检测电子商务应用中的逻辑漏洞

孙芳琦　　徐亮芳振东

加州大学戴维斯分校
{fqsun, leoxu, su}@ucdavis.edu

***摘要*-**
电子商务已经成为一种繁荣的商业模式。由于很容易获得各种工具和第三方收银机，创建和启动电子商务网络应用是很简单的。然而，要创建*安全的应用*仍然很困难。虽然第三方收银员有助于弥合商家和客户之间的信任差距，但收银员作为新的一方的参与使结账过程的逻辑流程变得复杂。即使是结账过程中的一个小漏洞也可能导致商家的经济损失，因此逻辑漏洞对电子商务应用的安全构成了严重威胁。由于逻辑流的多样性和结账过程的复杂性，执行人工代码审查是具有挑战性的。因此，开发自动检测技术很重要。

本文首次提出了对电子商务网络应用中的逻辑漏洞的*静态检测*。自动检测的主要困难是缺乏一个普遍而精确的*正确支付逻辑*的概念。我们的主要观点是，安全的结账过程有一个*共同的不变因素*。当一个结账过程保证了关键支付状态（订单ID、订单总额、商家ID和货币）的*完整性*和*真实性*时，它就是安全的。我们的方法结合了符号执行和污点分析，通过跟踪污点支付状态和分析商家、收银员和用户之间的关键逻辑流来检测违反不变量的情况。我们已经为PHP实现了一个符号执行框架。在我们对22个独特的支付模块的评估中，我们的工具检测到12个逻辑漏洞，其中11个是新的。我们还在现场网站上成功进行了概念验证实验，以证实我们的发现。

I. 简介

电子商务网络应用程序是一种为网上购物而设计的特殊类型的网络应用程序，在现代世界中发挥着重要作用。美国商务部人口普查局估计，2013年第二季度美国电子商务零售额达到648亿美元，比前一年增长18.4%[28]。互联网的普及和智能移动设备的崛起促进了电子商务网络应用的快速增长。不幸的是，电子商务应用的复杂性和第三方收银机API的多样性使得实施完美安全的结账过程变得困难。由于逻辑攻击是直接与

NDSS'14，2014年2月23-26日，美国加州圣地亚哥。

http://dx.doi.org/doi-info-to-be-provided-later

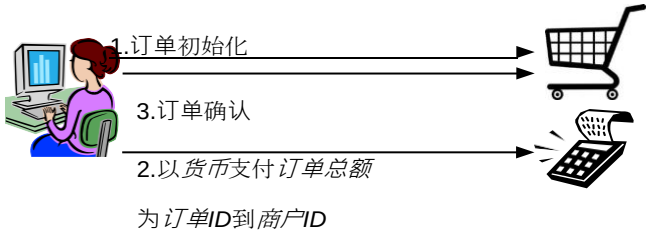

图1：电子商务网络应用的逻辑流程。

从经济损失到商家的尴尬，电子商务应用中的逻辑漏洞的影响往往是严重的。

业务或应用逻辑指的是*应用程序特定的*功能和行为。除了一般的功能（如用户认证），每个应用程序都有其独特的处理用户输入、用户行动和与第三方组件的通信。虽然逻辑漏洞不是最常见的网络漏洞类型，但它往往具有严重的影响，而且容易被利用。当攻击者违背开发者的意图，滥用合法的应用程序特定功能时，通常会存在逻辑漏洞[10]。WhiteHat Security的一份报告列出了七个逻辑漏洞的例子[16]。当构建一个应用程序时，开发人员往往在他们的头脑中对理想的应用程序有一个清晰的印象。不幸的是，在实践中，实现的应用程序往往比预期的要多。换句话说，出乎意料的用户输入和逻辑流会让攻击者以危险的方式滥用没有充分防护的特定应用功能。逻辑流的独特性和复杂性使建立一个针对特定应用程序攻击的一般防线变得复杂。

电子商务应用中的逻辑漏洞，作为一般逻辑漏洞的一个*子集*，允许攻击者以不正确或不付款的方式购买产品或服务，由商家承担费用。开发人员经常对用户输入的内容以及用户在结账时如何浏览网页做出假设。然而，当这种假设不成立，并且开发人员未能实施适当的安全检查时，攻击者可以利用电子商务应用程序中的逻辑漏洞来获取经济利益。CVE-2009-2039[9]描述了我们的激励性例子，Luottokunta（版本1.2），osCommerce软件[1]中的一个支付模块，有一个逻辑漏洞，允许攻击者篡改订单ID、订单总数和商家ID。最新版本的Luottokunta（1.3版）发布后，通过对支付状态的一些组件增加逻辑检查来修补这个漏洞。然而，经过仔细检查，我们惊讶地发现，它

图2：从易受攻击的网站收到的产品。

仍然是脆弱的。增加的对订单ID的检查是不充分的，因此攻击者可以为一个订单付款而绕过对未来订单的付款。这是我们发现的新漏洞之一。

在电子商务应用中使用第三方收银机会带来新的安全问题，即使收银机本身是安全的。为了提高灵活性，一个现代的网络应用程序通常在结账时通过为每个第三方收银台使用一个支付模块来展示几个支付选项。然而，收银机的整合也增加了结账过程中逻辑流的复杂性。图1说明了一个典型的结账过程中的三个关键步骤，其中涉及到一个商家、一个收银员和一个用户。1）在商家的服务器上启动订单，2）在收银员的服务器上进行支付交易，以及3）在商家的服务器上确认订单。在第一步中，商家启动订单的基本支付信息。从那时起，商家和收银员都会跟踪订单的支付状态。理想情况下，商家应该明确检查重要支付状态的*每一个*组成部分，或者直接与收银员沟通。在实践中，商家和收银员之间的错误沟通可能会损害支付状态的完整性或真实性。对支付状态的逻辑检查不充分或缺失，会使攻击者跳过第二步或错误地执行第二步。在一个成功的攻击中，商家会被误认为订单已被全额支付，而收银员实际上没有收到付款或部分付款。

为了证实电子商务应用中的逻辑漏洞所带来的真正危险，我们按照Wang *等人*[30]所设定的测试条件设计了负责任的概念验证实验。每个实验都是在一个使用有漏洞的支付模块的真实网站上进行的。具体来说，我们收到了三个网站的三个产品（图2），这些网站集成了有漏洞的支付模块。首先，对于支付模块RBS WorldPay，我们从Canonical有限公司的Ubuntu在线商店收到了一台Ubuntu笔记本。我们通过将货币从英磅换成英镑，少付了钱。美金。第二，对于支付模块Authorize.net信用卡SIM，我们从一家婴儿用品网店收到了一个尿布游戏包。我们通过重新播放之前订单的代币，没有支付任何费用。第三，对于支付模块PayPal Standard，我们从一家加州巧克力网店收到了三块巧克力。我们通过将商家ID从巧克力店主的ID改为我们的ID，没有向商家支付任何费用。在收到产品后，我们立即向三个商家赔偿了各自正确的全款金额。这些实验清楚地表明，不安全地使用第三方收银员，如经过严格审查的收银员PayPal，可能会给商家带来错误的保护感。

检测电子商务应用中的逻辑漏洞对人工和自动分析都是一种挑战，因为结账链中的任何薄弱环节都可能导致逻辑漏洞。一方面，手工代码审查很耗时且容易出错。安全分析员在检查支付状态的安全检查之前，往往要花很多时间了解电子商务应用中的不同逻辑流程。相反，支付模块开发人员熟悉逻辑流程，但不熟悉各种攻击载体。在这两种情况下，对结账过程中所有可能的逻辑流进行彻底的手工代码审查是一项不简单的任务。另一方面，如果不了解特定应用的业务背景，自动代码扫描器无法检测到逻辑漏洞。电子商务应用有各种特定的应用逻辑流程，每种支付方式有其独特的API和安全检查。因此，创建一般的规则来实现检测过程的自动化是具有挑战性的。

研究人员提出了各种技术来检测不同的逻辑漏洞，包括异常的逻辑行为[15]、多模块漏洞[3]和单点登录漏洞[31, 33]。每种技术都针对逻辑漏洞的一个特定领域，并根据给定领域的规范检查网络应用。Wang *等人*[30, 33]是第一个对基于Cashier-as-a-Service的电子商务应用进行安全分析的人。他们通过人工代码审查在一些流行的电子商务应用中发现了几个严重的逻辑漏洞[30]，并提出了一个基于代理的方法来动态地保护第三方网络服务的集成，包括收银机的集成[33]。

在本文中，我们首次提出了电子商务应用中的逻辑漏洞的*静态检测*。我们的关键观察点是，必须验证一个不变量才能保证付款的安全。商家M应该接受一个用户的订单O，当且仅当该用户已经以正确的金额和货币向收银员支付了与商家M相关的特定订单O。基于这一观察，我们设计了一个符号执行框架，详尽地探索关键控制流，跟踪支付状态的关键部分（订单ID、订单总额、商家ID和货币）的污点注释，以及暴露的签名令牌。我们的主要贡献是。

- 我们提供了一个独立于应用的不变因素，用于检测电子商务网络应用中的逻辑漏洞，并发现了一个新的攻击载体：篡改货币。
- 我们提出了第一个基于符号执行和支付状态污点跟踪的静态分析方法来检测电子商务应用程序的逻辑漏洞。
- 我们为PHP网络应用实现了一个可扩展的符号执行框架。我们的分析器系统地探索了控制流，以检查结账过程中的逻辑流。
- 我们在22个独特的真实世界的支付上评估我们的工具

在22个支付模块中，有12个模块存在逻辑漏洞，而这些模块来自不同的收银台。我们还在实时网站上进行了负责任的概念验证实验。在检测到的12个漏洞中，有11个是新的。评估结果表明，我们的方法是有效和可扩展的。

本文的其余部分组织如下。我们首先给出一个例子来说明我们方法的主要步骤（第二节）。第三节描述了我们的详细算法和方法。第四节介绍了我们开发的自动分析器的实现，第五节展示了漏洞报告、我们在真实网站上的实验细节以及我们的工具在真实世界的电子商务支付模块上的表现。最后，我们调查了相关工作（第六节）并得出结论（第七节）。

