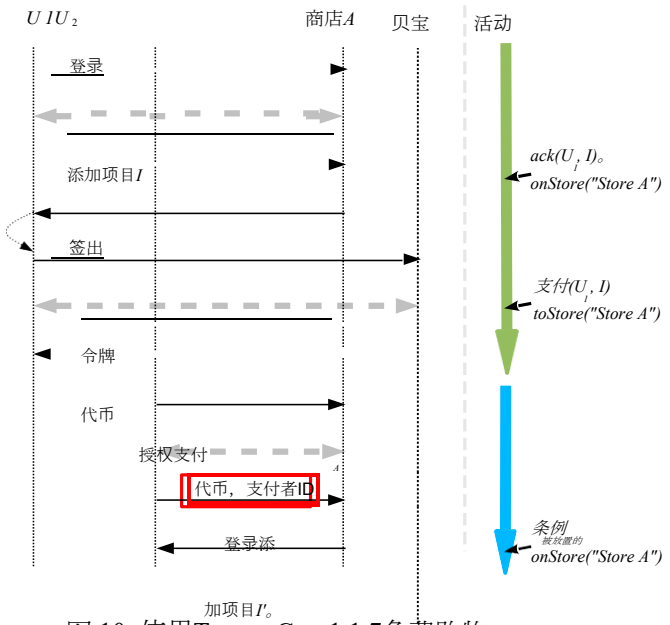

图 10: 使用TomatoCart 1.1.7免费购物

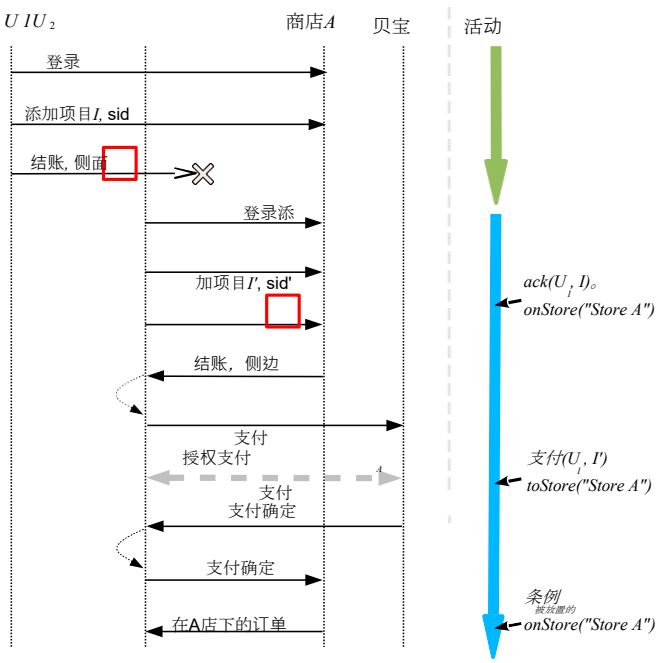

图 11: TomatoCart 1.1.7中的会话固定功能

这个测试可以变成一个攻击，首先购买一个便宜的商品，并拦截从PayPal到商店的重定向URL。然后，攻击者可以再次登录，将昂贵的商品添加到购物车，并重放之前捕获的URL，完成交易。更糟糕的是，我们验证了攻击者（或任何其他用户）可以重复使用相同的Token ID和PayerID来完成任意数量的额外假交易。这个过程只受PayPal对令牌设置的超时限制。

3) 带PayPal Express Checkout的TomatoCart - 免费购物

这个问题已经被用航点迂回模式生成的11个不同的测试案例所确认。一个有代表性的测试案例显示在图10中。图10显示，在第二个用户会话中，$\hat{}$ *ordplaced* onStore("Store A")被满足。然而，该公式的其他条款没有得到满足，因为既没有观察到用户的确认，也没有观察到付款。

人工检查证实，没有为I和为I付款。然而，订单列表中的I的订单处于"已付款"状态，并准备发货。这个测试案例可以变成前面所示的攻击，不同的是，攻击者在收到PayPal的Token和PayerID后结束第一个用户会话。

4) osCommerce, OpenCart和TomatoCart与PayPal Ex- press Checkout - Pay Less

在osCommerce中，测试是由waypoints迂回模式产生的，而在OpenCart和TomatoCart中，测试是由打破服务器生成的传播链产生的。

在osCommerce中，测试类似于图10所示，而对于OpenCart和TomatoCart，测试类似于图8所示。当PayPal Express Checkout是选择，商店和PayPal通过重定向来交换Token。在这里，当用户被重定向到PayPal进行支付时，该模式中断了Token的链条。在这两个案例中，甲骨文验证了用户U_2有一个确认，并且pay(U_2,I) ^ toStore(A)被满足。然而，甲骨文无法验证O(ack(U_2,I) ^ onStore(A))，因为它观察到O(ack(U_2,I) ^ onStore(A))。

