NoTamper:自动黑盒检测网络应用中的参数篡改机会

Prithvi Bisht
伊利诺伊大学芝加哥分校 美国
伊利诺伊州芝加哥,
pbisht@cs.uic.edu

Timothy Hinrichs
芝加哥大学 美国伊利诺伊州芝加哥市
tlh@uchicago.edu

纳扎里-斯克鲁普斯基
伊利诺伊大学芝加哥分校 美国
伊利诺伊州芝加哥,
nskroups@cs.uic.edu

Radoslaw Bobrowicz 伊利诺伊大学芝加哥分校 美国伊利诺伊州芝加哥,
rbobrowi@cs.uic.edu

V.N. Venkatakrishnan
伊利诺伊大学芝加哥分校 美国
伊利诺伊州芝加哥,
venkat@cs.uic.edu

ABSTRACT

网络应用程序在很大程度上依靠客户端计算来检查和验证由用户提供的表单输入（例如，"信用卡到期日必须有效"）。这通常是出于两个原因：减少服务器的负担和避免与服务器通信的延迟。然而，当服务器不能复制在客户端进行的验证时，它就有可能受到攻击。在本文中，我们提出了一种新的方法，通过黑箱分析，自动检测现有（遗留）网络应用中潜在的这种服务器端漏洞。我们讨论了NOTAMPER的设计和实现，它是一个实现这种方法的工具。NOTAMPER已经被用来在一些开源的网络应用程序和实时网站中发现一些以前未知的漏洞。

类别和主题描述符

D.4.6 [**安全和保护**]。验证；K.4.4[**电子商务**]。安全；K.6.5 [**安全和保护**]。未经许可的访问

一般条款

语言、安全、验证

关键词

参数篡改, 漏洞构建, 限制条件解决, 黑盒测试, 符号评估

1. 简介

交互式表单处理在当今的网络应用中无处不在。它对电子商务和银行网站至关重要，这些网站在很大程度上依赖网络表单进行计费和账户管理。最初，典型的表单处理只发生在网络应用的服务器端。然而，最近，随着网络应用程序的便利性，以及对其进行了改进。

通过在网页上使用JavaScript提供的便利，表单处理也在网络应用程序的客户端进行。使用客户端的JavaScript处理用户提供的输入到一个Web表单，消除了与服务器通信的延迟，因此为终端用户带来了更多的互动和响应体验。此外，客户端的表单处理减少了网络流量和服务器负载。

浏览器进行的表单处理主要涉及检查用户提供的输入是否有错误。例如，一个接受信用卡付款的电子商务应用要求信用卡的到期日是有效的（例如，是未来的一个日期，并且是有效的月/日组合）。一旦输入数据被验证，它就会作为HTTP请求的一部分被发送到服务器，而输入则作为请求的参数出现。

接受这种请求的服务器如果假定所提供的参数是有效的（例如，信用卡还没有过期），就有可能受到攻击。这种假设确实是由浏览器端JavaScript强制执行的；然而，恶意用户可以通过禁用JavaScript、改变代码本身或简单地用用户选择的任何参数值手工制作一个HTTP请求来规避客户端验证。有参数篡改漏洞的服务器会受到各种攻击（如启用未授权访问、SQL注入、跨站脚本）。

虽然已经有大量的工作来解决具体的服务器端输入验证问题，如SQL注入和跨站脚本，但参数篡改问题本身在研究文献中很少得到关注，尽管它很普遍。SWIFT[8]和Ripley[24]关注的是确保网络应用程序开发框架中数据完整性这一更广泛的问题。这些方法的目标是实现*新的*网络应用程序，使其有效地不受参数篡改攻击的影响。相比之下，本文的重点是检测已经部署的*现有*网络应用（或遗留应用）中的参数篡改漏洞。

我们的目标是开发一种方法和工具，可供测试专家、网站管理员或网络应用程序开发人员使用，以确定参数篡改的机会。具体来说，我们的目标是以黑箱方式确定一个给定的网站（即一个已部署的网络应用程序）是否容易受到参数篡改攻击，并产生一份潜在漏洞和引发这些漏洞的相关HTTP参数的报告。我们设想这个报告可以以多种方式使用：专业测试人员使用我们的工具生成的输入来开发和演示具体的漏洞；网络应用程序开发人员使用我们的工具生成的输入来开发和演示具体的陷阱。

CCS'10，2010年10月4-8日，美国伊利诺伊州芝加哥。Copyright 2010 ACM 978-1-4503-0244-9/10/10 ...$10.00。

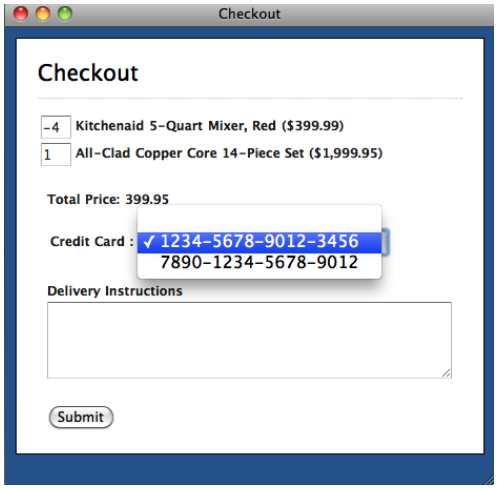

图1： 一个购物应用程序的运行实例

最后，网站管理员使用该报告来估计他们的网站受到攻击的可能性，并提醒有关的开发人员。

贡献摘要。

- 我们开发了第一个系统的方法来检测网络应用中的参数篡改机会。我们将我们的方法落实到一个工具中，我们称之为NOTAMPER。我们的方法在技术上取得了以下进展。
  - 客户端JavaScript代码分析技术，专门针对表单验证代码。
  - 应对黑盒漏洞分析的诸多挑战的输入生成技术。
  - 新颖的启发式方法，生成并优先处理可能导致漏洞的输入。
- 我们通过报告8个开源应用程序和5个在线网站的一些参数篡改机会来实证证明NOTAMPER的使用。此外，从这些机会出发，我们为这些应用程序/网站中的大多数开发了具体的实例。我们的实例证明了严重的安全问题：在银行进行未经授权的货币交易，在购物车中添加未经授权的折扣，等等。

本文的组织结构如下。在第2节中，我们通过一个运行的例子提供了动机，预先明确地制定了问题，并提出了我们方法的高层次概述。第3节描述了NOTAMPER的结构和我们的方法所解决的主要技术挑战。第4节描述了NOTAMPER使用的算法。第5节介绍了我们对几个真实世界的例子和网站的评估。第6节介绍了相关的工作，在第7节我们得出结论。

2. 高级别的概述

图1展示了一个小型网络应用的客户端，作为本文的运行实例。这个例子是基于真实世界的场景。它展示了一个购物车应用程序的结账形式，其中用户已经选择了

```
<script type="text/javascript">

函数validateForm() {
 var copies, copies2;
 copies = document.getElementById('copies');
 copies2 =
 document.getElementById('copies2'); if(
 copies.value < 0 || copies2.value < 0){
   alert("Error: Need positive
   copies"); return false;
 }
 返回true。
}

函数validateText() {
 var dir;
 dir = document.getElementById('direction');

 var textRE = / ([a-zA-Z]) */;

 var bReturn = textRE.match(dir);
 if(! bReturn)
   alert("Error: No special
 characters."); return bReturn;
}
</脚本>
```

图2：
运行实例的JavaScript验证。`validateForm()`在表单提交时被调用，`validateText()`在交付说明改变时被调用。

两个产品的购买。该表格要求用户提供每种产品的数量、要收取的信用卡（显示在以前使用过的卡的下拉列表中）以及任何特殊的交货说明。在这些数据被提交给服务器之前，客户端的JavaScript代码（图2）确保每个产品的数量都是非负数，并且交付指示不包括特殊字符。`onsubmit`事件处理程序执行这一验证，如果发现数据有效，就将其提交给服务器，或者要求用户重新输入，并给出适当的错误信息。然而，服务器未能复制这些验证检查，使一些攻击成为可能。

攻击1：负数。
我们在一家在线计算机设备零售商的网站上发现了以下攻击。通过禁用JavaScript，恶意用户可以绕过对每个产品（参数copies和copies2）数量的验证检查，为一个或两个产品提交一个负数。提交两个产品的负数有可能导致用户的账户被*记入*；然而，由于服务器上涉及借记和贷记的信用卡交易的差异，这种方法很可能被挫败。然而，如果一个产品的数量为负数，而另一个产品的数量为正数，这样得出的总数为正数，则负数作为总价格的回扣。在图中，选择的数量分别是-4和1，结果是1600美元的 "折扣"。