II. 说明性的例子

本节使用支付模块Luottokunta（版本1.3）来说明我们方法的主要步骤。这个模块修补了CVE-2009-2039[9]中描述的漏洞，但由于对不信任的订单ID检查不充分，所以仍然存在漏洞。在结账过程中，用户发出了以下四个关键的HTTP请求，其中最后两个是状态代码为302的HTTP响应的重定向。

```
R1.用户>商家(checkout_confirmation.php) R2.用户
> 收银员(https://dmp2.luottokunta.fi) R3.用户 >
商家(checkout_process.php), 302 R4.用户 > 商家
(checkout_success.php), 302
```

通过这个支付模块，商家可以整合第三方收银员Luottokunta的服务。在四个请求中，第二个请求被发送给收银员，其余的被发送给商户。当用户浏览到checkout_confirmation.php页面时，第一个请求（R1）初始化了订单的结帐过程。第二个请求（R2）让用户将商家生成的订单信息传递给收银员。当用户在收银员的服务器上完成支付交易后，收银员向用户发送一个响应，将用户重定向到商家服务器上的checkout_process.php页面（R3）以处理订单。如果订单被接受，商家会将用户重定向到checkout_success.php页面（R4）。

我们的符号化执行从结账过程中的第一个商家页面checkout_confirmation.php开始。为了模拟第一个请求（R1），它象征性地执行这个页面的中间表示（IR），并同时解析其象征性的HTML输出以寻找关键的HTML表单元素。该分析最终找到了一个HTTP表单，作为商家和收银员之间的通信渠道。它的元素记录了订单信息，其行动URL指向收银员的URL（https://dmp2.luottokunta.fi）。这个表单还包含一个返回的URL（checkout_process.php），一旦支付交易完成，收银员将使用这个URL将用户重定向到商家的服务器。

由于我们的分析将收银员视为能正确工作的黑匣子，我们假设收银员会正确地完成与用户的支付交易（R2）并将用户重定向到商家（R3）。为了继续探索逻辑流程，我们的分析器象征性地执行了checkout_process.php页面，这是返回URL的一部分。彻底的检查需要对收银台的所有可能的反应进行建模。因此，我们对R3的请求变量使用象征性的顶值（ ），*即*表示任何可能的值的最保守的值。我们的分析器首先将终端执行状态从上一个页面checkout_confirmation.php传播到当前页面checkout_process.php，然后象征性地执行当前页面的IR。执行最终到达函数before_process()，该函数对支付状态有以下检查。

```
功能 before_process() {
  if (!isset($_GET['orderID']))
    {
    tep_redirect(FILE_PAYMENT);
  } else {
    $orderID = $_GET['orderID'];
  }

  $price = $_SESSION['order']->info['total'];
  $tarkiste = SECRET_KEY
    .$price .$orderID .MERCHANT_ID。
  $mac = strtoupper(md5($tarkiste))。

  如果(($_POST['LKMAC'] != $mac)
      && ($_GET['LKMAC'] != $mac))
    { tep_redirect(FILE_PAYMENT);
  }
}
```

因为请求变量$_GET['orderId']有一个象征性的顶值，第一个if语句的两个分支都是可行的。对于真分支，用户被重定向到商家页面FILE_PAYMENT。这个重定向形成了一个向后的流程，这对检测逻辑漏洞没有帮助。因此，这个后向逻辑流被自动丢弃了。对于假的分支，一个MD5值被计算并存储在变量$mac中。请注意，计算中使用的$orderID的值来自于一个不受信任的请求变量$_GET['orderId']，它是在攻击者的控制之下。

我们的污点分析跟踪结账过程中跨逻辑流的关键支付状态的组成部分。最初，订单ID、订单总额、商家ID、货币和秘密都是污点。*秘密是*指只有商家和收银员知道的不可预测的值。因此，收银员可以用它来签署信息。对于污点的操作，我们有一套规则。当一个条件检查针对一个受信任的组件验证一个不可信任的值时，有一条规则会删除污点注释。对于函数before_process()中的最后一个条件，我们对假分支有以下符号约束。

```
[或
  ($_POST['LKMAC'] = strtoupper(md5(SECRET_KEY)
    .$_SESSION['order']->info['total']
    .$_GET['orderID'] .merchant_id))。
  ($_GET['LKMAC'] = strtoupper(md5(SECRET_KEY)
    .$_SESSION['order']->info['total'] 。
    .$_GET['orderID'] .merchant_id))。
]
```

在上述约束条件中的符号值中。
$_SESSION['order']->info['total']
是一个可信的

会话值，而MERCHANT_ID和SECRET_KEY是在商家的数据库中定义的可信常数。这种条件性检查保证了收银员已经代表商户收到了全额付款。因此，我们的分析器删除了订单总额、商家ID和秘密的污点注释。相比之下，$_GET['orderId']是一个不受信任的请求变量，而且没有对货币的检查。

在探索了before_process()之后，符号化执行最终将用户重定向到最终页面checkout_success.php（R4）。当符号化执行到达这个页面时，意味着结账过程已经完成，我们的分析产生了一个最终的漏洞报告。在这个支付模块的报告中，订单ID和货币仍然被玷污，表明这个模块容易受到两种类型的逻辑攻击。第一类攻击允许攻击者通过重放$_POST['LKMAC']的值，为一个订单付款，并避免未来订单的付款。$_GET['LKMAC'] 的付费订单。请注意，攻击者可以很容易地拦截$_POST['LKMAC'] 的值或$_GET['LKMAC']的任何已支付的订单，通过改变返回URL到她自己选择的R2中。第二类攻击允许攻击者通过向收银员支付$_SESSION['order']->info['total']所指示的正确金额，但以不同的货币支付，从而少付一些。例如，商家可能以欧洲欧元列出产品价格，但攻击者可以用美元支付。[1]

为了说明对订单ID的第一类逻辑攻击，假设一个用户在商家的服务器上下了两个ID为1001和1002的订单。对于这两个订单，假定指定的收银员会生成一个秘密的MD5值，并且只有在向收银员支付了全额款项后才会访问带有该秘密值的页面checkout_process.php的URL。订单1001和1002的秘密值应该是不同的和不可预测的。这两个订单的URL如下所示。

```
URL1: http://merchant.com/checkout_process.php?
  orderID=1001&LKMAC=SecretMD5For1001

URL2: http://merchant.com/checkout_process.php?
  orderID=1002&LKMAC=SecretMD5For1002
```

检测到的逻辑漏洞的关键问题是，这个电子商务应用没有根据可信的订单ID和MD5值检查订单ID和LKMAC的请求参数值。假设攻击者已经为订单1001付款，并通过将返回URL（URL1）中的服务器从merchant.com改为attacker.com，截获了秘密值SecretMD5For1001。对于订单1002，攻击者可以跳过付款并直接跳到一个伪造的商家URL（如上图所示的URL1），而不是进行付款并被重定向到URL2。攻击者可以将订单1001的GET和POST请求参数值用于订单1002以避免付款。这种替换导致商家错误地认为订单1002已经被支付，而收银员实际上没有代表商家收到任何关于这个订单的信息。同样地，这个漏洞允许只要订单总额与1001号订单的总额一致，攻击者就可以绕过未来订单的付款。

[1]当收银员接受多种货币付款时，就可以发起这种攻击。

III. 办法

本节介绍了我们的高级方法。我们首先定义电子商务应用中的逻辑漏洞，提出我们的假设，然后描述我们方法的核心算法。

A. 定义

定义1（商家）。当用户通过第三方收银机正确支付订单时，商家就会接受该订单。商家是电子商务应用的核心角色，在结账过程中协调用户和收银员之间的沟通。商家负责初始化订单、跟踪支付状态、记录订单细节、最终确定订单并向用户运送产品（或提供服务）。

定义2（收银员）。第三方*收银员*代表商家接受用户的订单付款。当商家和用户缺乏相互信任时，收银员在他们之间架起了桥梁。用户信任收银员的私人信息，而商家则希望收银员能正确地向用户收费。

定义3（用户）。*用户*在商家的网站上发起一个结账过程，选择一个第三方收银台，向收银台付款，并从商家那里获得产品（或服务）。用户的输入和行动推动了结账过程的逻辑流程。有些用户是恶意的，因此商家需要防御不受信任的用户输入和行动。

定义4（电子商务应用中的逻辑流）。*电子商务应用中的逻辑流*是三个可能的当事方之间的通信：商家节点、收银员节点和用户。结账时的任何逻辑流都可能影响支付状态。请注意，根据其HTTP请求变量的运行时间值，一个商家网页可能被划分为多个商家节点。例如，一个页面可以执行 "插入"、"更新 "或 "删除 "操作，这取决于在$_GET['action']的值上。我们的分析从一个结账过程的开始商家节点n_0开始，到接受订单的目的地商家节点n_k结束。追踪支付状态的污点注释和跨逻辑流的签名令牌。假设对于结账过程中的任何有效节点n_i，我们以执行状态集Q_i开始对n_i的分析。在对n_i的分析结束时，我们会有执行状态集Q_j和下一个要访问的节点，即n_j。形式上，电子商务应用中的逻辑流可以表示为$\Pi = \{(n_i, Q_i) \rightarrow (n_j, Q_j) \mid 0 \leq i,j \leq k\}$。

定义5（逻辑状态）。一个*逻辑状态*由污点注释和与结账过程中其他有效节点的链接组成。逻辑状态的传播反映了支付状态的变化。具体来说，对于用户在集成了第三方收银机的商家网站上的任何订单，逻辑状态存储了以下支付状态组件的污点注释和暴露的签名令牌。

- ***订单ID***。订单的标识符，应在接受前支付。
- ***订单总额***。收银员应代表商家从用户那里收到的总金额。
- ***商户ID***。收银员用于销售产品或服务的商户的标识符。收银员将最终把收到的钱从用户那里转到商家那里。
- ***货币***。订单付款应使用的货币（货币系统）。
- ***暴露的签名令牌***。一个加密的值，在商家和收银员之间用秘密签署。它可以作为收银员的签名，当它在商家页面的文档对象模型（DOM）树中对用户可见时，被认为是暴露的。

定义6（电子商务应用程序的逻辑漏洞）。当对于任何被接受的订单ID，商家无法验证用户已经正确地将订单总额的预期货币支付给收银员的时候，*电子商务应用中*就存在*逻辑漏洞*。我们的定义是受收银员文档、开源电子商务应用和相关工作的广泛研究启发而制定的[30, 33]。当支付状态的完整性和真实性都得到保证时，支付就是安全的。篡改货币是我们在研究中发现的一个新的攻击媒介。

