

《物联网安全结课报告》

姓 名： 秦友黄涛 班号： 193162

学 号： 20161002314 组长： 秦友黄涛

院（系）： 计算机学院 专业： 网络工程

指导教师： 姚宏 职称： 教授

2019 年 5 月

独立工作成果声明

本人声明所呈交的《物联网安全结课》，是我个人在导师指导下进行的程序编制工作及取得的成果。

尽我所知，除文中已经标明的引用内容，和已经标明的他人工作外，本报告未包含任何抄袭自他人的工作成果。对本报告的工作做出贡献的个人，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

报告作者签名：

日期： 年 月 日

目录

[第一章 DES简介 1](#_Toc4332776)

[§1.1 DES加/解密流程 1](#_Toc4332777)

[§1.1.1加密流程图 1](#_Toc4332778)

[§1.1.2解密流程图 1](#_Toc4332779)

[§1.1.3 子密匙生成流程图 1](#_Toc4332780)

[§1.1.4开发环境介绍 2](#_Toc4332781)

[第二章 关键代码实现及运行结果 2](#_Toc4332782)

[§2.1代码实现及运行结果 2](#_Toc4332783)

[§2.1.1功能函数介绍 2](#_Toc4332784)

[§2.1.2 关键代码介绍 3](#_Toc4332785)

[§2.1.3 运行结果 3](#_Toc4332786)

[第三章 实现难点、证明题 4](#_Toc4332787)

[§3.1实现难点 4](#_Toc4332788)

[§3.2证明题 4](#_Toc4332789)

1. IPSec,SSL原理

IPSec

IPSec简介

互联网安全协议（Internet Protocol Security 缩写IPSec）。工作于第三层网络层。由于网络设计之初并未考虑到安全性，网络层只保证消息可达并不安全，存在源欺骗、数据包重放和数据篡改的安全风险。IPSec就是为了解决这类问题而诞生的通过对IP协议的分组进行加密认证来保证IP协议的网络传输协议族。可提供CIA（Confidentiality：保密性；Integrality：完整性；Authentication：认证）。

其工作方式分为以下两种：

* **传输模式：**IPSec默认模式（End-to-End），适用于两台主机之间的通信。此模式下仅对IP报文的负载进行保护，即仅对第三层以上的协议提供保护且不能隐藏主机IP。要求参与通信的主机之间必须安装IPSec协议，启用该协议后会在原IP报文的头部和负载之间插入一个IPSec头部形成新的IP报文，结构如下图所示。接收方先处理IP头部，在处理IPSec头部，最后将负载交给上层协议。



* **隧道模式：**适用于两个网关之间（Site-to-Site）的通信，参与通信的两个网关是为以其为边界的网络中的其他设备提供安全通信服务。不同于传输模式，隧道模式为整个IP报文包括头部提供安全保护，所以它可以隐藏内部设备的IP地址。启用该模式后，会在原IP报文的前加上一个IPSec的头部然后继续在前面添加一个新的IP头部，结构如下图所示。内部的IP头部是网络内部的设备创建的，是通信的终点。外部的IP头部是提给IPSec功能的设备创建的，是IPSec的终点。

****

IPsec 架构



图1-1.IPSec架构

**AH：**Authentication Header（认证头），该协议主要提供数据的认证服务（验证数据完整性）以及源验证（验证数据来源）的服务，还能有效的抗重放。但不提供加密功能。如果前一个头部中“下一个头部”字段值为51，则前一个头部后面为AH头部，AH详细定义如下表1-1所示。

表1-1.AH详细定义

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8bit | 8bit | 16bit | 32bit | 32bit | N bit |  |  |  |  |
| 下一个头  编号 | 头部长度 | 保留 | 安全参数索引（SPI） | 序列号 | 验证数据 |  |  |  |  |