攻击2：向另一个用户的账户收费。
我们在一家金融机构发现了一个类似的漏洞，并能够在任意账户之间转移资金。当表格被创建时，一个下拉列表被填充了用户的信用卡账号（参数支付）。通过提交一个不在此列表中的账号，恶意用户可以购买产品并向其他人的账户收费。

攻击3。模式验证绕过。
这种攻击使我们能够进行跨站脚本攻击，并升级到管理权限。网络表格确保交付指示（参数

仪表 方 向）只包含大写和小写字母。特别是，不允许使用特殊字符和标点符号，以防止对服务器的命令注入攻击。通过规避这些检查，恶意用户可以发起诸如XSS或SQL注入等攻击。

2.1 问题描述

在表单提交中，一个网络应用程序的客户端从用户那里索取n个字符串输入，并将它们发送到服务器进行处理。从形式上看，每个字符串输入是一个来自某个字母表Σ的有限字符序列。我们将此类输入的n个元组表示为I，所有此类I的集合表示为I。

$$\mathrm{i} = \sigma^* \times \sigma^* \times \cdots \times \sigma^*$$

从概念上讲，客户端和服务器都执行两个任务：检查用户提供的输入是否满足某些约束条件，并将错误传达给用户或处理这些输入。对于目前的问题，我们忽略了客户端和服务器上的第二个任务，而完全关注约束检查任务。从形式上看，约束检查代码可以被表述为一个func- $tion\{true, false\}$，其中$true$和$false$表示一个错误。我们用p_{client}来表示客户端的约束检查功能，用p_{server}来表示服务器上的约束检查功能。

问题的提出。我们的方法是基于这样的观察：对于许多典型的表单处理网络应用，在p_{server}和p_{client}之间存在一种特殊的关系：p_{server}比p_{client}更有限制性。因为服务器通常比客户端能获得更多的信息，所以p_{server}有时会拒绝p_{client}所接受的输入。例如，当为一个网站注册一个新用户时，服务器会保证用户的ID是唯一的，但客户端不会。相反，如果p_{server}接受了一个输入，那么我们希望p_{client}也能接受它；否则，客户端就会对合法用户隐藏服务器端的功能。因此，我们希望对于所有的输入I

$$p_{server}(I) = true \Rightarrow p_{client}(I) = true。 \quad (1)$$

服务器端的约束检查对于那些在
当这一含义的否定成立时，就把*我放*在了这里。

$$p_{server}(I) = true \wedge p_{client}(I) = false。 \quad (2)$$

我们把满足(2)的每个输入称为潜在的*参数篡改攻击向量*。

在实践中，参数篡改的攻击矢量有时会出现，因为开发者根本没有意识到客户端的检查应该在服务器上复制。但是，即使开发者想在服务器上复制客户端的检查，服务器和客户端通常是用不同的语言编写的，这就要求客户端和服务器的检查要独立实现和维护。随着时间的推移，这两个代码库中的验证检查可能变得不同步，为参数篡改攻击打开大门。

2.2 方法概述

我们的目标是通过对服务器的黑箱分析，自动构建行使参数篡改漏洞的输入。黑盒服务器分析的好处是，我们的方法对服务器的实现（例如，PHP、JSP、ASP）是不可知的，因此是广泛适用的，甚至包括过时的和专有的服务器技术。黑盒服务器分析的缺点是，我们可能没有足够的信息来消除假阳性和假阴性。特别是，我们可能无法合理地生成服务器应该被测试的所有输入信息即使是那些我们生成的输入，也没有可靠的方法来知道服务器是否接受它们。因此，我们的目标是识别参数篡改的*机会*，同时尽可能不需要人工指导。特别是，我们要求人类开发者/测试者做两件事：提供关于客户端不存在的重要信息的提示，以及检查我们发现的参数篡改机会是否是真正的漏洞（也许通过生成实际的漏洞）。

我们的高层次方法如下。在来源为HTML和JavaScript的客户端上，我们提取f_{client}：使用程序分析的技术对p_{client}进行逻辑表示。随后，使用逻辑工具，我们生成输入$_{h1,..,i}$我们把每个这样的输入称为*敌对的*，因为它是为了说明可能的参数篡改攻击。此外，我们还生成输入$_{b1,.}$
$..client_j$我们称每个这样的输入为*良性的*，因为它是一个服务器会正常处理的输入。在我们的方法中，我们接受开发者的提示，以确认这些生成的输入确实被正常处理。

良性输入有助于评估哪些敌对输入是实际的机会。我们向服务器提交每个敌对和良性的输入，产生响应$_{H1}$，
$.., H_n$和$_{B1}$，$..,$
$_{Bm}$，分别。然后我们将每个敌对响应H_i与良性响应$_{B1}$，$..,$
$_{Bm}$来产生一个分数，代表服务器接受h_i的可能性。直观地说，每一个良性响应都代表来自服务器的成功信息，敌对响应与良性响应的相似度越高，敌对输入就越可能是成功的，因此是一个参数篡改机会。

最后，敌意输入和响应被提交给人类测试者，并按照与良性响应的相似度进行排序。然后，测试者可以自由地验证敌意输入是真正的参数篡改漏洞，并通过向服务器发送修改后的敌意输入来探索每个漏洞的严重程度。

讨论。虽然我们认为观察结果(1)在许多交互式表单处理应用中是成立的，但有时并不成立，例如，当服务器是一个通用的网络服务（如谷歌地图），而客户端是一个使用该服务的一部分的应用（如伊利诺伊州的地图）。虽然这不在我们的研究范围之内，但NOTAMPER可以在这种情况下使用，即用手动构建的f_{client}取代从HTML/JavaScript中自动提取的f_{client}。良性/恶意输入的构建和它们的评估将如上所述进行。换句话说，NOTAMPER将f_{client}（无论它是如何生成的）视为服务器预期行为的近似规范，然后试图找到无法满足该规范的输入。因此，NOTAMPER可以被看作是一个具有程序分析前端的形式验证工具，用于提取预期行为的规范。

最后，由于黑箱分析的固有局限性，我们的方法不能提供完整性的保证；相反，我们通过我们发现的真实漏洞的严重程度来证明我们方法的效用。

3. 架构与挑战

在这一节中，我们讨论了NOTAMPER的结构和它的每个组件所解决的高水平的挑战。在第4节，我们讨论了我们的实现，重点是我们的con- straint语言和算法。

图3显示了高层次的架构：构成NOTAMPER的三个组件以及它们如何互动。首先，给定一个网页，HTML/JavaScript分析器构建逻辑表格，代表每个表格的约束检查功能。

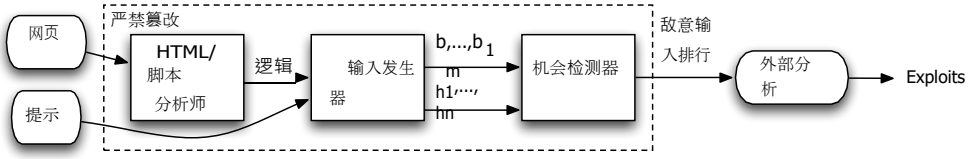

图3： NOTAMPER**端到端架构和应用。**

该网页。对于我们正在运行的例子，HTML / JavaScript分析器构建了以下公式（*fclient*），表示参数copies和copies2必须大于或等于0；参数方向不能包含特殊字符；参数payment必须是下拉列表中的一个值。

$$\begin{aligned} & \textit{copies} \geq 0 \wedge \text{copies2} \geq 0 \\ & \textit{directions} \in \texttt{[a-zA-Z]*} \\ & \textit{payment} \in \\ & \quad \texttt{(1234-5678-9012-3456 | 7890-1234-5678-9012)} \end{aligned}$$

输入生成器采用所产生的公式和用户提供的任何提示，为服务器构建两套输入。
(i) 那些服务器应该接受的（良性输入$b_1, ..., b_m$）和
(ii) 那些服务器应该拒绝的（敌对的输入$h_1,..., h_n$）。在我们的例子中，输入生成器构建了一个良性输入（满足上述公式的变量赋值）。

$$\{\textit{copies} \rightarrow 0, \text{copies2} \rightarrow 0, \textit{directions} \rightarrow \text{""}, \textit{payment} \rightarrow \texttt{1234-5678-9012-3456}\}。$$

输入生成器还可以构造一些敌对的输入（伪造上述公式的变量赋值）。下面是两个这样的输入，除了(1)副本小于0和(2)方向包含特殊字符外，其他都和上面一样。

1. $\{\textit{copies} \rightarrow \text{-1}, \text{copies2} \rightarrow 0, \textit{directions} \rightarrow \text{""}, \textit{payment} \rightarrow \texttt{1234-5678-9012-3456}\}$。
2. $\{\textit{copies} \rightarrow 0, \text{copies2} \rightarrow 0, \textit{directions} \rightarrow \text{''; *\&@''}, \textit{支付} \rightarrow \texttt{1234-5678-9012-3456}\}$。