假设

*第三方出纳员是安全的。*我们将第三方收银机视为黑匣子，并假设它们是完全安全的。大多数第三方收银员的源代码是不可用的，但许多收银员都经过了严格的审核。第三方收银机的安全性与它在电子商务网络应用中的集成安全性是正交的。支付模块的开发者通常没有收银员的安全意识，因此支付模块通常更容易出现逻辑漏洞。

基于电子商务网络应用的逻辑漏洞，很容易对实时网站发起攻击。只需使用浏览器扩展，攻击者就可以扣留HTTP请求、修改请求或完全伪造请求。此外，攻击者可以利用已签署的令牌冒充收银员，重新使用以前订单的付款信息，或通过改变HTTP表单中的返回URL来拦截收银员的响应。

综上所述，电子商务应用中的逻辑漏洞是由以下五种污点注释造成的。

- ***被玷污的订单ID***。为了绕过订单付款，攻击者可以重放同一商家以前的订单的付款信息。只要未支付的订单总额和货币与之前支付的订单相符，未支付的订单就会被接受，因为订单ID没有被验证。
- ***污损的订单总额***。如果订单总额没有得到验证，攻击者可以通过篡改发送给第三方收银员的订单总额来为订单支付任意金额。向收银员支付部分款项仍然是必要的。
- ***污损的商家ID***。当商家ID被玷污时，攻击者可以在指定的收银台服务器上建立她自己的商家账户，而原来的商家ID就是在那里建立的。这使得攻击者可以将在商家网站上的订单的付款发送给自己而不是商家。请注意，对商家和收银员之间的秘密的检查可以取代对商家ID的检查，因为秘密是由商家设置的唯一的、可验证的值。
- ***污损的货币***。对于接受多种货币的收银员来说，有可能通过使用不同的货币为订单支付更少的费用，而不改变订单的总金额。
- ***暴露的签名令牌***。一个暴露的签名令牌会使任何针对可信符号值的安全检查失效。这是因为这样的签名请求可能是由攻击者伪造的，而不是来自受信任的出纳员。

B. 自动分析

第III-B1节介绍了我们的检测算法，该算法探讨了电子商务应用中三方（商家、收银员和用户）之间的关键逻辑流。第III-B2节描述了反映支付状态变化的污点操纵规则。

*1) 逻辑漏洞检测算法。*图3展示了我们的漏洞检测算法，该算法构成了我们方法的核心。它整合了商家节点的符号执行和污点分析，并连接各个节点以探索电子商务应用中的有效逻辑流。我们有四对可能的HTTP请求从客户端到服务器端。(用户，商家），（用户，收银员），（收银员，商家）和（商家，收银员）。攻击者可以跳过用户到收银台的请求，但他们需要发送相同数量的用户到商家的请求来执行结账过程中的所有必要步骤。因此，我们的算法依次分析属于结账导航路径的商家节点。

图3中有三个函数，第一个函数DETECTVULS是我们分析算法的主要函数。第二个函数ANALYZENODE单独分析每个商家节点，第三个函数GETNEXTNODE将节点连接在一起以获得有效的逻辑流。分析从起始节点n_s开始，有一个起始执行状态q_s。n_s和q_s都是从规范*Spec*中提取的。一个执行状态q包含一个逻辑状态，全局和局部变量的内存映射，别名信息等。我们的算法首先分析逻辑流$(user, \text{MERCHANT}(n_s))$，然后继续分析，直到所有有效的逻辑流都被探索出来。最后，对于最终执行状态集Q_f中的每个执行状态q_f，函数CHECKLOGICVULS检查q_f中的逻辑状态并报告任何检测到的逻辑漏洞$Vuls$。

函数ANALYZENODE递归地分析有效逻辑流的商人节点，直到达到最终节点n_f。只有当一个新的最终执行状态q_f具有唯一的新逻辑状态时，最终执行状态集Q_f才会被更新。这种更新策略背后的原因是，在一个执行中的其他数据

```
DETECTVULS(Spec)
 1  n_s ← GETSTARTNODE(Spec)
 2  q_s ← INITSTATE(Spec)
 3  q_s ← ADDCOMM(user, MERCHANT(n_s), q_s)
 4  Q_f ← ANALYZENODE(n_s, q_s, ∅, Spec)
 5  Vuls ← CHECKLOGICVULS(Q_f)
 6  返回 Vuls

ANALYZENODE(n, q, Q_f, Spec)
 1  n_f ← GETFINALNODE(Spec)
 2  如果 n=n_f
 3     那么 Q_f ← Q_f ∪ {q_f}
 4          返回 Q_f
 5  q ← PROPAGATENODESTATE(n, q)
 6  Q ← SYMBOLICEXECUTION(n, q, Spec)
 7  对于 Q 中的每个 q_i
 8  do n_i, q_i ← GETNEXTNODE(q_i, Spec)
 9     Q_f ← ANALYZENODE(n_i, q_i, Q_f, Spec)
10  返回 Q_f

GETNEXTNODE(q, Spec)
 1  n ← RESETREDIRECTION(q)
 2  如果 n = null
 3     则 n ← RESETFORMACTION(q)
 4  如果 ISCASHIER(n, Spec)
 5     那么 q ← ADDCOMM(user, CASHIER(n), q)
 6          n ← RESETCALLBACKURL(q)
 7          如果 n = null
 8             那么 n ← RESETRETURNURL(q)
 9  q ← ADDCOMM(user, MERCHANT(n), q)
10  返回 n, q
```

图3：漏洞检测的算法。

状态对最终的漏洞结果没有影响。函数PROPAGATENODESTATE将一个执行状态q从之前的节点（n_p）传播到当前的商家节点（n），对q进行一些操作。具体来说，这个函数更新运行时常量，如`$_SERVER['PHP_SELF']`，更新数组`$_GET`基于节点n的查询字符串，更新数组`$_POST`基于节点n的形式元素$_p$，并重置局部变量的内存映射。默认情况下，请求变量具有象征性的顶值，它代表所有可能的值，包括null。接下来，商家节点n通过函数SYMBOLICEXECUTION被符号化执行，Q是n的结束执行状态集。在符号化执行过程中，HTML表单动作URL、表单元素和商家到收银台的cURL[2]请求的参数值被监控，以寻找到其他商家节点或收银台节点的链接。

为了连接有效逻辑流的节点，函数GETNEXTNODE检查并重置四种类型的链接：重定向URL、表单动作URL、回调URL和返回URL。重定向URL或表单动作URL可以指向收银员节点或商家节点，而回调URL或返回URL只能指向商家节点。为了只沿着有效的逻辑流进行导航，我们放弃了那些形成反向或自循环逻辑流，以及与结账无关的URL。在函数GETNEXTNODE中的每个重置函数都在n中存储了特定类型URL的提取值，并将URL重置为空。首先检查头重定向URL（通过函数RESETREDIRECTION获得）和表单动作URL（通过函数RESETFORMACTION获得）的值。当URL n指向一个收银员节点时，会添加一个逻辑流（用户，CASHIER(n)）。为了模拟收银员的响应，该函数检查回调URL（通过函数RESETCALLBACKURL获得）和返回URL（通过函数RESETRETURNURL获得）的值。注意，回调URL和返回URL只能在出纳员被访问后设置。回调URL是可选的，可以由收银员首先访问，以通知其商家完成支付交易。返回URL是必需的，用户必须转发收银员对这个URL的响应，以确认商家服务器上的付费订单。函数GETNEXTNODE的返回值是一对接下来应该被访问的商家节点n和更新的状态q。

2) 污点规则。为了跟踪支付状态的完整性和真实性，我们设计了一些污点操作规则。支付状态的完整性可以挫败HTTP参数的篡改攻击，而支付状态的真实性可以抵御伪造的支付状态，即用可预测的或暴露的请求变量的值来伪造。更具体地说，未受污染的订单ID、订单总额、商家ID和货币确保了完整性，而没有暴露的签名令牌确保了真实性。污点规则的基本假设是。1）来自用户的请求是不被信任的。
2）通过不安全渠道发送的未签名的收银请求是不被信任的；3）用户通过HTTP重定向（状态代码302）转发给商家的收银响应也是不被信任的。最初，订单ID、订单总额、商家ID和货币都是有污点的。

当商家正确地验证了一个支付状态组件时，被检查组件的污点注释应该被移除。我们的方法对以下三种情况使用污点清除规则。

- ***条件性检查***。当一个（不）平等的条件检查针对一个支付状态组件的可信符号值验证一个不可信的值时，从被检查的支付状态组件中去除污点。
- ***写入商户数据库***。当一个有污点的值通过INSERT或UPDATE查询被写入商家的数据库时，保守地从组件中删除污点。在商家员工为一个订单运送产品或提供服务之前，她需要审查从数据库表中检索的订单细节。如果一个修改过的组件被写入数据库，商家员工可以很容易地发现修改过的组件，从而拒绝带有修改过的组件的订单。
- ***安全的通信渠道***。对于同步的商家到收银台的cURL请求，当cURL请求参数中包含订单总数、商家ID或货币时，要去除这些污点，并且无条件地去除订单ID的污点。同步请求是通过安全通信渠道发送的。

[2]http://curl.haxx.se/

因此，可以保证通过这种渠道的支付状态变化的真实性。

我们的方法有一个污点添加规则。当收银员到商户请求的条件检查依赖于暴露的签名令牌，在暴露的签名令牌上添加污点。token。我们跟踪所有在DOM树中披露给用户的签名令牌值（通常在隐藏的HTTP表单元素中）。尽管隐藏的HTTP表单元素在HTML页面的表现层中是不可见的，但攻击者只需查看网页的源代码就可以获得它们的值。请注意，并不是所有暴露的签名令牌都是污点。污点添加规则只适用于当暴露的签名令牌被用作收银员到商家请求的条件检查中的不可预测的值。一旦签名的令牌被暴露，它就不再是不可预测的了，因此不应该被用于条件检查。例如，假设我们在一个隐藏的HTML表格中拥有一个符号值为`md5($secret.$orderId.$orderTotal)`的签名令牌。如果我们的分析遇到平等检查`$_GET['hash'] == md5($secret.$_GET['oId'].$_GET['oTotal'])`，它会给暴露的签名令牌增加污点。这是因为