人工检查证实，只有I的订单在订单列表中的状态是"已支付"，而I的订单仍然是"待支付"。然而，在商户的资产负债表中，I的付款是由U_1而不是U完成的。
在这种情况下，U_1授权PayPal为I付款，而她的信用卡却为I付款。

为了将这种测试变成真正的攻击，攻击者需要拦截通过SSL/TLS通道进行的重定向URL。此外，它必须阻止用户-受害者执行重定向。这可能需要攻击者打破SSL/TLS加密层，或者进行SSL/TLS MITM（中间人）攻击。然而，在这两种情况下，攻击者将能够捕获受害者的支付数据，使她在任何情况下都能免费购物。

5) 带PayPal Express Checkout的TomatoCart--会话修复

我们的实验发现了一个会话固定的漏洞，其中U_2可以冒充另一个用户。测试案例是通过在两个点上打破参数sid的传播链创建的。图11显示了其中的一个。图11的事件不符合公式，因为付款I'的金额与用户承认的I的金额不同。

参数sid在cookie中带有相同的值，破坏它将导致*会话固定*，在我们的案例中，U_2

从那时起，U_2可以访问U的数据$_1$。
因此，U_2（现在以U_1的身份登录）支付了U的购物车$_1$。
然而，我们无法找到任何利用这一缺陷的方法$_1$。假设受害者（即U_2）"点击
"由攻击者（U_1）制作的URL，那么受害者可以在三个时刻注意到欺诈行为（i）在检查订单摘要时，（ii）在提供送货地址时（显示攻击者的地址），和（iii）在付款时，因为金额不同。

V. 限制条件

我们的方法使用了篡改观察到的数据流和工作流程的攻击模式。然而，它没有测试其他类型的逻辑漏洞，如对资源的未授权访问。此外，我们没有考虑攻击者也可以扮演恶意商店的角色的情况，也没有考虑攻击者可以拦截和篡改应用程序和支付服务之间的信息的情况。我们相信，我们的技术也可以有效地检测其他类型的逻辑缺陷，尽管我们没有对这一假设进行实验测试。这可以通过增加特权用户（如管理员）的输入痕迹，增加其他行为模式，或增加新的攻击模式来实现。

第二，测试的生成偏向于效率而不是覆盖率。这意味着每个测试类别只使用几个值，以最大限度地在有限的时间内发现漏洞。对攻击空间进行更彻底的探索可以用来发现更多的漏洞，然而这可能需要大量的执行时间。本文的重点是展示如何利用自动化方法来发现许多实词应用程序的逻辑漏洞，而不是深入分析一个单一的应用程序（这种情况也需要更多的输入痕迹来更好地探索应用程序的逻辑）。

最后，我们用LTL对逻辑属性进行建模。LTL的使用使我们能够验证具有时间依赖性的事件。然而，LTL不支持代数的条款在执行的不同时刻出现。例如，我们的神谕不能验证付款是否是用户在过去某个时刻添加到购物车中的物品的总和。有一些工作通过对整数的约束来扩展LTL[10]，它们可以被我们的口令用于检查更精细的属性。

VI. 相关的工作

已经提出了大量的解决方案来检测Web应用程序中的漏洞。然而，以前的工作大多集中在自动检测与输入验证不足有关的著名漏洞类别，如跨站脚本（XSS）[26]、跨站请求伪造（CSRF）[2, 27]和SQL注入[22]。由于我们的目标是寻找逻辑缺陷，我们将不在本节中介绍这些解决方案。

a) 逻辑漏洞的检测

当应用程序的源代码可用时，诸如MiMoSA[9]、Waler[21]和Swaddler[16]等工具可用于发现逻辑漏洞。MiMoSA和Waler从源代码中提取一个模型，然后使用模型检查器来检测违反不变性的情况。Swaddler[16]在软件处于其生命周期的部署阶段时检测攻击。它首先学习应用程序的正常行为，然后在运行时监控状态变量，寻找偏离正常行为的情况。