下一个头编号：跟在AH头部后面的首部的类型，例如6表示TCP，17表示TCP。

头部长度：整个AH首部（包括变长的验证数据）长度以32bit为单位再减2

保留：目前置0

SPI：值为[256,]，标识发送方在处理数据包使使用的安全策略

序列号：单增的计数器，在通信双方建立SA时初始化为0，发送/接受一个报文该字段都会+1，用于抗重放攻击。

验证数据：可变长度，为当前报文相关数据提供验证

**ESP：**Encapsulating Security Payload（封装安全载荷），提供数据完整性认证，数据源身份认证，和数据加密。如果前一个头部中“下一个头部”字段值为50，则前一个头部后面为ESP报文。ESP详细定义如下图1-2所示。其中SPI和序列号为ESP头部，填充数据、填充数据长度以及下一个头部组成ESP尾部



图1-2.ESP头部定义

载荷数据：如果SA采用加密则这一部分为密文，如果没有，则为明文

填充数据：为某些块加密算法使用此数据填充至合适长度

填充长度：记录填充数据的长度

**SA：**由一个三元组唯一标识（SPI，目的IP，安全协议号），SA是IPSec的基础，也是IPSec的本质。SA是通信对等体间对某些要素的约定，例如，使用哪种协议（AH、ESP还是两者结合使用）、协议的封装模式（传输模式和隧道模式）、加密算法（DES、3DES和AES）、特定流中保护数据的共享密钥以及密钥的生存周期等。建立SA的方式有手工配置和IKE自动协商两种。SA是单向的，在两个对等体之间的双向通信，最少需要两个SA来分别对两个方向的数据流进行安全保护。同时，如果两个对等体希望同时使用AH和ESP来进行安全通信，则每个对等体都会针对每一种协议来构建一个独立的SA。

SPI是用于唯一标识SA的一个32比特数值，它在AH和ESP头中传输。在手工配置SA时（长期有效），需要手工指定SPI的取值。使用IKE协商产生SA时，SPI将随机生成。

通过IKE协商的SA具有基于时间或流量的生存周期。SA失效前，IKE将为IPSec建立新的SA。

**IKE：**在实施IPsec的过程中，可以使用IKE（Internet Key Exchange，因特网密钥交换）协议来建立SA，IKE为IPsec提供了自动协商交换密钥、建立SA的服务，能够简化IPsec的使用和管理，大大简化IPsec的配置和维护工作。

IKE为IPSec协商SA的过程有两个阶段第一阶段建立两个对等体之间的安全管理链接，为第二阶段协商SA提供保护；第二阶段利用第一阶段建立的安全链接，对IPSec的SA进行协商。下面将具体描述两个阶段过程

1. 第一阶段有主模式和野蛮模式两种，流程如图1-3，图1-4所示。第一阶段的报文交换主要完成以下任务。a）协商有关安全策略即确定加解密算法、hash算法等。b）密匙生成信息交换即交换Diffie-Hellman所需参数。c）完成身份认证。



图1-3.IPSec SA协商第一阶段IKE主模式工作过程



图1-4.IPSec SA协商第一阶段IKE野蛮模式工作过程

1. 利用第一阶段建立的安全管理链接，协商构建安全数据链接参数IPSec SA。这个协商过程是安全的，加密的。此过程完成的任务如下：a）协商封装技术AH或ESP还是两者都用。b）协商加密算法和HMAC方式。c）协商传输模式，隧道模式还是传输模式

IPSec工作原理

简要的IPSec工作过程如下：

1. 设备A向设备B发送信息
2. 设备A上的IPSec驱动程序检查IP筛选器，查看数据包是否需要保护以及需要何种保护
3. 驱动程序通知IKE进行安全协商，最终建立IPSec SA（出站和入站）
4. 设备A上IPSec驱动利用SA（出站）对数据包进行更改，具体更改详情后文给出
5. 提交数据包给IP层，由IP层将数据包发送给设备B
6. 设备B接受数据包，并将数据包提交IPSec驱动
7. 设备B的IPSec驱动使用SA（入站）对数据包进行操作
8. 将操作结果提交上层给协议（例如TCP，UDP或其他协议）。

第4步对数据包的更改根据协议（AH或ESP）和工作方式（传输模式、隧道模式）的选取不同而不同。假设原始报文如图1-5所示，各个不同的数据包更改方式如对应的图片所示。