尽管`$secret`是不可预测的，但三个请求变量的值是可预测的。为了通过检查，攻击者可以在`$_GET['hash']`中使用暴露的签名令牌，在`$_GET['oId']`和`$_GET['oTotal']`中分别使用与暴露的签名令牌相关的订单ID和订单总数。

IV. 执行情况

我们开发了一个符号执行框架，将污点分析与PHP结合起来，PHP是构建Web应用最普遍的语言之一。我们扩展了用OCaml编写的静态字符串分析器[23, 27, 32]的PHP词法和解析器。我们的工具可以处理PHP的面向对象特性，包括类、对象和方法调用。我们为内置的PHP库函数编写了转移函数，其中包括字符串函数、数据库函数、I/O函数等。我们的工具咨询了Satisfiability Modulo Theories（SMT）求解器Z3[13]以获得分支的可行性，支持算术约束、简单的字符串约束和一些其他类型的约束。我们的实现共包含25，113行OCaml代码。虽然我们的实现以PHP语言为目标，但高层方法是通用的，适用于用其他语言编写的电子商务软件。

图4显示了我们框架的结构。给定一个电子商务网络应用程序的源代码和它的规范，我们的分析从商家节点n_s，即结账过程的第一个节点的单一执行状态q_s开始。对于每个商家节点n_i，我们的PHP词典和解析器将相应的商家页面转化为抽象语法树AST_i，然后由我们的IR构造器将其转化为内部表示IR_i。在符号执行引擎探索了IR的所有可能的控制流路径后$_i$，我们有一组终端执行状态Q_i。接下来，导航器搜索有效的逻辑流，并继续对新的商户节点进行符号执行，直到到达最终的商户节点n_f。最后，逻辑分析器检查最终执行状态集Q_f的所有唯一逻辑状态，然后报告任何检测到的逻辑漏洞。

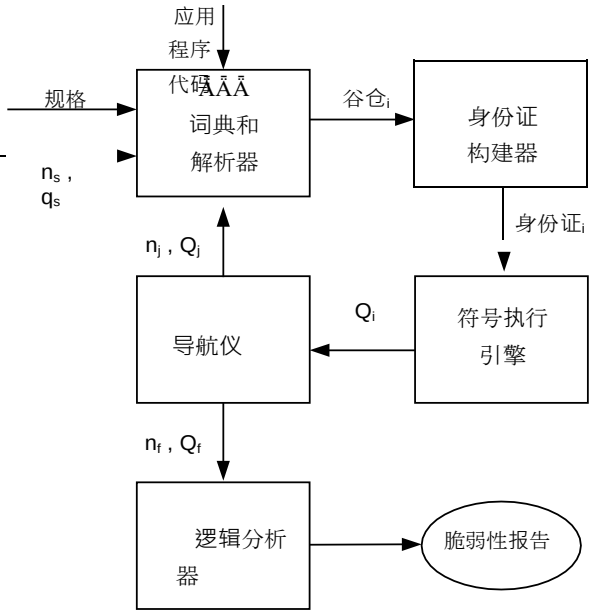

图4：符号执行框架。

为了指导我们的自动分析，我们需要开发者指定支付状态组件、结账过程中的关键商家页面、收银员URL、回调URL、返回URL、数据库中定义的可配置常量以及用于解决动态文件包含和类构建的几个变量的运行时值等应用特定变量名称。例如，一个支付类可以根据用户对支付方式的选择来动态构建。如果用户选择PayPal Standard作为支付方式，我们可以指定运行时变量`$_SESSION['payment']`的值为"Paypal standard"，以精确解决类`$payment`的目标。

A. 符号化执行

对于电子商务应用的结账过程中的每个商家页面，我们的PHP词典和解析器将其源代码转化为IR。我们遵循PHP语言参考，仔细编写解析规则以解决减少/还原冲突，分配运算符优先级以解决移位/还原冲突，并使用关联性来解决其他类型的冲突。我们观察到，一个PHP页面可以通过PHP include或`iframe`静态或动态地包含其他页面，而被包含的页面又可以反过来包含其他页面。为了充分扩展一个PHP页面，我们的分析器在可能的情况下推断出所包含页面的静态目标，而在目标只能在运行时决定的情况下，则采用规范。例如，静态include `require(DIRS_CLASSES.'cart.php')`取决于常数`DIRS_CLASSES`的值，而动态include `require($language.'.php')`则取决于运行时变量`$language`。

对于堆建模，我们的工具使用了五个变量映射：一个变量到符号值的内存映射，一个实例到类名的映射，一个别名到变量的映射，一个数组-父-数组-元素的映射和一个对象-父-对象-属性的映射。

地图。首先，变量到符号值映射允许我们对堆进行符号化建模。符号值是一个递归的数据结构，由以下类型组成：字面意思、基本符号PHP变量、库函数调用、两个符号值的连接、算术表达式、比较表达式和符号PHP资源。例如，符号值md5("hello".$_GET['orderID'])表示对库函数md5的调用，其符号参数是两个符号值的连接：一个字符串字面"hello"和一个整数类型的基本符号变量$orderID。第二，给定一个类的实例和一个方法名，实例到类名的映射使我们能够快速检索到相应的类方法定义。第三，别名到变量的映射使我们能够正确地更新符号堆。当以下情况时，别名就会被创建：一个方法从对象上下文中被调用（$this变得可用）；一个变量通过引用被分配；一个函数/方法有通过引用的参数或返回一个引用。最后，两个用于数组和对象变量的映射使我们能够分别跟踪数组和对象的子变量。我们的工具为全局变量使用一个内存映射，为局部变量使用一个内存映射。

为了在PHP中建立数组和对象的模型，我们采用McCarthy规则来进行列表操作[13]。给定一个数组a，一个数组元素e和数组索引i，让$a[i]$代表一个数组选择，$a\{i \leftarrow e\}$代表一个数组存储，索引i的元素设置为e。

$$(\forall \text{阵列} a)(\forall \text{元素} e)(\forall \text{索引} i,j)\ i = j \longrightarrow a\{i \leftarrow e\}[j] = e$$
$$\wedge\ i \neq j \longrightarrow a\{i \leftarrow e\}[j] = a[j]。$$

我们的实现尽可能精确地检索和更新数组元素（或对象属性）。否则，当一个数组变量的索引（或一个对象属性的字段）是⊤，数组元素（或对象属性）的所有可能的值都被合并。例如，假设我们有一个简单的数组$arr=array(1=>"x",2=>"y")。如果数组索引$i的值是⊤，则$arr[$i]的值是"x"或"y"。我们也使用麦卡锡规则来象征性地表示数组和对象。作为一个例子，$arr的符号表示是。

$$array\_update(array\_update(array(), 1, \text{"}x\text{"}), 2, \text{"}y\text{"})$$

B. *路径探索*

给定一个起始执行状态，我们的目标是探索商家节点的控制流图（CFG）中所有可能的程序内和程序间的边。我们使用基于工作列表的算法，用深度优先的策略来探索CFG的边。一方面，为了探索一个函数/方法体中所有可能的控制流，工作列表存储了尚未探索的可行分支的执行状态。每个执行状态包括一个程序计数器（由基本块编号和基本块内的语句编号组成）、一个逻辑状态、路径条件、全局和局部变量的内存图等。我们为一个工作列表中类似执行状态的最大数量设置了一个可配置的配额，以避免状态爆炸。当一个工作列表的配额用完后，我们只在一个执行状态有新的程序计数器或新的逻辑状态时才把它添加到工作列表中。在另一方面，为了探索所有可能的程序间边缘，我们的方法采用了一个全局调用栈，在函数调用前存储先前的函数环境快照。一个函数环境快照包括即将被探索的程序间函数调用的参数-
参数图，当前函数的工作列表和结束执行状态等。

我们的工具咨询了SMT求解器Z3来解决约束问题。当在符号执行过程中遇到条件时，我们的分析器将条件转化为smtlib2格式的公式，将新的公式与当前的路径条件连接起来，并将合并后的路径条件反馈给Z3以获得答案。当两个分支都可行时，我们选择一个分支首先进行探索，并将另一个分支添加到当前工作列表中。我们的约束条件支持以下类型：布尔值、整数、实数、字符串、数组、对象、资源、空和⊤.我们试图推断简单的字符串约束的可满足性，它可以包含字面意义、字符串变量和操作符，如=、≠、<、≤、>和≥。为了用符号表示PHP库的函数调用，我们使用Z3中的define-fun来表示函数声明。

```
$error = false。
如果 ($_POST['x_response_code'] =='1')  {
  如果(tep_not_null(AUTHORIZENET_MD5_HASH) &&
    ($_POST['x_MD5_Hash'] != strtoupper(
      md5(AUTHORIZENET_MD5_HASH .授权网络_login_id 。
      $_POST['x_trans_id'] .
      $this->format_raw($order->info['total'])
    ))){
    $error = '验证'。
  } elseif ($_POST['x_amount'] !=
      $this->format_raw($order->info['total'])) {
    $error = '验证'。
  }
} elseif ($_POST['x_response_code'] == '2') {
  $error = 'declined';
} else {
  $error = '一般'。
}

如果($error != false) {
  tep_redirect(tep_href_link(
    filename_checkout_payment。
    'payment_error=' . $this->code
    .'&error=' . $error, 'SSL', true,
    false))。
}
```

图5：路径探索的例子。

考虑图5中路径探索的例子。由于请求变量的默认值是⊤，所有可能的控制流边都被探索了。例子中只有一条探索路径通向有效的逻辑流，而其他路径将用户重定向到一个支付节点，错误信息在$error。对于有效路径上的第二个if条件，有一个方法调用$this->format_raw($order->info['total'])。为了跟踪这个程序间的边缘，我们的分析器首先查找对象$this的类名，然后查找相应类中format_raw方法的定义。接下来，分析器更新了包括$this在内的逐一引用参数的别名，根据方法调用的参数初始化参数值，传递全局变量的内存图，并推送了当前的快照

函数环境进入全局调用栈。在format_raw 的符号执行结束时，我们有一组执行状态*Q*。在方法调用返回后，我们的分析器从全局调用栈中弹出函数环境，并将*Q*映射到*Q*，以更新具有逐一引用参数的方法参数。为了在调用后继续进行路径探索，*Q*被添加到当前工作列表中。当所有可能的路径都被探索过后，商家的ID和订单总数在有效流程的执行状态中是不被污染的，这使得$error的值保持不变。