当源代码不可用时，提取模型的问题变得更加困难。Doupe´等人[19]和Li和Xue[28]提出了两个黑盒测试工具。前者提出了一个状态感知的输入模糊器来检测XSS和SQLi漏洞。该工具推断出一个模型，作为选择下一个要抓取的URL的神谕来使用。我们的方法和这种技术都推断出模型，以提高对漏洞的自动检测。然而，我们使用的是为生成检测逻辑缺陷的测试案例而定制的被动学习技术，而不是驱动输入模糊器的主动扫描。第二项工作提出了BLOCK，一个通过观察HTTP对话来学习模型和不变式的工具，然后检测认证绕过攻击。与BLOCK不同的是，我们的方法不以拦截攻击为目的，而是为检测缺陷而生成安全测试。这两项工作都不能用来发现这一类的漏洞。前者的工作提出了一个带有输入模糊器的有状态的爬虫，不试图违反应用程序的逻辑。后者的重点是通过推断会话变量不变性来检测认证绕过攻击。

与BLOCK类似的一个方法是InteGuard[37]。InteGuard的目的是保护多方网络应用，防止API集成中的漏洞被利用。InteGuard主要关注浏览器中继消息，其中数据值在各方之间通过网络浏览器进行交换。特别是，InteGuard使用基于数据流分析和差异分析的被动模型推理技术来提取服务间数据流相关的不变性。前者用于提取数据值的流动，而后者则用于检测数据流的属性，如交易特定值或实施特定值。我们的方法使用类似的技术来提取这些类型的不变因素。然而，除此之外，它还提取了应用程序的可观察工作流的不变性，并考虑到了服务内的不变性，例如，相同的有效操作，以及服务间的不变性，例如，多步骤操作。

鉴于自动黑箱技术的成功率有限，最近有人提出了人工方法学。我们的工作主要受到Wang等人[34, 35]的启发，他们提出了对基于CaaS（Cashier as a Service）的网络商店的分析，以及对网络单点登录协议的大规模分析。前一项工作描述了一种黑箱方法，给定一些HTTP对话，标记API参数，并显示攻击者可以用哪些参数来试图违反安全不变性。后者通过(i)考虑攻击者在协议执行过程中扮演的角色和(ii)为协议参数添加语义和句法标签来完善前者。这两种技术

由于作者能够在现实世界的应用中发现严重的漏洞，因此产生了很大的影响。然而，这些论文提出了需要由安全专家手动应用的技术和准则。我们的工作在四个方面扩展了他们的技术。首先，它从一组HTTP对话中推断出一个模型。第二，它将单个跟踪的传播链的概念概括为应用模型的传播链。第三，它推断出业务功能的工作流程的可观察特征。最后，它自动生成并执行使用一些攻击模式的测试案例。

AUTHSCAN [8]是一种与我们的工作类似的方法。它结合白盒和黑盒技术从实现中推断出一个模型。AUTHSCAN专注于检测认证协议特有的缺陷（参见Lowe等人[29]对认证属性的调查），它需要一个特定于应用程序的JavaScript函数签名列表，以便推断出协议参与者的准确模型。相反，我们的方法专注于业务相关的网络应用程序属性，并使用独立于应用程序的模型推断技术。

b) 模型推理

有大量的工作在解决为测试目的推断模型的问题。模型推断分为两类：主动学习和被动学习。主动学习技术与被推断的应用进行交互，以探索其行为，而被动学习技术则从一组观测数据中建立模型。Hossen等人[25]提出应用主动学习算法L*[5]来推断一个确定性的有限自动机，并通过测试对其进行完善。Dury等人[20]描述了一种基于网络商业应用的被动学习的方法。他们使用了参数化有限自动机（PFA），它丰富了有限自动机的经典概念[24]，并在过渡和状态上设置了守卫参数。PFA为应用程序的控制流和数据流建模。使用数据挖掘算法如C4.5[31]来推断防护措施。然后，模型被翻译成Promela语言，并送入模型检查器SPIN[23]，以验证与应用相关的属性。然而，在第一种方法中，作者提出了一个方向，对他们旨在检测的缺陷类型说得很少，而在第二种方法中，作者专注于推理部分，没有涉及实际测试。

c) 基于模型的安全测试

为了在有模型的情况下使用模型对网络应用程序进行（半）自动安全测试，人们提出了新的想法。例如，Armando等人[7]提出了检测逻辑缺陷和测试基于网络的安全方案。该方法包括使用基于SAT的模型检查器[6]来验证形式化规范与安全要求的关系。如果发生违规行为，就会针对真实的实现来执行。Büchler等人[13]提出了一种方法，假设(*i*)给出了一个模型(*ii*)并且该模型是安全的。然后他们建议通过注入漏洞来突变模型，并使用模型检查器来检测违规行为。如果发现问题，那么他们使用模型检查器返回的反例作为测试的抽象测试案例。