第三个组件，机会检测器（Opportunity Detector）接收敌意和良性输入，为每个输入生成服务器响应，根据它们作为参数篡改机会的可能性对敌意输入进行排序，并将结果提交给外部测试人员进行进一步分析。

下面我们将更详细地讨论这三个部分各自应对的挑战。

3.1 HTML/JavaScript分析器

网页初始化。NOTAMPER的JavaScript分析特别关注有关表单验证和提交的功能/属性。为了分析与表单处理有关的JavaScript代码，NOTAMPER模拟了一个类似于浏览器中的JavaScript解释器的环境，包括文档对象模型（DOM）。在这样的环境中，用户的互动导致JavaScript代码被执行，从而导致JavaScript环境和DOM的变化。(用户互动可以通过AJAX触发异步服务器请求，但我们的实施方案目前不支持AJAX）。

为了分析实际执行验证的JavaScript代码，了解浏览器首次加载表单时存在的全局JavaScript状态往往很重要。为了计算这个全局状态，NOTAMPER执行了所有的初始化代码，这些代码包括确切地说是网络表格。它下载外部的JavaScript，执行内联的JavaScript片段，并跟踪全局变量的变化。

识别JavaScript验证代码。为了构建*fclient*，HTML/JavaScript分析器必须识别与参数验证有关的代码片断，并了解这些片断是如何交互的。这可能很困难，因为验证程序可以以两种不同的方式运行。(1)当表单被提交时；(2)当用户在表单上输入或更改数据时，在事件处理程序中。

一个状态机自然地模拟了JavaScript的事件驱动执行。每个状态都代表了用户输入的数据和指示哪些数据包含错误的标志。当用户提供或编辑数据时，JavaScript代码会验证数据并相应地更新错误标志，从而产生一个状态转换。客户端对某些特定数据集的约束在理论上可能取决于用户通过状态机输入数据的路径，因此公式*fclient*可能取决于该状态机的结构。

NOTAMPER通过分析JavaScript事件处理程序来解决这个难题，就好像它们在表单被提交时都被执行了一样。这样做的好处是计算上的：它避免了手动模拟事件或考虑事件发生的顺序的需要。但它也反映了用户对数据输入的一个合理假设--数据输入的顺序并不影响数据的有效性。对于那些数据输入的顺序很重要的情况，我们的分析可能会有过多的限制，例如，考虑到所有的事件处理程序可能会模拟出相互排斥的事件的发生。

分析JavaScript的验证代码。一旦确定了有助于*fclient*的验证程序，就必须对它们进行分析。这些代码可能跨越几个功能，每个功能可能包括多个控制路径。每条控制路径都可能对输入有一套独特的约束，这就要求对*所有路径的程序间*进行分析。此外，JavaScript可能强制执行不依赖于用户输入的约束，例如，不允许通过全局变量重复提交表单。我们面临的挑战是，只提取某个特定的JavaScript验证代码对输入的约束。

NOTAMPER通过采用混合的具体-符号执行方法[9]来分析JavaScript，并确定对用户提供的数据实施的约束，从而解决这一挑战。符号执行提供了对验证代码中所有控制路径的覆盖，并模拟了对用户提供的数据的验证。具体执行使NOTAMPER能够忽略不依赖符号输入的代码片段，并为符号执行提供一个适当的初始化环境。

解决文档对象模型（DOM）引用。Java-脚本验证程序通常使用DOM来访问表单输入控件。在我们模拟的JavaScript环境中，将JavaScript中的DOM引用与HTML输入控件相关联。

编写DOM是一件非同小可的事情，但对于构建*fclient*来说是必要的。此外，DOM可以被JavaScript动态地修改，通过添加
/删除额外的输入控制或禁用/启用现有的输入控制。

NOTAMPER通过从给定的HTML中构建DOM的相关部分来解决这一难题，在执行过程中，它可用于JavaScript的具体符号评估引擎。此外，这个DOM是通过模拟用于修改DOM结构的DOM函数来维持的。

3.2 输入发生器

给予输入发生器的逻辑公式是用字符串约束的语言写的（在第4节中描述）。输入生成器包括两个独立的任务。(i) 构建新的逻辑公式，其解对应于敌对的和无意义的输入；(ii) 解决这些公式以建立具体的输入。在这里，我们重点讨论第一项任务，第二项任务留待第四节讨论。

避免虚假的拒绝。两种表面但常见的服务器端参数验证形式从天真的分析中隐藏了服务器的漏洞：检查所有 "必需"的变量是否有值，以及检查所有变量是否有正确类型的值。如果不考虑这种简单的参数验证，NOTAMPER就只能发现一些参数被篡改的机会。

为了应对这一挑战，输入生成器构建了敌对和良性的输入，其中所有需要的变量都有值，所有的值都是正确的类型。NOTAMPER采用启发式方法（第4节）来计算所需变量和变量类型的列表，该方法可以被手动覆盖。

产生正交的敌意输入。每一个敌意的输入最好都能探测到服务器上的一个独特的弱点。两个被服务器以相同原因拒绝的敌对输入（由服务器上的相同代码路径）是多余的。在我们运行的例子中，客户端要求一个变量（`copies`）大于或等于0，另一个变量（`direction`）被赋予一个不包含标点符号的值。为了避免冗余，NOTAMPER应该生成一个敌对的输入，其中`copies`违反了约束（小于0），但`direction`满足了约束（不包含标点符号），另一个输入是`copies`满足了约束，但`direction`不满足。

为了生成这样的正交输入，输入生成器将*fclient* 转换为二元正则形式（DNF）[1]并为每一个disjunct构建一个敌对的输入。一般来说，每个二分法代表的是违反*fclient*的输入，其原因与其他二分法不同。

应对不完整的信息。有时公式*fclient*未能包含足够的信息来生成一个真正的be-
nign输入或暴露真正漏洞的敌意输入，但人类测试人员却愿意提供这些信息。例如，许多网络表单只接受包括有效的登录ID和密码的输入，但客户端代码本身并不提供有效的ID和密码列表；在这种情况下，*fclient*不包含足够的信息来生成将被服务器接受的输入。

为了解决这个问题，输入生成器接受提示，指导搜索敌意和良性输入。这些提示采取逻辑约束的形式（与*fclient*的语言相同），并去掉了σ。例如，为了强迫登录变量`user`的值为"alice"，密码变量`pass`的值为 "alicepwd"，则用户将提供逻辑语句 `user` = "*alice*" ∧ `pass` = "*alicepwd*"。

解决状态变化问题。网络应用程序经常在服务器上存储信息，而网络表单提交会改变这种状态。这可能导致有效输入的集合随着时间的推移而改变。例如，一个用户注册的Web表单会要求提供一个尚未被选择的登录ID。用相同的登录ID提交两次表格将导致第二次尝试时被拒绝。这是有问题的，因为NOTAMPER提交了许多不同的输入，以检查不同类别的潜在漏洞，但登录ID是必须的，而且必须是唯一的。

为了解决这个问题，输入生成器把需要有唯一值的变量列表作为一个可选参数，并确保分配给这些变量的值*在不同的*提交中是不同的。在我们的评估中，生成的输入中某些变量都有唯一的值，这足以解决服务器端的状态变化，尽管在一般情况下，更复杂的灰盒机制是必要的（例如，在测试案例之间回滚服务器端数据库的能力）。

摘要。
总的来说，输入生成器希望得到以下参数（1）公式逻辑*fclient*（代表客户端接受的输入集合），（2）所需变量的列表。(3) 变量的类型，(4) 一组手工提供的约束条件（提示），(5) 唯一变量的列表（(4)和(5)是可选的）。它生成有敌意的输入（一组I，使$f_{client}(I)=false$）和良性的输入（一组I，使$f_{client}(I)=true$），这样所有需要的变量都有值，所有的值都是正确的类型，所有的手动约束都被满足，并且每个独特的变量在所有的输入中都有不同的值。输入生成器的所有参数都是由HTML/JavaScript分析器计算出来的（如第4节所述）。

3.3 机会检测器

输入生成器产生一组敌对的输入$_{h1,..,\ hm}$和一组良性输入$_{b1,\ ..,\ bm}$。机会检测器的目标是确定哪些敌对的输入实际上是参数篡改的机会。主要的挑战是，NOTAMPER必须确定一个给定的敌意输入是否被服务器接受，同时把服务器当作一个黑盒子。