C. 逻辑流程

我们分析的重点是一个成功的结账过程的关键逻辑流。我们抛弃了后退流、错误流或中止流，因为它们与我们的安全分析无关。首先，当一个商家节点***n***发生错误时，就会出现*后退流*，用户被重定向到之前的商家节点或相同的商家节点***n***。第二，*错误流*指的是重定向到一个特殊的错误页面或一个在请求变量中带有错误信息的访问页面。在第一种情况下，特殊错误页面不属于关键的结账过程，流向这个页面的流量被丢弃。在第二种情况下，流向具有象征性错误信息变量的页面是后向流，会被自动丢弃。最后，当一个严重的错误发生时，一个被中止*的流*就会发生，商家页面的渲染被一个退出语句所停止。

在搜索其他节点的链接时，我们的分析器会解析HTTP表单和cURL参数的符号值。由于字符串字面意义不是符号值可以代表的唯一类型，我们不能简单地使用正则表达式，如<form s*action s*=s*[^>]*>来提取链接。因此，我们的分析器递归地检查符号值的每一个组成部分，以正确处理非符号值。在大多数情况下，商家在向收银员发出的HTTP请求中嵌入了URL，我们的分析器可以找到这样的URL。然而，商家也可能在收银员的服务器上存储回调URL和返回URL的配置。对于这种情况，我们需要指定预先配置好的商家URL，以便在用户向收银台发出请求后继续探索逻辑流。

来自收银员的请求通常在其参数中存储关键的支付状态。尽管不同收银员的请求参数名称不同，但除非其值被写入数据库，否则没有必要将其名称与支付状态组件联系起来。一方面，当一个不受信任的请求参数与一个受信任的支付状态组件进行比较时，我们的工具可以推断出一个请求参数与哪个支付状态组件相关联，并对涉及的支付状态组件应用污点规则。例如，对于$_POST['x_amount']==$order->info['total']，我们的分析器就会根据可信的支付状态组件$order->info['total']而不是不可信的支付状态组件$order->info['total']来清除订单总额的污点。$_POST['x_amount']。另一方面，当来自收银员的不可信任的请求变量通过INSERT或UPDATE查询被写入数据库时，我们需要规范请求变量与哪些支付状态组件相关。例如，假设一个规范将$_GET['v1']带订单ID，$_GET['v2']带订单总数。如果这两个请求参数被写入商家的数据库，它们将被从数据库中读取并清楚地显示给商家的员工。由于她需要在接受订单之前审查订单细节，她可能会拒绝任何付款状态异常的订单。因此，应该根据$_GET['v1']和$_GET['v2']的规范，删除订单ID和订单总数的污点注释。

表一:收银员的支付模块。

收银员	模块	独特的	回调
2Checkout	1	1	N
授权网（Authorize.net	2	2	N
计时支付	1	1	Y
缴费	1	1	Y
iPayment	3	1	Y
罗托昆塔(Luottokunta)	2	2	N
Moneybookers	23	1	Y
NOCHEX	1	1	N
贝宝	5	5	Y
PayPoint.net	1	1	N
呼叫中心	1	1	N
RBS WorldPay	1	1	Y
赛捷支付	3	3	Y
验证信息	1	1	Y
总数	46	22	8

V. 实证评价

为了评估我们工具的有效性和性能，我们在osCommerce[1]上进行了实验，它是最流行的开源电子商务应用程序之一。它有13年的悠久历史，为超过14000个注册网站提供动力[1]。osCommerce的最新稳定版本（2.3版）包含987个文件，有38991行的PHP代码。它通过不同的支付模块支持各种第三方收银员和多种货币，这些模块作为附加组件被集成在主框架中。每个支付模块都提供一种支付方式，用户可以在结账时选择。

我们总共评估了46个支付模块，其中22个有不同的CFG。osCommerce有928个支付模块，而且自2003年以来一直在积极添加新的支付模块。此外，支付模块也随着时间的推移而不断发展。例如，模块Luottokunta（1.2版）被报告为有漏洞[9]，Luottokunta（1.3版）被发布以修补报告的漏洞。osCommerce中默认包含46个支付模块，其中44个模块是为了整合第三方收银机而开发的。剩下的两个支付模块与我们的安全分析无关。一个允许商户接受货到付款，另一个使商户接受邮寄的汇票。接受在线支付的44个支付模块有20个独特的CFG。在变量名称和收银员URL方面彼此略有不同的模块可能有相同的CFG。因此，我们评估了20个具有独特CFG的默认支付模块，以及两个Luottokunta支付模块。所有的实验都是在一台具有四核CPU（2.40GHz）和4GB内存的台式电脑上运行。

TABLE II: Logic Vulnerability Analysis Results.

支付模块	污损/暴露的					安全
	订单编号	订单总数	**MerchantId**	货币	已签名的令牌	
2Checkout						〃
授权网络信用卡AIM						
Authorize.net信用卡SIM卡						〃
计时支付						〃
缴费						
iPayment (Credit Card)						〃
罗托昆塔 (v1.2)						〃
罗托昆塔 (v1.3)						〃
Moneybookers						
NOCHEX						〃
PayPal Express						
PayPal Pro - 直接支付						
PayPal Pro (Payflow) - 直接支付						
PayPal Pro (Payflow) - Express Checkout						
PayPal标准						〃
PayPoint.net SECPay						〃
呼叫中心						〃
苏格兰皇家银行WorldPay托管						〃
赛捷直接支付						
赛格支付表						〃
赛捷支付服务器						✓*
共计						9761129+
1*						

表一显示了来自14个不同收银员的支付模块。模块"一栏显示了一个收银员拥有的支付模块的数量，"独特"一栏列出了具有独特CFG的支付模块的数量。所有的支付模块都是最新的版本，除了Luottokunta，我们为它包括了两个具有不同CFG的版本。Cashier Moneybookers为不同的国家和货币提供了23个支付模块，但我们观察到，所有这些模块都共享相同的CFG。因此，只挑选一个Moneybookers模块进行安全分析就足够了。相比之下，PayPal有5个支付模块，每个模块都有一个独特的CFG。

A. 分析结果

表二显示了对22个独特支付模块的分析结果。在"污损/暴露"一栏中，显示了每个模块中存在的支付状态的污损成分和暴露的签名令牌。在这些列中，标有""的表格单元格意味着一个支付状态组件是有污点的，或者一个签名令牌是暴露的。最后一列"安全"总结了一个支付模块的安全性。当一个支付模块验证了支付状态的所有组件，并且没有暴露任何签名令牌时，它被认为是安全的，并标记为"✓"；否则，它被标记为〃"。

表二显示，当一个支付模块不安全时，它往往容易受到支付状态的不同组成部分的几种逻辑攻击。首先，9个模块未能正确验证订单ID。这使得攻击者可以为一个订单支付一次，并重复使用已支付订单的支付状态值来绕过为未来的订单付款。第二，7个模块未能验证订单总额，允许攻击者支付任意金额。第三，6个模块未能验证商家的ID，允许攻击者代替自己付款。请注意，对秘密的验证可以取代对商家ID的验证。第四，11个模块未能验证货币，使其成为支付状态中最被忽视的部分。当收银员被配置为只接受商家的一种货币时，不验证货币是安全和可以接受的。然而，我们认为，最好的做法是始终验证货币，以便将来可以很容易地添加额外的货币。最后，2个已签署的令牌意外地以纯文本形式暴露出来，使攻击者可以冒充收银员。我们在评估中也跟踪了暴露的秘密。当秘密被暴露时，攻击者可以任意伪造订单ID、订单总额、商家ID和货币的值。幸运的是，没有一个模块犯这样的错误。

总之，如表二最后一栏所示，22个模块中的9个是安全的；当只接受一种货币时，模块Sofortberweisung Direkt是安全的；其余12个模块是脆弱的。我们起初预计Luottokunta（v1.3）的补丁版本是安全的，但惊讶地发现它仍然是脆弱的。这表明编写一个完美安全的支付模块的难度。我们在osCommerce的本地部署中手动确认了检测到的漏洞，在由osCommerce驱动的实时网店上成功进行了负责任的实验，并与osCommerce的开发者就检测到的漏洞进行了沟通。我们将检测到的逻辑漏洞分为以下几类。

*1) 不受信任的请求变量。*支付模块的开发者有时会犯这样的错误：根据不可信任的请求变量来检查支付状态。验证不受信任的请求变量既不能保证支付状态的完整性，也不能保证支付状态的真实性，但可能会给开发人员一种错误的安全感。四个模块，即Authorize.net信用卡AIM、iPayment（信用卡）、Luottokunta（v1.3）和PayPoint.net SECPay都属于这个类别。通过这种不充分检查的不可信任的请求变量的值可能与实际支付状态组件不一致。例如，模块iPayment（信用卡）在以下代码中根据不受信任的请求变量`$_GET['ret_booknr']`对订单ID进行了检查。

```
$_GET['ret_param_checksum'] !=
md5(MODULE_PAYMENT_IPAYMENT_CC_USER_ID)
  .($this->format_raw($order->info['total'] )
  * 100) .$currency
  .$_GET['ret_authcode'] .$_GET['ret_booknr'] 。
  .ipayment_cc_secret_hash_password)
```

攻击者可以为一个订单支付一次，并通过修改前面商家到收银台请求的返回URL，拦截已支付订单的收银台到商家的请求。在上述例子中，攻击者需要记录以下数值`$_GET['ret_param_checksum']`, `$_GET['ret_authcode']`和`$_GET['ret_booknr']`。对于未来的订单，只要订单总额和货币与已支付的订单相同，攻击者可以购买不同的产品并绕过付款。请注意，`$_GET['ret_param_checksum']`应该是一个不可预测的、独特的、用秘密`IPAYMENT_CC_SECRET_HASH_PASSWORD`签名的值。然而，简单地重新播放三个GET变量的拦截值就可以让攻击者通过上述支付状态检查。例子中的检查是不充分的，因为条件中的订单ID值来自不受信任的`$_GET['ret_booknr']`。