实现。Bodei等人[11]提议在CaSPiS（带有管道和会话的服务微积分）中对面向服务的应用程序进行建模，这是一个带有会话和管道概念的过程微积分[12]，以进行控制流分析，检测应用程序的滥用情况。作者在CyberOffice购物车的一个已知的脆弱版本上测试了他们的技术，检测到了价格修改攻击。然而，所有这些工作仍然存在一个问题，即应用程序的模型在实践中往往无法使用。

VII. 结论

在本文中，我们提出了一种新的技术，用于对网络应用中的逻辑缺陷进行黑箱检测。我们的方法使用了一种被动的模型推理技术，从一组网络跟踪中建立了一个导航图。然后我们应用一些启发式方法来提取可能与底层应用逻辑有关的行为模式。这些行为，连同一些攻击模式，被用来生成测试案例。

我们开发了一个原型工具并测试了七个电子商务应用。该原型生成并执行了3100多个测试案例，其中900个案例违反了应用程序的预期行为。结果，我们的工具在被测试的应用程序中发现了10个以前不知道的逻辑漏洞。其中五个允许攻击者少付钱甚至免费购物。

鸣谢

这项工作得到了欧盟第七框架计划的部分支持，拨款协议号257007（项目SysSec）和257876（项目SPaCIoS服务互联网的安全供应和消费）。

参考文献

[1] "谷歌黑客数据库在为慈善而黑。"[在线]。Available: http://www.hackersforcharity.org/ ghdb/

[2] "Requestrodeo:客户端对会话的保护，"在*OWASP欧洲2006年会议上，报告CW448，Departement Computerwetenschappen，KU Leuven，2006年5月*，2006。

[3] "Paypalexpresscheckout整合指南"，2012年8月。[在线]。Available: https://cms.paypal.com/cms content/US/en US/files/developer/PP ExpressCheckout IntegrationGuide.pdf

[4] "Paypalpaymentsstandard integrationguide"，2012年6月。[在线]。Available :https://cms.paypal.com/cms content/US/en US/files/developer/PP WebsitePaymentsStandard IntegrationGuide.pdf

[5] D.Angluin, "Learning regular sets from queries and counterexamples,"
Inf.Comput.，第75卷，第2期，1987年11月。

[6] A.Armando, R. Carbone, and L. Compagna, "Ltl model checking for security protocols," in *Computer Security Foundations Symposium, 2007.CSF '07.第20届IEEE*，2007年7月，第385-396页。

[7] A.Armando, G. Pellegrino, R. Carbone, A. Merlo, and D.Balzarotti,
"从模型检查到安全协议的自动测试。弥合差距"，在 *TAP*，ser.LNCS, A. D. Brucker and J. Julliand, Eds., vol. 7305.Springer, 2012.

[8] G. Bai, J. Lei, G. Meng, S. S. Venkatraman, P. Saxena, J.Sun, Y. Liu, and J. S. Dong, "Authscan:Automatic extraction of web authentication protocols from implementations," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, San Diego, California, USA, February 24-27, 2013。

[9] D.Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna, "Multi-module vulnerability analysis of web-based applications," in *Proceedings of the 14th ACM conference on Computer and Communications Security*, ser.CCS '07.美国纽约：ACM，2007。

[10] M.M. Bersani, L. Cavallaro, A. Frigeri, M. Pradella, and M. Rossi, "Smt-based verification of ltl specifi- cations with integer constraints and its application to runtime checking of service substitutability, " *CoRR*, vol. abs/1004.2873, 2010.

[11] C.Bodei, L. Brodo, and R. Bruni, "Static detection of logic flaws in service-oriented applications," in *ARSPA-WITS*, ser. LNCS, P. Degano和L. Vigo, Eds., vol. 5511.Springer, 2009.

[12] M.Boreale, R. Bruni, R. De Nicola, and M. Loreti, "Sessions and pipelines for structured service programming," in *FMOODS*, ser.LNCS, G. Barthe and F. S. de Boer, Eds., vol. 5051.Springer, 2008.