NOTAMPER通过对敌意输入的服务器响应与良性输入的服务器响应在结构上的相似程度进行排序来应对这一挑战。敌意响应与良性响应越相似，敌意输入就越可能是一个篡改参数的机会。

在我们运行的例子中，考虑一个敌对的输入，其中参数`copies`被分配为一个负数。如果服务器未能验证`copies`是一个正数，敌对和良性响应都将呈现一个确认屏幕，唯一的区别是副本的数量和总价。另一方面，如果服务器检查出一个负数的副本，敌意响应将是一个错误页面，这可能与确认屏幕有很大的不同。

4. 算法与实施

本节详细介绍了NOTAM-
PER采用的核心算法。除了一个算法外，所有的算法都操纵着一种逻辑语言，用于代表客户对用户数据实施的限制。目前，NOTAMPER采用的语言是建立在算术和字符串约束之上的。它包括通常的布尔连接词：连接（∧），分离（∨），和否定（¬）。原子连接使用<，≤，>，≥，=，和变量-
来限制变量长度。
除了上述运算符外，还可以用∈、/∈来表示能值。唯一不显眼的运算符∈和/∈，表达的是记忆。

[1]根据我们的经验，DNF的转换费用很低（尽管它的最坏情况下的指数特性），因为*fclient*的结构很简单。

```
< 发送> ::= < 原子> < 共轭>|< 分歧>| < 否定>
< conj> ::= (< sent ∧ ><sent> )
< disj> ::= (< sent ∨ ><sent> )
<否定> ::=¬( <发送> )
< atom> ::= (< term> < op> < term> )
< op> ::= < |≤ |> |≥ |= ||   |∈ |∉
< term> ::= < var>|‖< num> < str> < len> < reg>
< reg> ::= perl regexp
< len> ::= len (< var> )
< str> ::= "< var> "
< var> ::= ? [a-zA-Z0-9]*
< num> ::= [0-9]*
```

表1:由NOTAMPER生成的公式的语言

正则语言的成员资格约束。例如，下面的约束要求*x*是一个非负的整数：*x* [0-9]+。表1显示了定义约束语言的Backus-Naur Form（BNF）语法。

下面我们按照NOTAMPER执行的顺序描述算法：（1）从HTML和JavaScript中提取客户端约束，（2）生成Input Generator组件所接受的额外输入，（3）构建逻辑公式，其解决方案是敌对和良性输入，（4）解决这种逻辑formulas，以及（5）识别敌对和良性服务器响应之间的相似性。

4.1 客户端约束提取

提取客户端对用户提供的数据所实施的约束，并将其逻辑地表示为*fclient*，分两步完成。首先，一个HTML分析器从一个给定的网页中提取三个项目。(1)通过HTML强制执行的单个表单字段的约束 (2)代表加载网页时执行的JavaScript以及客户端执行的参数估值的JavaScript的代码片断，以及 (3)表单的DOM表示。第二，我们的具体/象征性的JavaScript评估器使用(3)在对(2)进行符号评估时，提取额外的约束条件，然后与(1)相结合。其结果是公式*fclient*。

第1步：HTML分析器。

表2通过实例总结了每个HTML内放控件所施加的约束。在我们的运行示例中，有一个下拉列表的支付控件，其中包括两个信用卡值。由此产生的约束要求支付必须是该列表中的一个值，如下图所示。

```
支付 ∈
  (1234-5678-9012-3456 | 7890-1234-5678-9012).
```

通过收集所有的事件处理程序（和相关的脚本），并生成一个单一的函数来调用所有的事件处理程序，当所有的事件处理程序返回真值时，就返回真值，从而构建一个代表客户端所执行的参数验证的JavaScript片段。所有网页中的JavaScript都被添加到上述脚本的前言中，以便为表单验证的JavaScript提供初始化环境。表单的DOM表示是通过递归建立上述JavaScript片段中的文档对象来构建的，也就是说，被分析的表单被初始化为文档对象的一个属性，该对象将输入控件作为属性捕获。此外，文档对象模拟了处理表单所需的一小部分核心方法，例如getElementById。目前，我们不支持document.write或document.innerHTML，我们正在努力增加对它们的支持。

控制	例子	限制条件
选择	< 选择 name=**x**> <选项值="**1**″> <选项值="**2**" > <选项值="**3**" >	**x** ∈ **(1|2|3)**
广播电台/检查箱	< 输入 type=radio name=**x** value="**10**″> < 输入 type=radio name=**x** value="**20**″>	**x** ∈ **(10|20)**
隐藏	< 输入 name=**x** type=**hidden** value="20″>	**x = 20**
最大长度	< 输入 name=**x maxlength=10** type=text/password>	**len(x) ≤ 10**
只读	< 输入名称=**x 只读** value="20″>	**x = 20**

表2： 由HTML表单控件施加的约束。

第二步：JavaScript符号评估器。从一个给定的JavaScript片段中提取参数验证约束的关键观察点是，只有当代码返回真时才会发生表单提交。在最简单的情况下，代码包括状态ment return true或return <boolexp>，其中<boolexp>是一个布尔表达式。理论上，该代码可以返回任何被JavaScript转换为真的值，但根据我们的经验，前两种情况更为常见。这个观察结果导致了提取约束条件的关键视点：*确定所有导致所有事件必理函数返回值为真的程序条件。*

为了提取验证约束，符号分析器开始具体地执行验证代码。当遇到带有符号变量的布尔表达式时，执行分叉：一个假设布尔表达式为真，另一个假设它为假。这两种执行方式都会复制现有的变量值（程序状态），除了那些受假设布尔表达式为真或假影响的变量。然后再恢复具体的执行。支持的DOM修改API在DOM上的作用是专门针对分叉的。

对于一个给定的程序位置，程序条件是控制到达该点必须满足的条件集合。如果一个叉子返回false，它将被停止并被丢弃。如果一个叉子返回真，它就会被停止，到达该点的程序条件会被注意到。此外，此时的DOM表示法重新反映了提交表单时HTML输入控件的状态，包括由JavaScript完成的任何修改。在这个分叉上检查的约束条件是通过结合DOM表示中启用的控件的约束条件和使用连词（ ）的程序条件计算出来的。

一旦所有的分叉都被停止，*fclient*的计算方法是将每个返回真值的路径的公式与disjunction（）结合起来。对于运行中的例子，一个控制路径成功地返回了为真，从而得出以下公式。

```
∧ ¬(copies < 0 ∨ copies2 < 0))
  方向 ∈ [a-zA-Z]*
```

然后将上述内容与可变支付的约束条件相结合之前提到的生成*fclient*的方法。

4.2 敌意输入指导

NOTAMPER的整体成功主要取决于产生有趣的敌意输入。下面我们讨论一下HTML的启发式方法/JavaScript组件用来从一个给定的网页中计算这些值。这些启发式方法通过手动测试和完善

对我们的两个测试应用程序（SMF和LegalCase）进行了检查，但在我们的其余实验中没有改变。

初始值。
在生成*fclient*时，NOTAMPER使用启发式方法来确定表单字段的默认值的意图。一些表单字段被初始化为简单说明预期输入类型的值，例如，产品拷贝数的值是1。其他表单字段被初始化为一个不能改变的值，如果提交成功，例如，一个隐藏字段被初始化为一个会话标识符。目前，NOTAMPER使用隐藏字段的默认值作为*fclient*中的约束，并认为所有其他字段的默认值是预期值的说明。在这两种情况下，初始值的列表被提供给输入生成器，并用于其他启发式方法，如下所述。

类型。每个变量的类型控制着敌对和良性输入中可能出现的值的集合。选择适当的类型可以极大地提高成功的几率。在我们的例子中，如果`copies`的类型是正整数，那么输入生成器就不会发现`copies`小于0时出现的漏洞。同样地，如果副本的类型都是字符串，那么生成器随机选择一个代表负整数的字符串的可能性就不大。目前，NOTAMPER为每个变量选择类型的依据是：(i)它在算术约束中的出现，(ii)与该变量相关的HTML小部件，以及(iii)它的初始值。出现在一个算术约束中意味着一个数字类型。一个列举了一组可能的值的HTML部件意味着从列举的值中的所有字符集中抽取一个值。一个初始值是数字的，也意味着一个数字类型。除非有证据表明需要真实的值，否则就假定是整数。

必要的变量。所需变量的清单确保每个敌意的输入都包括列表中每个变量的值。选择太小的列表有可能使敌意输入被拒绝，因为它们没有通过服务器对所需值的要求，而选择太大的列表可能导致服务器拒绝敌意输入，因为不必要的变量被赋予无效的值。NOTAMPER采用了两种技术来估计所需变量。一种是分析HTML，看是否有迹象表明需要某个变量，例如，字段标签旁边的星号。另一种方法是从*fclient*中提取需要非空的变量，例如，该变量不能是空字符串，或者该变量必须是几个值中的一个（来自下拉列表）。