*2) 暴露的签名令牌。*暴露的签名令牌使支付状态的验证无效。ChronoPay和RBS WorldPay Hosted两个模块暴露了他们的签名令牌。基于暴露的签名令牌的验证不能确保支付状态的真实性。攻击者可以记录隐藏在HTML表格中的签名令牌的值，并伪造一个请求，以伪造一个已完成的付款。例如，以下来自表单元素`M_hash`的暴露的签名令牌，使对订单ID、订单总额和商家ID（秘密的`RBSWORLDPAY_HOSTED_MD5_PASSWORD`也可以唯一地识别一个商家）的验证无效。

```
tep_draw_hidden_field('M_hash',
  md5(tep_session_id() .$customer_id
    .$order_id . $language
    . number_format($order->info['total'], 2)
    .rbsworldpay_hosted_md5_password) ) 。
```

从根本上说，暴露的签名令牌是由商家签名和收银员签名使用相同的秘密造成的。我们观察到，当商家希望用已签名的令牌向收银员验证自己的身份时，已签名的令牌往往会被暴露。一个已签名的令牌既可以作为商家签名，也可以作为非cURL HTTP请求的收银员签名。当一个签名的令牌被用于这两个目的时，如果攻击者可以拦截收银员到商家的请求，那么它就被认为是暴露的。有有两种方法可以解决这个问题。第一种是只使用一个秘密，但有两种计算方式，以使签署的令牌不同。例如，通过简单地改变计算中支付状态的组成部分的顺序，我们可以用同一个秘密产生不同的签名令牌。一个更好的方法是使用两个秘密来避免暴露重要的签名令牌。我们可以使用一个秘密来验证商家，另一个秘密来验证使用相同计算的收银员，而不必担心签名令牌的安全性。

*3) 不完整的支付验证。*支付模块有时只对支付状态的组成部分进行部分核查。换句话说，对支付状态的某些组成部分的检查是缺失的，而不是不充分的。三个模块，即Sage Pay Form、Überweisung Direkt和PayPal Standard都属于这个类别。Sage Pay Form模块将部分付款状态写入数据库，但缺少对订单总额和货币的检查。Überweisung Direkt模块没有验证货币，因此，如果出纳员被配置为支持多种货币，就容易受到货币篡改攻击。模块PayPal Standard错过了对商家ID的检查，使攻击者可以自己付款。

*4) 缺少支付验证。*一些支付模块的设计并不是为了防御逻辑攻击，根本没有对支付状态的安全检查。它们很容易成为攻击者的游乐场。以下五个支付模块不幸地属于这个类别。ChronoPay, Luottokunta (v1.2), NOCHEX, 2Checkout 和 PSiGate。此类支付模块应尽快打上补丁。

B. 在真实网站上的实验

为了展示基于表二所列检测到的逻辑漏洞的攻击的可行性和简易性，我们以负责任的态度在三个现场网站上进行了实验。我们咨询了我们大学的律师，并按照Wang等人的例子，设置了攻击者的匿名性，在超市用现金购买VISA礼品卡，并在第三方收银台注册账户[30]。没有浏览器扩展的谷歌Chrome浏览器足以作为我们的攻击工具。虽然我们最初没有或少付给商家三个订单的费用，但在收到图2所示的产品后，我们向商家全额支付了费用。我们将实验结果报告给了osCommerce的开发者。实验的细节将在下文中详述。

Canonical有限公司的Ubuntu在线商店。(RBS WorldPay Hosted)。RBS WorldPay是一个主要用于英国的收银系统，支持多种货币。Ubuntu在线商店是一个有特色的osCommerce商店，它使用脆弱的模块RBS WorldPay Hosted。如表二所示，这个支付模块很容易受到货币攻击。我们用英磅下了一个订单，但用美元向收银员WorldPay支付了相同的金额。大约一周后，我们收到了一台Ubuntu笔记本（如图2所示），尽管我们一开始并没有支付全额款项。

一家婴儿用品网店(Authorize.net Credit Card SIM)。模块Authorize.net信用卡SIM容易受到订单ID攻击。在我们对婴儿用品网店的实验中，我们下了两个订单，订单总额相同，但

表三：性能结果。

支付模块	文件	节点(%)	边缘(%)	报告(%)	国家	流动性	时间 (s)
2Checkout	105	5,194 (19.09%)	6,176 (19.15%)	8,385 (25.01%)	40	4	16.04
授权网络信用卡AIM	105	5,274 (19.95%)	6,284 (19.96%)	8,545 (25.97%)	43	4	17.65
Authorize.net信用卡SIM卡	105	5,221 (19.66%)	6,221 (19.72%)	8,435 (25.52%)	46	4	16.89
计时支付	99	5,013 (15.67%)	5,969 (15.61%)	8,084 (20.75%)	69	5	31.51
缴费	100	5,118 (18.31%)	6,109 (18.42%)	8,408 (23.68%)	335	6	125.29
iPayment (Credit Card)	99	4,999 (16.09%)	5,932 (16.14%)	7,918 (21.62%)	38	5	21.86
罗托昆塔 (v1.2)	105	5,158 (18.94%)	6,127 (18.96%)	8,291 (24.72%)	34	4	15.33
罗托昆塔 (v1.3)	105	5,164 (18.99%)	6,135 (19.03%)	8,308 (24.80%)	35	4	15.33
Moneybookers	99	5,082 (15.90%)	6,059 (15.85%)	8,215 (21.08%)	66	4	80.85
NOCHEX	105	5,145 (18.90%)	6,111 (18.89%)	8,237 (24.67%)	33	4	15.03
PayPal Express	104	5,351 (12.63%)	6,379 (12.64%)	8,596 (17.95%)	62	11	42.15
PayPal Pro - 直接支付	105	5,302 (19.85%)	6,339 (19.77%)	8,700 (25.61%)	65	4	20.76
PayPal Pro (Payflow) - 直接支付	105	5,302 (19.92%)	6,339 (19.85%)	8,714 (25.71%)	63	4	20.85
PayPal Pro (Payflow) - Express Checkout	99	5,128 (14.41%)	6,107 (14.35%)	8,197 (20.08%)	31	10	31.95
PayPal标准	99	5,040 (16.03%)	6,006 (16.01%)	8,170 (21.04%)	68	6	33.01
PayPoint.net SECPay	105	5,174 (19.09%)	6,152 (19.10%)	8,332 (24.97%)	40	4	15.80
呼叫中心	106	5,231 (19.07%)	6,228 (19.04%)	8,436 (24.95%)	44	4	16.82
苏格兰皇家银行WorldPay托管	99	5,019 (15.84%)	5,977 (15.92%)	8,121 (21.09%)	79	5	36.12
赛捷直接支付	106	5,447 (20.71%)	6,515 (20.55%)	8,984 (25.97%)	95	4	26.20
赛格支付表	106	5,315 (19.52%)	6,329 (19.54%)	8,762 (24.55%)	55	4	19.96
赛捷支付服务器	101	5,100 (14.72%)	6,067 (14.62%)	8,268 (19.78%)	42	6	28.26
许可证制度	98	5,038 (16.01%)	6,003 (15.96%)	8,160 (21.20%)	97	5	43.86
平均值	102.73	5,173 (17.70%)	6,162 (17.69%)	8,376 (23.21%)	67.27	5.05	31.43

只为第一个订单付款。我们在服务器上设置了一个简单的网页来记录HTTP请求变量的值。对于第一笔订单，我们把返回URL的值从商家的URL改为我们网页的URL。这一改变让收银员Authorize.net将付款通知请求发送给我们而不是商家。我们重新播放了第一个订单的请求变量的记录值，用于第二个订单的收银员到商家的请求。我们一开始没有为第二笔订单支付任何费用，但收到了一个从加州运来的脏尿布游戏包。

一家巧克力网店（PayPal标准）。模块PayPal Standard容易受到商家ID的攻击。PayPal是美国最流行的收银机之一，但在这个支付模块中却没有安全地使用它。在我们对巧克力网店的实验中，我们只是将巧克力商家的PayPal账户的商家ID改为我们自己的PayPal账户，用于用户对收银员的支付请求。这样一来，虽然一开始没有向巧克力商户付款，但我们还是收到了三块巧克力。

C. 绩效评估

表三显示了我们在符号执行期间收集的一些数据，以证明我们工具的性能。从 "文件 "栏到 "状态 "栏显示了所有商家节点的平均数字，而 "流量 "和 "时间（秒）"栏显示了商家节点的总数量。对于每个商家节点的IR，我们报告解析的文件数量（列 "文件"）、节点数量和节点覆盖率（列 "节点（%）"）、边的数量和边的覆盖率（列 "边（%）"）以及语句数量和语句覆盖率（列 "Stmts（%）"）。此外，"状态 "一栏显示了终端执行状态的总数；"流量 "一栏显示了用户、收银员和商户之间的逻辑流量总数。结账；而 "时间（s）"一栏显示了每个支付模块的总分析时间（秒）。

商人节点的分析并不复杂。每个商家节点包括的文件数量从98到106不等，平均为102.73。一个IR平均有5173个基本块（节点），6162个控制流边和8376个语句。节点、边和语句的覆盖率是针对主函数、函数体和方法体计算的。一些定义的函数、定义的类方法，甚至主函数的一些分支在检查过程中可能根本没有被执行。平均来说，每个商家节点的符号执行的CFG节点覆盖率为17.70%，边覆盖率为17.69%，语句覆盖率为23.21%。

为了估计人工代码审查的工作量，我们还统计了与支付模块的结账过程有关的代码行数。一般来说，每一个支付模块的代码行数都略高于表三中列出的语句数。例如，对于PayPal Express模块，总共有8727行代码需要审查，而其IR有8596条语句。除了代码之外，人工审核人员还需要检查数据库表格和收银员的文件。

平均来说，在每个支付模块的5.05个逻辑流程中，探索67.72个执行状态需要31.43秒。在简单的情况下，只需要4个逻辑流程就可以启动结账过程，在收银员的服务器上进行支付，通知商家付款并完成订单。模块PayPal Express有最复杂的逻辑流。它使用11个逻辑流为每个支付交易获得一个`pe_token`，用函数`setExpressCheckout`启动快速结账，在PayPal服务器上付款，用函数`getExpressCheckoutDetails`获得付款人的详细资料，用函数`doExpressCheckoutPayment`完成销售。inpay模块有