[13] M. Büchler, J. Oudinet, and A. Pretschner, "从安全模型对网络应用进行半自动安全测试", in *SERE*.IEEE，2012年。

[14] J.Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher:使用自动协议逆向工程实现主动僵尸网络渗透，"在*第16届ACM计算机和通信安全会议论文集中*，芝加哥，IL，2009年11月。

[15] P.M. Comparetti, G. Wondracek, C. Kruegel, and E.Kirda, "Prospex: Protocol specification extraction," in *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, ser.SP '09.Washington, DC, USA: IEEE Computer Society, 2009, pp.[在线]。Available: http://dx.doi.org/10.1109/SP.2009.14

[16] M.Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler:基于异常的网络应用状态检测方法，"在*RAID*, ser.LNCS, C. Kruegel, R. Lippmann, and A. Clark, Eds., vol. 4637.Springer，2007年。

[17] D.Crockford, "RFC4627: The application/json media type for javascript object notation (json)," July 2006. [在线].Available: http://tools.ietf.org/html/rfc4627

[18] A.Doupé, B. Boe, C. Kruegel, and G. Vigna, "Fear the ear: discovering and mitigating execution after redirect vulnerabilities," in *Proceedings of the 18th ACM conference on Computer and Communications Security*, ser.CCS '11.New York, NY, USA: ACM, 2011.

[19] A.Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "En- emy of the State:A State-Aware Black-Box Vulnerability Scanner"，载于*2012年USENIX安全研讨会（USENIX 2012）论文集*，华盛顿州贝尔维尤，2012年8月。

[20] A.Dury, H. H. Hallal, and A. Petrenko, "从商业应用的痕迹中推断be- havioural模型", in *Proceedings of the 2009 IEEE International Conference on Web Services*, ser.ICWS '09。美国华盛顿特区：IEEE计算机协会，2009。

[21] V.Felmetsger, L. Cavedon, C. Kruegel, and G. Vi-gna, "Toward automated detection of logic vulnerabil- ities in web applications," in *Proceedings of the 19th USENIX conference on Security*, ser.USENIX Security'10。Berkeley, CA, USA: USENIX Association, 2010.

[22] W.G. Halfond, J. Viegas, and A. Orso, "A Classification of SQL-Injection Attacks and Countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, Arlington, VA, USA, March 2006.

[23] G. J. Holzmann, *The SPIN Model Checker - primer and reference manual*.Addison-Wesley, 2004.

[24] J.E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*.美国马萨诸塞州波士顿：Addison-Wesley Longman Publishing Co., Inc., 2006。

[25] K.Hossen, R. Groz, and J. Richier, "Security vulnerabilities detection using model inference for applications and security protocols," in *Software Testing, Verification and Validation Workshops（ICSTW）, 2011 IEEE Fourth International Conference on*, march 2011.

[26] M.约翰斯，"网络应用中的代码注入漏洞。以跨站脚本为例"，博士论文，2011年。

[27] N.Jovanovic, E. Kirda, and C. Kruegel, "Preventing cross site request forgery attacks," in *SecureComm*.IEEE，2006。

[28] X.Li and Y. Xue, "Block: a black-box approach for detection of state violation attacks towards web applications," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ser.ACSAC '11.New York, NY, USA: ACM, 2011.

[29] G. Lowe, "A hierarchy of authentication specifications," in *Computer Security Foundations Workshop, 1997.Proceedings., 10th*, 1997, pp.31-43.

[30] A.Pnueli, "The temporal logic of programs," in *FOCS*.IEEE计算机协会，1977。

[31] J.R. Quinlan, *C4.5: 机器学习的程序*。圣

美国加州旧金山：摩根考夫曼出版公司，1993年。

[32] jQuery基金会，"jQuery"，2013年1月。[在线]。Available: http://jquery.com/

[33] OWASP基金会，"OWASP测试指南"，2008年12月。[在线]。Available: https://www.owasp.org/index.php/OWASP_Testing_Project

[34] R.Wang, S. Chen, and X. Wang, "通过FACEBOOK和GOOGLE把我签到你的账户上：商业部署的单点登录网络服务的流量引导安全研究。"
在2012年IEEE安全和隐私研讨会上。IEEE计算机协会，2012年。

[35] R.Wang, S. Chen, X. Wang, and S. Qadeer, "How to shop for free online - security analysis of cashier-as-a-service based web stores, " in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser.SP '11.Washington, DC, USA: IEEE Computer Society, 2011.

[36] World Wide Web Consortium, "Simple Object Access Protocol (SOAP) 1.2," April 2007.[在线]。见：http://www.w3.org/TR/soap/

[37] L.Xing, Y. Chen, X. Wang, and S. Chen, "Integuard:Toward automatic protection of third-party web service integrations," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013*, San Diego, California, USA, February 24-27, 2013.