独特的变量。当一个变量出现在唯一变量列表中时，每一对主机输入都会因该变量的值而不同。这很有用，例如，在测试用户注册页面时，提交相同的用户ID两次将导致拒绝，因为该ID已经存在。然而，选择太大的列表，会导致产生较少的敌意输入，因此发现的漏洞也较少。例如，如果一个字段只能取三个值中的一个，并且要求在所有敌对输入中是唯一的，那么最多只能产生三个输入。目前，NOTAMPER对它所猜测的变量应该是唯一的，这是很保守的。如果有任何迹象表明，一个变量只能取少量的值，那么它就不会被列入唯一列表。

4.3 输入生成

输入生成器在constraint语言中构建了一系列公式，这些公式的解对应于敌对和良性输入。这里我们将详细介绍良性输入和敌意输入的公式构造有何不同。

良性输入。为了生成满足*fclient*的良性输入，NOTAMPER将*fclient*转换为DNF[1]，并将每个disjunct增强到

len(< var>) = *len*(< var>)	< var> ⊗ < var> ⊗ < var>
< var> ≠< var>	< var> ⊗ *len*(< var>)
< var> ≠*len*(< var>)	*len*(< var>) ⊗ *len*(< var>)
len(< var>) ≠*len*(< var>)	< var> ⊕ < reg>

表 3: 缩小的约束语言。∧ 和 ∨ 覆盖上述原子。⊗是<, >, ≤, ≥中的一个。⊕是∈或/∈。

在用户提供的约束条件σ和所需的变量和类型约束条件下，为每个disjunct找到一个解决方案。

在运行的例子中，假设*fclient*是公式

$$(copies > 0 \vee copies = 0) \wedge (direction \in \texttt{[a-zA-Z]*})。$$

NOTAMPER为*副本*>0个∧ *方向* ∈找到一个解决方案`[a-zA-Z]*`，另一个是*copies* = 0 *directions* ∈`[a-zA-Z]*`。如果*copies*的类型是`[0-9]+`，*direction*的类型是`[a-zA-Z0-9]*`，NOTAMPER包括约束*copies* ∈`[0-9]+`和*方向* ∈`[a-zA-Z0-9]*`。如果变量*名称*是必需的，并且类型为`[a-zA-Z]*`，NOTAMPER包括con-straint *name* `[a-zA-Z]*`。如果σ是非空的，NOTAMPER也包括它。

满足唯一变量的约束是通过跟踪每个生成的输入分配给每个变量的值和添加约束来完成的，以确保每个唯一变量的下一个生成值与之前生成的变量不同。

敌意输入。为了产生敌对的输入，NOTAMPER从*fclient*开始，而不是*从fclient开始*，然后像良性的情况一样进行，但有一个例外：为所需的变量填值。考虑*fclient*的DNF中的任何disjunct δ。如果所有需要的变量都出现在δ中，NOTAMPER只需找到一个满足δ的变量赋值并返回结果；否则，NOTAMPER用δ中没有出现的需要的变量的值来增加赋值。

为此，它找到满足*fclient*的值。我们希望，如果服务器拒绝输入，那是因为出现在δ中的变量，而不是其余的变量；否则，就不清楚服务器是否进行了充分的验证以避免潜在的漏洞δ。

在上面的例子中，*fclient*的二分法形式是¬产生一个有两个不连接点的公式。

$$\neg(\textit{副本}>0) \wedge \neg(\textit{副本}=0)$$
$$\neg(\textit{方向} \in \texttt{[a-zA-Z]*})$$

假设*副本*和*指示*都是需要的。第一个disjunct不包括*方向*，第二个disjunct也不包括*副本*。在用例如*copies* =1解决第一个disjunct之后，NOTAMPER给*direction*分配了一个满足原始公式的值，即满足*direction* `[a-zA-Z]*`。同样地，在解决了产生*方向* ∈值的第二个disjunct后，NOTAMPER赋予*copies*一个满足原公式的值，例如，*copies* = 1。

4.4 限制条件的解决

为了解决约束语言中的公式，NOTAMPER使用了一个建立在HAMPI[13]之上的自定义编写的约束解算器，这个解算器可以处理固定长度的*单个*变量上的常规语言约束的组合。我们的公式涉及多个变量，因此我们开发了自己的程序，使用HAMPI，如下所述。

NOTAMPER通过将给定的公式转换为DNF[1]，并独立地解决每一个disjunct，来处理disjunction。对于一个给定的

算法1 SOLVE(vars, φ, asgn, BOUNDS) 1: **if**

vars = **then** ∅**eturn** asgn
2：价值 := ∅
3: var := CHOOSE(vars, φ, asgn, BOUNDS)
4: **for all** i in LOW(BOUNDS(var)) ... HIGH(BOUNDS(var)) **do**
5: 如果NUMERIC-VAR(var)，那么
6: 如果SAT(φ, ∪ asgn{vari})那么
7:newasgn:=SOLVE(vars-{var}, φ, ∪ asgn{vari}, → 边界)
8: 如果newasgn = *unsat*，则返回newasgn
9: 否则
10: 如果不是SAT(φ ∧ len(var)=i, asgn)，则转到下一个i
11: 循环
12:val:= HAMPI(φ ∧varvar∨alues, i)
13: **if** val = unsat **then** goto next i
14:value:= ∪ values{val}
15: 如果SAT(φ, ∪ asgn{varval})那么
16:newasgn := SOLVE(vars-{var}, φ ,asgn ∪ {var→val}, BOUNDS)
17: 如果newasgn = *unsat*，则返回newasgn
18: 返回 *unsat*

在Disjunct（这是一个联结）中，NOTAMPER进行类型判断以确定哪些变量是数字，哪些是字符串，提取所有变量的大小界限，并简化Disjunct以产生表3中的原子联结。然后应用算法1来搜索满足所产生的联结的变量赋值。

算法1的输入是一个需要估值的变量列表，一个逻辑公式，一个部分变量赋值，以及一个将每个变量映射到该变量边界的函数。它要么返回*unsat*（表示不可能有可满足的赋值），要么返回一个满足逻辑公式的给定变量赋值的扩展。

该算法的第一步是选择一个要分配的变量。目前，NOTAMPER选择了可能长度范围最小的变量。然后开始搜索。字符串变量和数字变量的处理方式不同。对于数字变量，NOTAMPER在可能的值上进行循环，并为每一个值检查分配给变量的当前循环值是否满足条件。如果满足，变量就被分配为循环值。

对于字符串，NOTAMPER在可能的长度（而不是可能的值）上循环，对于每一个满足长度条件的字符串，调用HAMPI来生成一个变量分配。HAMPI接受一个逻辑公式作为输入，其中有一个变量和该变量的长度。它要么返回*unsat*，要么返回一个满足公式的值。通过选择只有所选变量出现的约束条件子集，将给定的多变量公式φ还原为只有所选变量的公式，表示为φ_{var}。如果HAMPI找到一个满意的值，该算法会检查该值是否满足HAMPI不检查的相关约束：那些约束多个变量的约束。此外，该算法保留了一个HAMPI返回的值的列表，这样，如果在搜索的后期失败，并且需要为当前变量生成另一个值，我们可以增加给HAMPI的逻辑公式，以要求一个尚未选择的值。

一旦一个变量被赋值，算法1在删除了所选变量、原始逻辑公式、用所选变量的赋值增加的原始变量赋值和原始变量边界后，对原始变量列表进行复查。当变量列表变为空时，算法返回给定的变量赋值，表明该赋值满足所有约束。如果找不到这样的赋值，该算法就会返回*unsat*。

4.5 HTML响应比较

为了确定服务器是否接受了恶意输入，我们的方法是将服务器的响应与已知由良性（有效）输入产生的响应进行比较。由于服务器的响应是在HTML中，我们必须采用HTML相似性检测。文献中有许多针对HTML响应的相似性检测算法，其中最多的是计算树状编辑距离的算法（参考[5]）。这些算法对于来自不同来源的、可能包含类似内容的文件（例如，来自不同报纸的新闻文章）特别有用。在我们的案例中，由于HTML文档是由一个单一的网络应用程序产生的，这些响应很可能在结构上比来自不同来源的文档更加一致，因此我们使用了基于Ratcliff和Obershelp算法[16]的近似字符串匹配的自制的文档比较策略。