表四：覆盖结果。

支付模块	主			函数语句	(%)类语句(%)
	节点(%)	边缘(%)	报告(%)		
2Checkout	498 (39.60%)	693 (28.86%)	1,246 (58.89%)	2,249 (17.65%)	4,891 (19.76%)
授权网络信用卡AIM	498 (40.20%)	693 (29.37%)	1,246 (59.94%)	2,249 (19.65%)	5,051 (20.40%)
Authorize.net信用卡SIM卡	498 (39.60%)	693 (28.86%)	1,246 (58.89%)	2,249 (18.45%)	4,941 (20.32%)
计时支付	463 (36.04%)	647 (26.24%)	1,130 (54.34%)	2,249 (14.64%)	4,705 (15.61%)
缴费	510 (39.70%)	709 (30.22%)	1,218 (56.17%)	2,276 (17.27%)	4,915 (18.60%)
iPayment (Credit Card)	454 (38.25%)	632 (27.90%)	1,116 (59.10%)	2,249 (16.10%)	4,554 (15.15%)
罗托昆塔 (v1.2)	498 (39.60%)	693 (28.86%)	1,246 (58.89%)	2,249 (17.30%)	4,797 (19.32%)
罗托昆塔 (v1.3)	498 (39.60%)	693 (28.86%)	1,246 (58.89%)	2,249 (17.34%)	4,814 (19.46%)
Moneybookers	471 (36.12%)	656 (26.63%)	1,139 (54.32%)	2,249 (14.52%)	4,828 (16.29%)
NOCHEX	498 (39.60%)	693 (28.86%)	1,246 (58.89%)	2,249 (17.30%)	4,743 (19.19%)
PayPal Express	575 (28.91%)	797 (21.32%)	1,324 (44.76%)	2,249 (11.75%)	5,024 (13.66%)
PayPal Pro - 直接支付	498 (40.20%)	693 (29.37%)	1,246 (59.94%)	2,249 (19.88%)	5,206 (19.87%)
PayPal Pro (Payflow) - 直接支付	498 (40.20%)	693 (29.37%)	1,246 (59.94%)	2,249 (19.81%)	5,220 (20.09%)
PayPal Pro (Payflow) - Express Checkout	508 (34.70%)	706 (25.71%)	1,201 (52.96%)	2,249 (13.12%)	4,747 (15.07%)
PayPal标准	477 (36.76%)	665 (27.08%)	1,151 (54.88%)	2,249 (15.09%)	4,770 (15.69%)
PayPoint.net SECPay	498 (39.60%)	693 (28.86%)	1,246 (58.89%)	2,249 (17.67%)	4,838 (19.64%)
呼叫中心	498 (40.20%)	693 (29.37%)	1,246 (59.98%)	2,249 (17.74%)	4,942 (19.39%)
苏格兰皇家银行WorldPay托管	461 (36.63%)	643 (27.02%)	1,132 (55.35%)	2,249 (14.97%)	4,740 (15.81%)
赛捷直接支付	498 (40.20%)	693 (29.37%)	1,246 (59.94%)	2,249 (20.08%)	5,490 (20.67%)
赛格支付表	498 (39.70%)	693 (29.00%)	1,246 (58.97%)	2,251 (17.55%)	5,266 (19.41%)
赛捷支付服务器	463 (36.07%)	645 (26.28%)	1,151 (55.42%)	2,249 (13.45%)	4,868 (14.27%)
许可证制度	470 (36.69%)	653 (26.94%)	1,136 (55.28%)	2,249 (15.41%)	4,776 (15.82%)
平均值	492 (38.10%)	685 (27.92%)	1,211 (57.03%)	2,250 (16.67%)	4,915 (17.89%)

实验结果显示，我们的分析时间最长（125.29秒），执行状态的数量也最多（335个状态）。性能结果表明，我们的自动检测比人工分析更有效、更全面。当我们手动确认检测到的逻辑漏洞时，我们对每个支付模块需要大约30分钟。我们花了大约15分钟来阅读商家页面和收银员文档的控制流程，另外15分钟来寻找导致逻辑攻击的有效输入。

我们采取了一些优化措施来加快分析速度，其中有两项大大减少了分析时间。第一个优化将工作列表中类似执行状态的最大数量设置为1。这意味着，每当分析在工作列表中存储一个新的执行状态时，它首先检查是否已经存在一个具有相同程序计数器和逻辑状态的执行状态。如果是，新的执行状态就会被丢弃。由于这样的两个执行状态通常只有微小的差别，丢弃第二个状态对漏洞分析结果没有影响。每个支付模块的分析时间被限制在10分钟之内。当我们把一个工作列表的长度增加到两个时，在分析完成之前就发生了超时事件。第二项优化将一些象征性的会话变量设置为不为空，就像它们在正常结账过程中应该有的样子。比如说。`$_SESSION['customer_id']` 和 `$_SESSION['cartId']` 被指定为非空。商家节点的IR中的前几个基本块经常检查一些会话变量是否为空。第二个优化是在符号执行过程的早期状态下排除无关的分支。考虑到状态的数量通常以指数级的速度增长，这加速了我们的分析。

表四显示了详细的覆盖结果。本表中的所有数字是所有商家节点的平均数字。在"主要"一栏中，显示了被分析的商家节点中主要函数的节点、边和语句的平均数和覆盖结果（列在括号中）。"Func Stmts (%)"和"Class Stmts (%)"两栏分别显示了被分析商家节点中定义的函数和类的平均数和覆盖结果（列在括号内）。

平均来说，我们的符号执行覆盖了492个主函数节点的38.10%，685条主函数边的27.92%和1211条主函数语句的57.03%。此外，它还覆盖了定义函数体中2250条语句的16.67%和定义类中4915条语句的17.89%。主函数是每个商家节点的入口，主函数的语句的平均覆盖率远远高于定义函数和类的覆盖率。从表四中可以看出，类的语句覆盖率的偏差是最大的。这是因为不同的支付模块作为插件被集成到结账过程中，具有动态的类构造，它们对主函数和定义函数的语句数影响不大。

我们的符号执行是为了安全分析而开发的，而不是为了实现高覆盖率。所有三类语句的平均覆盖率（23.21%，如表三所示）低于主函数语句的覆盖率（57.03%），但高于定义函数语句的覆盖率（16.67%）和类语句的覆盖率（17.89%）。覆盖率低的原因有三个。首先，并不是所有的定义函数和类方法的语句都在每个商家节点中使用。一个商家节点可能只需要由定义的函数和类所提供的少数功能。

方法。其次，我们的工具基于分支的可行性来探索CFG的控制流。请注意，一个商家页面往往根据不同的请求参数值被划分为多个商家节点。我们的探索是基于商家节点的，但覆盖率是用商家页面计算的。这就解释了为什么有些模块的覆盖率很低。例如，PayPal Express的回调页面，在靠近页面开始的地方有一个基于请求变量`$_GET['osC_Action']`值的开关语句。它有不同的分支来处理"cancel"、"callbackSet"、"retrieve"和默认动作。对于这个页面的一个商家节点，只采取一个开关分支。第三，我们对一些用户输入的规范有助于我们避免对一些不相关的控制流的探索。并非所有可能的用户输入组合都需要进行漏洞检测，因此我们的分析集中在与结账过程有关的用户输入上。

D. 讨论

我们的检测工具的实现既不健全也不完整。对于我们的工具检测到的所有逻辑漏洞，我们仔细检查并测试了每一个，以确认它们是真正的阳性。据我们所知，没有观察到假阳性。我们不能保证没有逻辑漏洞，因为在大型现实世界的电子商务应用中探索所有可能的逻辑流程是很困难的。我们希望我们的工具可以帮助开发者编写安全的支付模块，并提高他们的安全意识。

我们的静态分析仍然面临着不小的挑战，其中包括PHP的动态特性、约束解法和正则表达式。通常情况下，静态分析在处理动态语言特征（如动态包含、动态类、数组和对象构造）方面受到限制，而PHP的动态特征也对我们分析的可扩展性和精确性产生了最显著的影响。为了精确解决动态特征，对一些关键代码进行了规范。

我们目前的实现还不支持JavaScript分析。有可能一些指向商家节点或收银员节点的链接是由客户端的JavaScript代码生成的。我们在实验中没有遇到任何JavaScript链接，但我们的测试对象可能不代表其他电子商务应用。由于JavaScript语言的各种动态特性，检测JavaScript代码中的链接是一项困难的任务。例如，它的`eval`函数，在运行时执行作为字符串提供的语句，可以以许多不同的方式调用。对于大量使用JavaScript的电子商务应用，我们可能需要结合JavaScript分析来检测动态生成的关键URL。

自动分析会产生大量的工程工作，对于具有大量支付模块的电子商务软件来说，摊销后的开发成本可以保持在较低水平。符号执行允许系统性的探索，并且对收银员和用户的HTTP请求/响应建模特别有用，因为可以使用符号值（而不是具体值）。相比之下，手工代码审查容易出错，而且很难涵盖所有可能的攻击载体和重要的控制流（这可能解释了为什么许多严重的漏洞仍然存在）。支付模块的数量(osCommerce为928)和两个有漏洞的Luottokunta模块说明了检测遗漏/不充分的检查的难度。然而，对于只有几个支付模块的基本电子商务软件，手工代码审查可能是一个可行的替代方案。

在结账过程中，当我们到达最后的商家节点时，有可能存在多个执行状态，并具有独特的逻辑状态。对于有效的逻辑流来说，哪个逻辑状态应该被选中而不是另一个逻辑状态，并没有通用的标准，我们将逻辑状态的选择留给拥有最佳判断力的开发者。我们目前的工具包括所有的污点操作和逻辑状态下的流量作为参考，并使用基于我们观察的启发式方法对逻辑状态进行排序。应该挑选的逻辑状态往往是污点注释数量最少的状态，不包括暴露的签名标记。原因是，我们的符号执行可能会保守地探索一个在实践中不会采取的分支，而只有相反的分支包含对支付状态的检查。

VI. 相关的工作

电子商务应用中的逻辑漏洞。逻辑漏洞的独特性，加上其巨大的影响，近年来吸引了研究人员的注意。Wang等人[30]是第一个分析基于Cashier-as-a-Service的网络商店的逻辑漏洞的人。通过人工安全分析，他们发现了严重的逻辑漏洞，这些漏洞会导致商家的服务器和收银员的服务器之间出现不一致的支付状态。他们的后续工作InteGuard[33]为第三方网络服务的集成提供了动态保护，包括商家网站中收银服务的集成。与他们的工作相比，我们试图全面检查支付状态的各种攻击载体，并在电子商务应用部署之前自动检测逻辑漏洞。我们发现了一个新的货币攻击向量，它允许攻击者为自己的利益修改支付的货币，并设计了一个符号执行框架来系统地探索结账过程的关键逻辑流。

网络应用程序中的参数污染漏洞。另一个活跃的研究方向是网络应用中的HTTP参数污染（HPP）。它是各种漏洞的一个常见攻击载体，其中包括逻辑漏洞。WAPTEC[5]采用白盒方法，结合符号执行和动态分析来检测PHP应用程序中的参数篡改漏洞，而NoTamper[4]和PAPAS[2]采用基于黑盒的方法。NoTamper[4]检测服务器端验证的不足，即服务器未能在客户端复制验证。PAPAS[2]旨在基于黑盒扫描技术自动发现参数污染，以获得脆弱的参数。我们的方法也假设用户的输入是不可信任的。然而，与孤立地检查参数的参数污染检测不同，我们的方法通过连接和分析结账过程的逻辑流来检测电子商务应用中的逻辑漏洞。

网络应用程序中的其他逻辑漏洞。除了对电子商务应用的攻击外，逻辑漏洞还为其他攻击打开了大门，其中包括访问控制攻击、单点登录攻击和工作流程的违反。

网络应用程序。首先，访问控制漏洞将特权功能或资源暴露给未经认证的用户。Nemesis[12]根据指定的访问控制列表进行动态信息流跟踪，而静态方法分析源代码以检测未受保护的访问[14, 26, 27]。第二，Wang等人发现了新的单点登录攻击[31]，InteGuard向前迈进了一步[33]，使用基于代理的方法，检查一组推断的不变性，让商家安全地整合第三方网络服务。第三，为了检测正常工作流程的偏差，重要的是首先建立一个正确工作流程的良好准则。这样的准则可以由开发者指定[19]，从应该在服务器端复制的客户端验证中推断出[4, 17]，或者从动态分析中获得[3, 11, 15, 22]。挫败逻辑攻击的另一种方法是通过构建来保证安全。Swift[8]和Ripley[29]都旨在将一些计算卸载到客户端，同时确保现代网络应用的服务器和客户端之间逻辑状态的一致性。电子商务应用中的逻辑漏洞是网络应用中一般逻辑漏洞的一个重要子类型。专注于这个特定的领域，我们能够设计一个安全支付的不变量来检测特定应用的逻辑漏洞。

符号执行和污点分析。符号执行和污点分析是安全研究中广泛使用的两种技术。Schwartz等人[25]提供了一个关于动态污点分析和前向符号执行的高级观点。自King[21]的开创性工作以来，符号执行是一种强大的技术，可用于不同的语言和问题设置。对于传统程序，KLEE[6]能够自动生成测试，即使是复杂的程序也能实现高覆盖率。对于服务器端语言，Halfond等人[18]应用符号执行来精确识别Java企业版（JEE）框架中的接口，而Rubyx[7]通过符号执行Ruby-on-Rails网络应用来检测基于规范的安全漏洞。对于广泛用于网络应用的客户端语言JavaScript，Saxena等人[24]设计并实现了一个可以处理字符串约束的符号执行框架。Pixy[20]是一个为PHP应用程序建立的静态污点分析器，它通过基于污点源和污点汇的规范的污点分析来检测注入漏洞。我们的方法将符号执行和污点分析以一种新颖的方式结合起来，以检测潜在的对支付状态的逻辑攻击。

VII. 结论

商家应仔细核查支付状态的每个关键部分，以确保商家的服务器和收银员的服务器之间支付状态的一致性。本文提出了第一个自动检测电子商务网络应用中逻辑漏洞的静态方法。我们的主要观点是，对支付状态的安全检查必须验证订单ID、订单总额、商家ID和货币的完整性和真实性。我们的框架整合了符号执行和污点分析，以跟踪关键的逻辑状态，其中包括支付状态，跨结账节点。我们的工具探索了重要的逻辑流程，扩展到22个独特的真实世界的支付模块，并检测到11个未知的漏洞以及一个已知的漏洞。对于未来的工作，我们计划为我们的符号执行支持更多的路径探索策略，增加函数摘要以提高性能，并将我们的分析应用于更多的流行电子商务网络应用。

鸣谢

我们感谢匿名审稿人和我们的牧羊人Davide Balzarotti对本文早期版本的有益反馈。我们还感谢加州大学的Ellen Auriti、Michael Sweeney和Lynette Temple为我们的概念验证实验提供建议。这项研究得到了美国国家科学基金会0917392、1117603、1319187和1349528资助的部分支持。这里提供的信息不一定反映政府的立场或政策，不应推断为官方认可。

参考文献

[1] osCommerce Online Merchant. http://www.oscommerce.com/.

[2] M.Balduzzi, C. T. Gimenez, D. Balzarotti, and E. Kirda.自动发现网络应用中的参数污染漏洞。在*网络和分布式系统安全会议上*，2011年。

[3] D.Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna.基于网络的应用程序的多模块漏洞分析。在*计算机和通信安全会议上*，2007年。

[4] P.Bisht, T. Hinrichs, N. Skrupsky, R. Bobrowicz, and V.N. Venkatakrishnan.NoTamper:对网络应用中参数篡改机会的自动黑箱检测。In *Proceedings of Computer and Communi- cations Security*, 2010.

[5] Bisht, Prithvi and Hinrichs, Timothy and Skrupsky, Nazari and Venkatakrishnan, V. N. WAPTEC: Whitebox analysis of web applications for parameter tampering exploit construction.在*计算机和通信安全会议上*，2011年。

[6] C.Cadar, D. Dunbar, and D. Engler.KLEE：为复杂系统程序自动生成高覆盖率的测试。在*《操作系统设计与实现》会议上*，2008年。

[7] A.Chaudhuri and J. S. Foster.ruby-on-rails网络应用的符号安全分析。在*计算机和通信安全会议上*，2010年。

[8] S.Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng.通过自动分区的安全网络应用。在*操作系统原理研讨会论文集*，2007年。

[9] 常见漏洞和暴露。http://cve.mitre.org/cgi-bin/cvename.cgi?name= cve-2009-2039, 2009.

[10] 常见弱点列举。CWE-840业务逻辑错误。http://cwe.mitre.org/data/definitions/840.html。

[11] M.Cova, D. Balzarotti, V. Felmetsger, and G. Vigna。Swaddler:基于异常的网络应用程序状态违规检测方法。In *Proceedings of Recent Advances in Intrusion Detection*, 2007.

[12] M.Dalton, C. Kozyrakis, and N. Zeldovich.Nemesis:防止网络应用中的认证和访问控制漏洞。在*USENIX安全研讨会的论文集中*，2009年。

[13] L.De Moura and N. Bjørner.Z3：一个高效的SMT求解器。在*系统构建和分析的工具和算法的会议上*，2008年。

[14] A. Doupé, B. Boe, C. Kruegel, and G. Vigna. Fear the EAR:
发现和缓解重定向后的执行漏洞。在*计算机和通信安全会议上*，2011年。

[15] V.Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna.争取自动检测网络应用程序中的逻辑漏洞。在*USENIX安全研讨会论文集*，2010。

[16] J.格罗斯曼。http://www.whitehatsec.com/home/assets/WP bizlogic092407.pdf, 2007年，七个商业逻辑缺陷使你的网站处于风险之中。

[17] A.Guha, S. Krishnamurthi, and T. Jim.使用静态分析进行Ajax入侵检测。In *Proceedings of World Wide Web*, 2009.

[18] W.G. Halfond, S. Anand, and A. Orso.精确的界面识别以改善网络应用程序的测试和分析。在*软件测试和分析国际研讨会论文集*，2009年。

[19] S.Hallé, T. Ettema, C. Bunch, and T. Bultan.通过导航状态机的模型检查和运行时执行，消除网络应用中的导航错误。在*自动软件工程会议上*，2010年。

[20] N.Jovanovic, C. Kruegel, and E. Kirda.Pixy:用于检测网络应用程序漏洞的静态分析工具（短文）。在2006年*安全与隐私研讨会*的*会议记录*中。

[21] J.C. King.符号执行和程序测试.在*ACM通讯*, 1976.

[22] X.Li and Y. Xue.BLOCK：一种检测网络应用程序状态违规攻击的黑箱方法。*计算机安全应用年会论文集*，2011年。

[23] Y.Minamide.动态生成的网页的静态近似。In *Proceedings of World Wide Web*, 2005.

[24] P.Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song.一个用于JavaScript的符号执行框架。在*安全与隐私研讨会论文集*，2010年。

[25] E.J. Schwartz, T. Avgerinos, and D. Brumley.你想知道的所有关于动态污点分析和前向符号执行（但可能一直不敢问）。在2010年*安全与隐私研讨会论文集中*。

[26] S.Son, K. S. McKinley, and V. Shmatikov.Fix Me Up：修复网络应用中的访问控制错误。在*网络和分布式系统安全会议上*，2013年。

[27] F.Sun, L. Xu, and Z. Su.网络应用中访问控制漏洞的静态检测。在*USENIX安全研讨会论文集*，2011。

[28] 美国人口普查局。季度零售电子商务销售。http://www.census.gov/retail/mrts/www/data/pdf/ec current.pdf，2013年。

[29] K.Vikram, A. Prateek, and B. Livshits.Ripley:通过复制执行自动确保Web 2.0应用程序的安全。在*计算机和通信安全会议上*，2009年。

[30] R.Wang, S. Chen, X. Wang, and S. Qadeer.如何在网上免费购物--
基于收银员即服务的网络商店的安全分析。*在安全与隐私研讨会论文集*，2011年。

[31] R.Wang, S. Chen, and X. Wang.通过Facebook和Google把我签到你的账户上。对商业化部署的单点登录网络服务进行流量引导的安全研究。在*安全与隐私研讨会论文集*，2012。

[32] G. Wassermann和Z. Su.健全和精确地分析网络应用的注入漏洞。在*编程语言设计与实现会议上*，2007年。

[33] L.Xing, Y. Chen, X. Wang, and S. Chen.InteGuard:实现对第三方网络服务集成的自动保护。在*网络和分布式系统安全会议上*，2013年。