近似匹配。
响应比较中需要解决的一个重要问题是，HTML响应的内容经常包括一些不依赖于服务器输入的可变元素，例如，时间戳、用户名、登录人数。大量这样的元素会在良性响应中引入差异，即使输入是相同的；因此，我们采用近似的匹配策略，在与敌对响应比较之前，从良性响应中过滤掉这些噪音。

假设我们只有两个良性反应$_{B1}$和$_{B2}$。分析这些反应并提取它们的差异，往往会使页面中的嘈杂元素变迟。然后，这些嘈杂的元素可以被删除。为此，我们开发了一个工具，分析这两个响应并返回以下内容。(1) $_{B1}$和$_{B2}$中的共同句子 (2) $_{B1}$中不在$_{B2}$中的内容，以及 (3) $_{B2}$中不在$_{B1}$中的内容。元素(2)和(3)包括噪音，一旦分别从$_{B1}$和$_{B2}$中排除，我们就得到了相同的HTML文档$_{C1}$。

为了分析敌对的回应$_{hi}$，我们重复消除噪音的程序，只是这次是用文件$_{B1}$和$_{Hi}$。由此产生的HTML，$_{C2}$，产生两种可能性，取决于输入$_{hi}$是否被接受。如果输入被接受，根据我们上面的观察，服务器响应$_{Hi}$很可能与$_{B1}$相似（去掉噪声），因此结果$_{C2}$在结构上可能与$_{C1}$相同。如果输入被拒绝，服务器返回的响应很可能在结构上是不相似的，因此$_{C2}$与$_{C1}$的相似度较低。

最后一步是$_{C1}$和$_{C2}$之间的比较。同样，一个天真的比较不会起作用，因为有可能在前面的步骤中，并不是所有造成噪音的元素都被移除。例如，页面的生成时间通常被嵌入到页面本身，如果$_{B1}$和$_{B2}$的时间相同，但$_{H1}$的时间不同，那么$_{C1}$和$_{C2}$在结构上就不会严格相同。因此，我们在$_{C1}$和$_{C2}$上再次使用我们的近似匹配策略作为输入。只是这一次，我们计算两个结构之间的编辑距离，为每个敌对的输入产生一个数字值（我们称之为*差异等级*）。一个给定的敌意输入的等级越高，该输入指向潜在漏洞的可能性就越小。

复杂性。我们对HTML文件的比较策略是基于格式塔模式匹配程序[16]，它本身可以找到HTML文件之间最长的共同子序列，然后递归地找到该子序列左右两侧的共同元素。

应用	弗-标准	敌对的输入	普特。奥普公司。	忏悔。剥削？	忏悔。斐济
SMF	5	56	42		8
Ezybiz	3	37	35		16
ÄÄÄ	1	10	8		1
MyBloggie	1	8	8		7
B2evolution	1	25	21		2
凤凰卫视	1	6	5		4
呼叫中心	3	28	27		0
法律案例	2	13	9		0
smi-online.co.uk	1	23	4		2
wiley.com	1	15	4		2
garena.com	1	4	4		1
自力更生网	1	5	1		0
codemicro.com	1	6	1		0

表4：
NOTAMPER**结果的总结（机会：169，检查：50，确认的漏洞：9，假阳性：43）。**

共同序列。我们的程序在最佳情况下具有线性复杂度，在最坏情况下具有二次复杂度。

4.6 实施

HTML分析是在HTML解析器提供的API的基础上实现的。[2] 提供的API，特别是使用<form>和<script>标签的访问者。JavaScript分析是使用一个修改过的基于Narcissus JavaScript引擎的符号评估器进行的。Narcissus是一个元循环的JavaScript解释器，使用Spider-Monkey JavaScript引擎的接口。

输入发生器是作为解算器HAMPI[13]的一个封装器建立的，使用子程序库Epilog[3]，用于处理用KIF编写的逻辑表达。[4].它由1700行Lisp代码组成。

机会检测器主要用Java实现。根据约束解算器产生的输入，一个基于Java的模块将HTTP请求转发到测试服务器，保存重新赞助的处理，并实施算法来计算差异等级。

5. 评价

测试套件和设置。我们选择了8个开放源码应用程序和5个实时网站。为了选择开源应用程序，我们浏览了http://opensourcescripts.com，并找到了严重依赖网络表单的应用程序（主要是博客、商业和管理应用程序），并且没有使用AJAX。为了选择真实的网站，我们选择了我们个人使用的表单，这些表单似乎可能包含缺陷（例如，作者之一在被利用的银行有一个账户）。表5，提供了这些应用程序的一些背景细节。对于开放源码的应用程序，第2栏和第3栏分别显示了代码行数和文件的数量。第4列显示了被评估的形式所执行的约束类型，最后一列显示了应用程序所提供的功能。我们在Linux Apache网络服务器（2.8GHz双英特尔至强，6.0GB内存）上部署了应用程序，我们的原型实现NOTAMPER在标准桌面（2.45Ghz Quad Intel，2.0GB内存）上的Ubuntu 9.10下运行。

[2]http://htmlparser.sourceforge.net/[3] http://logic.stanford.edu/
[4]http://www-ksl.stanford.edu/knowledge-sharing/kif/

应用	线路准则》中的	文件	客户-侧面	使用
Ezybiz	186,691	1,103	HTML+JS	商业管理
我的bloggie	9,431	59	HTML+JS	博客
ÄÄÄ	92,712	273	HTML+JS	库存
SMF	97,304	166	HTML+JS	论坛
呼叫中心	114,959	335	HTML+JS	支持
法律案例	58,198	195	HTML	库存
PHP-Nuke	228,058	1,745	HTML+JS	内容管理
B2evolution	167,087	531	HTML	博客
smi-online.co.uk			HTML	会议
wiley.com			HTML+JS	图书馆
garena.com			HTML	游戏
自力更生网			HTML	银行业
codemicro.com			HTML+JS	购物

表5： NOTAMPER**分析了8个开放源码应用程序和5个实时网站**

5.1 摘要

我们的实验结果总结在表4中。对于每个应用程序（第1列），该表包括被分析的形式的数量（第2列），NOTAMPER生成的敌对输入的数量（第3列），篡改机会的数量（第4列），以及我们是否能够确认该应用程序的漏洞（第5列）。最后一栏列出了确认的假阳性的数量。

当网络开发人员部署分析一个网络应用时，第4栏是最重要的。开发人员只需要查看那些被服务器接受的敌意输入，并为每个敌意输入手动决定服务器是否真的有漏洞。当测试人员（黑帽团队）部署时，他们可以通过进一步实验接受的敌意输入来确认漏洞。本着类似的精神，我们试图在每个应用程序中确认至少一个漏洞。检查50个to-tal 169个机会所涉及的努力是适度的，只需要一个本科生一周的努力。我们预计经验丰富的开发人员和熟悉其应用程序的测试人员需要的时间要少得多。在这项工作中，我们在13个应用中的9个开发了工作漏洞。下面我们强调我们发现的一些漏洞。

5.2 剥削的细节

未经授权的资金转移。网上银行网站www.selfreliance.com允许客户在他们的账户之间进行在线转账。客户登录网站，指定要转账的金额，使用一个下拉菜单选择转账的源账户，并使用另一个下拉菜单选择目标账户。这两个下拉菜单包括用户的所有账户号码。

事实证明，这个应用程序的服务器没有验证所提供的账户号码是否是从下拉菜单中抽取的。因此，向服务器发送在两个任意账户之间转移资金的请求是成功的，即使登录系统的用户是两个账户的所有者。

当NOTAMPER分析这个表格时，它产生了一个敌对的输入，其中一个帐号是一个零。服务器的反应与良性输入（从下拉菜单中抽取账号）的反应几乎相同。因此，这个输入被NOTAMPER列为高度潜在的漏洞。当我们试图确认该漏洞时，我们能够在两个无关的人的账户之间转移1美元。(请注意，如果服务器检查了有效的账户号码，但未能确保用户拥有所选的账户，NOTAMPER就不会发现这个问题。

应用	形式。汇编。	普特。奥普公司。	HT-ÃÃÃ	JS	隐蔽-穴位
SMF	17	42	28	4	10
Ezybiz	28	35	19	11	5
ÃÃÃ	29	8	8	0	0
MyBloggie	23	8	8	0	0
B2evolution	47	21	8	0	13
凤凰卫视	6	5	4	0	1
呼叫中心	20	27	21	3	3
法律案例	13	9	3	0	6
smi-online.co.uk	36	4	2	1	1
wiley.com	20	4	4	0	0
garena.com	10	4	4	0	0
自力更生网	9	1	1	0	0
codemicro.com	12	1	0	1	0

表6:NOTAMPER结果的细节。

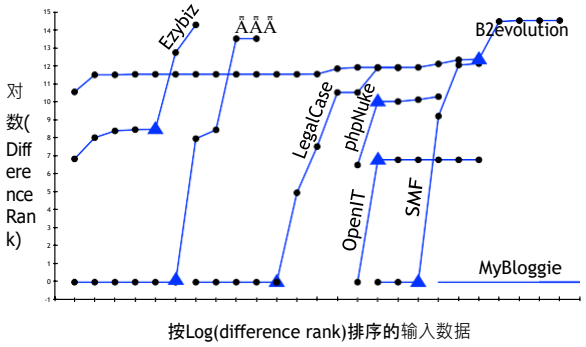

图4：
图说明了敌意输入排名的重要性，黑体三角形表示使用的阈值。

如果人类测试者提供了有效的账号作为提示，NOTAMPER就会发现问题）。

我们注意到，鉴于该银行有超过30,000名客户，这一漏洞可能产生重大影响。此外，一个成功的漏洞只需要知道受害者的账户号码，而这些号码在开具支票时通常都是共享的。银行被联系到这个漏洞，并在不到24小时内修复了它，在此期间，转账的功能被完全禁用。此外，Selfreliance从ESP Solutions (www.espsolution.net)获得了包含该漏洞的软件许可，ESP Solutions为其所有使用该功能的客户打了一个全球补丁，并在其提供网上银行功能的其他关键产品FORZA中修复了类似问题。

无限的购物回扣。网上购物网站www.codemicro.com，销售计算机设备，如硬盘、打印机、网络交换机。该表格显示了购物车的内容，并允许用户修改所选产品的数量。数量字段采用了JavaScript，限制购物者只能输入正数的数值。

当NOTAMPER分析这个表格时，它为其中一个数量字段提供了一个负数（并通过一个代理提交）。由此产生的HTML页面，虽然包含了与良性输入不同的total和数量，但其他方面是相同的，因此NOTAMPER将其列为参数篡改的机会。

我们能够进一步将其发展为另一个严重的漏洞：我们能够通过在浏览器中禁用JavaScript来添加一个负数的项目。当JavaScript被重新启用时，应用程序通过将每个产品的数量乘以其价格来计算总购买价格。因此，负数使任何购买都能获得无限的回扣。此外，这些负数被服务器成功接受，从而允许用户以优惠价格购买。

由于该网站包含了大量的计算机设备库存，因此利用这一漏洞的可能性是非常大的。网站管理员确认了该漏洞并在24小时内修复了它。

特权升级。OpenIT应用程序存储了用户的专业文件，并采用了一个网络表格来允许用户编辑他们的个人资料。登录后，应用程序为用户提供了一个网络表格来编辑她的个人资料。该表格中包括一个隐藏的字段userid，应用程序在这里存储用户的唯一标识符。当表格被提交时，服务器会更新与userid相对应的用户标识符的资料。通过将用户ID改为另一个用户的ID，就有可能更新任何用户的资料。

当NOTAMPER分析这个表格时，它产生了一个敌对的in-

放，其中userid的值是数字2（而不是初始值1）。服务器的响应与良性的输入响应（值被设置为1）几乎相同，因此被报告为一个篡改机会。

在确认了这个漏洞后，我们加强了利用，以便修改管理员用户的资料，包括一个跨站脚本（XSS）有效载荷。每次管理员用户登录时，该脚本就会执行，并将管理员的cookie发送到我们控制的服务器上。在被盗cookie的帮助下，我们重新构建并劫持了管理员会话，从而获得了管理员的所有权限。这个实验表明，参数篡改的漏洞可以作为其他特权升级攻击的启动平台。

其他漏洞的总结。补充网站[1]提供了上述漏洞和NOTAMPER发现的其他漏洞的细节。在phpNuke应用程序中，篡改一个隐藏的名字字段，允许我们在注册过程中绕过CAPTCHA挑战和confirmation页面（工作流攻击）。在OpenDB应用程序中，一个XSS脚本通过一个被篡改的国家字段被注入。在SMF应用程序中，篡改投票选项的单选按钮侵犯了投票结果的完整性。

5.3 其他实验细节

假阳性。所有的FP要么是(a)与表单输入的最大长度限制有关，不能被利用到任何严重的漏洞，要么是(b)被服务器重写，在HTML输出中没有任何可观察到的差异（Ezybiz应用中的12个）。

对潜在的漏洞进行分类。表6提供了我们实验的更多细节，按应用分类。第2栏显示了客户端约束的平均公式复杂性，即布尔连接词和原子约束的平均数量。第3栏显示了篡改机会的总数。第4栏显示了除隐藏字段外，由HTML输入控件产生的潜在漏洞的数量；第5栏显示了由Java-脚本产生的潜在漏洞的数量；第6栏显示了由隐藏字段产生的数量。**敌意输入的排名。**

对于每个表单输入，NOTAMPER向适当的应用程序发出一个HTTP请求，并计算响应的差异等级（以字节为单位的编辑距离），如前所述。为每个应用程序生成一个差异等级的排序列表。根据我们的经验，很容易确定潜在的参数篡改机会的阈值限制，因为可能被服务器接受的输入之间的差异等级往往比潜在的输入至少小一个数量级。被服务器拒绝。

我们使用图4中的图表来说明阈值。由于空间的原因，我们只从每个应用中选择了一种形式来表示，尽管我们的方法在每个应用中测试了几种形式。由于我们只对显示阈值感兴趣，该图在Y轴上画出了差异等级的对数，X轴代表根据其差异等级排序的各种输入点。我们用一个粗体的三角形来识别各种形式的阈值，我们把那些低于阈值的输入归为参数篡改机会。从图中可以清楚地看出，这种阈值是存在的，如差值等级的急剧上升所表示的。

人工干预。对于每个网络表单，我们手动向NOTAMPER提供某些类型的提示，这些提示涉及到客户端上不存在但人类测试人员可能提供的信息。例如，在SMF应用程序中，服务器需要一个*有效的*登录名来访问表单，因此我们向NOTAMPER提供了这样一个名称。在所有的表单中，我们添加了三种提示之一：证书或会话cookies，服务器要求的输入（必要的变量列表），以及要求在不同的调用中是唯一的变量（唯一变量列表）。(为了发现这些限制，我们使用NOTAMPER来生成一个满足客户端限制的输入（*felient*）。如果这个输入被拒绝，我们就检查原因并提供提示，以确保NOTAMPER可以生成一个被服务器接受的良性输入。

在我们的实验中，总共添加了3个唯一变量的提示（SMF：2，phpNuke：1）。对于除phpNuke以外的每个应用程序，我们提供了一个带有有效会话ID的cookie。此外，在所有的表格中共提供了12个重新要求的变量提示（SMF：3个表格中的5个，phpNuke：4个，B2evolution：1个，garena.com：2个）。这种人工干预受制于表格上的输入字段数量，通常每个表格需要不到5分钟。我们期望这个过程对于熟悉被测试的应用程序的真正的测试者来说会更简单。

性能。NOTAMPER中计算成本最高的部分是输入发生器。对于我们测试套件中最复杂的形式，HTML/JavaScript分析器的运行时间不到一秒。机会检测器对每个应用程序的运行时间都在亚秒级，忽略了连续HTTP请求之间的延迟，以避免服务器过载。输入生成的最昂贵的步骤是解决约束；输入生成组件的其余部分的运行时间在一秒以内。在22个表格中，约束解算器总共用了219秒解决了315个公式，每个输入的平均时间为0.7秒。对于像NOTAMPER这样的离线分析工具，这样的性能是可以接受的。

6. 相关的工作

符号评估。许多研究方法已经使用符号执行来解决广泛的安全问题，例如，自动指纹生成[7]和协议重新播放[15]。我们自己最近的工作[6]也应用了这种技术，通过自动代码转换改造PREPARE语句来消除传统网络应用中的SQL注入攻击。

关于输入验证方法的研究。缺乏足够的输入验证是网络应用中安全漏洞的主要来源。因此，在服务器端技术方面有相当发达的文献，试图遏制不受信任的数据的影响。诸如SQL注入[14, 12, 21, 4]和跨站脚本[20, 23, 22]等攻击都是研究得很好的例子，在这些攻击中，不受信任的数据会导致网络应用中的未授权行为。

漏洞分析。为了检测安全缺陷，人们对分析JavaScript代码产生了浓厚的兴趣。Kudzu[18]为了检测*客户端*的攻击，将JavaScript简化为字符串约束，而我们的重点是利用JavaScript分析来发现*服务器端的*缺陷。我们的问题设置使我们能够将我们的具体/符号评估和约束解决与表单处理的许多方面进行专业化，例如，处理客户端公式以生成可能成功的篡改漏洞的逻辑查询，以及开发许多实用启发式方法。还有一些方法是对服务器端代码进行白盒分析，以识别此类漏洞[2, 3]。然而，对于本文中提到的那种参数篡改问题的系统分析工作却很少。

模糊/定向测试。模糊测试和定向测试方法[9, 10, 19]旨在将随机/指导性突变应用于形式良好的输入，以黑盒[19]或白盒[10]的方式发现漏洞。在这个意义上，NOTAMPER与这些方法相似，因为它产生了敌对的输入来发现漏洞。然而，我们将参数篡改问题表述为检查服务器和客户端代码库的一致性，并开发了专门针对这个问题的方法，这使得它与这些方法不同。

预防架构。新的浏览器架构[11, 17, 25]建议对应用程序的客户端代码进行沙盒处理，以防止不希望发生的交互。最近的工作也旨在确保网络应用的服务器端免受恶意客户端的影响。Ripley[24]旨在通过在可信环境中复制客户端的执行来检测客户端的恶意行为。SWIFT[8]在开发新的应用程序时使用信息流分析，以确保有关信息流的保密性和完整性的约束将在客户端代码中得到满足。NOTAMPER的目标与这些方法非常不同，因为我们专注于发现现有（遗留）应用程序的漏洞。

7. 结论

在本文中，我们描述了NOTAMPER，一种检测网络应用中服务器端HTTP参数篡改漏洞的新方法。我们用客户端代码对用户数据的约束来表述我们的问题，主张用程序分析作为提取这些约束的方法，并运用约束求解来产生篡改机会。我们的工作在现有的开源网络应用程序和网站中暴露了几个严重的漏洞，我们预计随着我们对更多的应用程序的分析，发现的漏洞数量会增加。我们的结果强调了在今天的网络应用中*应该*发生的服务器端参数验证和*确实*发生的服务器端验证之间的巨大差距。

NOTAMPER目前采用黑盒服务器端分析，但在未来，我们希望增加白盒分析。白盒分析将减少假阳性/阴性率，以及运行该工具和分析其结果所需的人工劳动；然而，白盒功能将是一个可选的功能，允许NOTAMPER继续适用于白盒分析不可行的网络表格。

鸣谢

这项工作得到了美国国家科学基金会拨款CNS-0716584、CNS-0551660、CNS-0845894 和CNS-0917229。感谢Mike Ter Louw和Kalpana

Gondi的有益意见。最后，我们感谢匿名裁判员的反馈。

8. 参考文献

[1] NOTAMPER补充网站。 http://sisl.rites.uic.edu/notamper。

[2] Balzarotti, D., Cova, M., Felmetsger, V.。 Jovanovic, N., Kirda, E., Kruegel, C., and vigna, G. Saner:将静态和动态分析结合起来，以验证网络应用的净化。在*SP'08中。第29届IEEE安全与隐私研讨会论文集*（美国加州奥克兰，2008）。

[3] Balzarotti, D., Cova, M., Felmetsger, V. V., and VIGNA, G. 多模块的脆弱性分析基于网络的应用。在*CCS'07：第14届ACM计算机和通信安全会议*（美国弗吉尼亚州亚历山大，2007）。

[4] BANDHAKAVI, S., BISHT, P., MADHUSUDAN, P., AND VENKATAKRISHNAN, V. CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations.在*CCS'07：第14届ACM计算机和通信安全会议论文集*（美国弗吉尼亚州亚历山大，2007）。

[5] BILLE, P. A survey on tree edit distance and related problems.*Theoretical Computer Science 337*, 1-3 (2005), 217-239.

[6] BISHT, P., SISTLA, A. P., AND VENKATAKRISHNAN, V. Automatically Preparing Safe SQL Queries.In *FC'10: Proceedings of the 14th International Conference on Financial Cryptography and Data Security* (Tenerife, Canary Islands, Spain, 2010).

[7] BRUMLEY, D., CABALLERO, J., LIANG, Z., NEWSOME, J., AND SONG, D. Towards Automatic Discovery of Deviations in Binary Implementations with Applications to Error Detection and Fingerprint Generation.In *SS'07: Proceedings of 16th USENIX Security Symposium* (Berkeley, California, USA, 2007).

[8] CHONG, S., LIU, J., MYERS, A. C., QI, X., VIKRAM, K., ZHENG, L., AND ZHENG, X. Secure Web Application via Automatic Partitioning.*SIGOPS Oper.Syst.Rev. 41*, 6 (2007), 31-44.

[9] GODEFROID, P., KLARLUND, N., AND SEN, K. DART: Directed Automated Random Testing.*SIGPLAN Not.40*, 6 (2005), 213-223.

[10] GODEFROID, P., LEVIN, M. Y., AND MOLNAR, D. A. Automated Whitebox Fuzz Testing。在*NDSS'08。第16届年度网络和分布式系统安全研讨会论文集*（美国加州圣地亚哥，2008）。

[11] GRIER, C., TANG, S., AND KING, S. T. Secure Web Browsing With the OP Web Browser.In *SP'08:第29届IEEE安全与隐私研讨会论文集*（美国加州奥克兰，2008）。

[12] HALFOND, W. G., VIEGAS, J., AND ORSO, A. A Classification of SQL-Injection Attacks and Countermeasures。在*ISSE'06：安全软件工程国际研讨会论文集*（美国华盛顿特区，2006）。

[13] Kiezun, A., Ganesh, V., Guo, P. J., Hooimeijer, P., and ernst, M. D. Hampi:字符串约束条件的求解器。In *ISSTA '09:第18届会议论文集 软件测试与分析国际研讨会* (美国伊利诺伊州芝加哥市，2009年）。

[14] LIVSHITS, V. B., AND LAM, M. S. Finding Security Vulnerabilities in Java Applications with Static Analysis.In *SS'05: Proceedings of the 14th USENIX Security Symposium* (Baltimore, Maryland, USA, 2005).

[15] Newsome, J., Brumley, D., Franklin, J., and Song, D.Replayer:通过二进制分析自动重放协议。在*CCS '06：第13届ACM计算机和通信安全会议论文集*（美国弗吉尼亚州亚历山大，2006）。

[16] RATCLIFF, J. W., AND METZENER, D. Pattern Matching: The Gestalt Approach.*Dobbs博士杂志*（1988年7月），46。

[17] REIS, C., AND GRIBBLE, S. D. Isolating Web Programs in Modern Browser Architectures.In *EuroSys'09:第四届ACM欧洲计算机系统会议论文集*（德国纽伦堡，2009）。

[18] Saxena, P., Akhawe, D., Hanna, S., Mao, F. , MCCAMANT, S., AND SONG, D. A Symbolic Execution Framework for JavaScript.In *SP'10: Proceedings of the 31st IEEE Symposium on Security and Privacy* (Oakland, California, USA, 2010).

[19] SAXENA, P., HANNA, S., POOSANKAM, P., AND SONG, D. FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications.In *NDSS'10: Proceedings of the 17th Annual Network and Distributed System Security Symposium* (San Diego, California, USA, 2010).

[20] SAXENA, P., SONG, D., AND NADJI, Y. Document Structure Integrity:跨站脚本防御的稳健基础。In *NDSS'09:第16届年度网络与分布式系统安全研讨会论文集*（美国加州圣地亚哥，2009）。

[21] SU, Z., AND WASSERMANN, G. The Essence of Command Injection Attacks in Web Applications.In *POPL'06: Proceedings of the 33rd symposium on Principles of programming languages* (Charleston, South Carolina, USA, 2006).

[22] TER LOUW, M., AND VENKATAKRISHNAN, V. BluePrint:对现有浏览器的跨站脚本攻击的有力预防。In *SP'09:第30届IEEE安全与隐私研讨会论文集*（美国加州奥克兰，2009）。

[23] VAN GUNDY, M., and CHEN, H. Noncespaces:使用随机化来执行信息流跟踪和挫败跨网站脚本攻击。In *NDSS'09:第16届年度网络与分布式系统安全研讨会论文集*（美国加州圣地亚哥，2009）。

[24] VIKRAM, K., PRATEEK, A., AND LIVSHITS, B. Ripley。通过复制执行自动保护分布式网络应用程序。在*CCS'09。第16届计算机和通信安全会议论文集*（美国伊利诺伊州芝加哥，2009年）。

[25] WANG, H. J., GRIER, C., MOSHCHUK, A., KING, S. T., CHOUDHURY, P., AND VENTER, H. The Multi-Principal OS Construction of the Gazelle Web Browser。在*SS'09。第18届USENIX安全研讨会论文集*（加拿大蒙特利尔，2009）。