图1-5.原始IP报文



图1-6.AH协议在传输模式下数据包格式



图1-7.AH协议在隧道模式下数据包格式



图1-8.ESP协议在传输模式下数据包格式



图1-9.ESP协议在隧道模式下数据包格式

从图1-6，1-7可看出AH协议对整个报文包括头部进行了验证，由于NAT会修改IP头部信息会导致接收端验证出错，所以AH不能和NAT共存。

第二章 SSL与IPSec在VPN上的应用

第三章 论文翻译

利用蚁群灵感并行算法改进加密伪随机数发生器

**摘要：我们提出了一种并行算法来计算可能的值，以便修改伪随机数发生器（PRNGs）的样本空间，以增加其周期长度。这对于不能增加样本空间大小的低功耗设备中的发生器来说很重要。通过类比蚁群的行为来缩短并行算法的运行时间，具体如下：如果两条道路相交，则结果路径方向与加速度方向一致。就像蚂蚁倾向于增强其他蚂蚁走过的道路一样。我们通过模拟和演示高并行来评估算法效率，这使得即使是GPU这样的大规模并行系统也很适合该算法。此外，该算法的加速路径变体相比直接实现有着4%的运行时间改进。**

介绍

伪随机数是各种加密协议以及应用的重要组成部分，包括资源有限的应用例如RFID芯片或物联网。因此，伪随机数发生器只能使用很少的能量，比如使用简单的算法，但是必须提供相当水平的安全性。伪随机数发生器的一个重要标准是周期长度，即输出序列重复之前的输出数量，另外还有着其他标准，例如输出序列的良好统计特征、前向和后向保密性等等。因此设计一个提供以上特性的状态空间大小适度的（因为资源限制例如电池电量）伪随机数发生器是非常复杂的。对于已经满足上述要求但需要增加周期长度的伪随机数发生器，我们在之前的研究中提出了一种方法，该方法仅需修改少量的状态转换使得周期长度显著增加。为了找到需要改变的状态转换，需要对伪随机数的样本空间进行采样，样本空间采样是计算密集型的，因此需要使用并行计算。

在这项工作中，我们提出了一种并行算法来找到那些可修改的状态，我们称之为候选状态。并行算法的灵感来自于蚂蚁走过的路径倾向于其他蚂蚁已经走过的路径，我们通过仿真证明了我们的算法优于直接并行实现。因为并行算法的渐进并行效率极大程度的依赖于状态转化图的结构，实验表明在实践中即使对于1000个线程来说也有着良好的并行效率（70%），因此，和其他的常规结构一样，这个算法也适用于大规模并行计算引擎，例如GPU。

本文的其余部分内如如下。在第二部分，我们总结了伪随机数发生器的背景信息。在第三节中，我们提出了一种并行算法来寻找候选状态。在第四节中，我们通过模拟实验证明了我们并行算法的适用性。第五部分我们给出了结论和对未来的展望。

基础

在通信协议中PRNGs被频繁的用来产生作为噪声或者挑战还是其他作用的伪随机数。在随机种子之间，虽然没有额外的输入熵位，但在每次调用PRNGs时输出的值取决于当前状态，再通过状态转换函数将当前状态装换到下一个状态，因此它就像一个没有输入的有限状态自动机。哈希链（重复应用于某个初始值的加密hash函数），在密码协议中也有应用（Lamports认证协议）。在对初始值进行一次hash过后，hash函数就作用于hash值集合上，就像上面提到的状态转换函数一样。其他加密原语（例如流密码）也可以用这种方式建模。

在资源有限的系统中（例如移动传感器），PRNGs通常使用中等大小的状态空间，例如在8位系统中，状态空间增加8位将导致每次加法或者逻辑操作增加一跳指令，这就导致了每次加密操作增加了能量消耗，从而对电池增加负担。因此，我们打算采用具有低计算负荷的64位输出的hash函数，例如SipHash1或BLAKE2s2，以及具有相似状态空间大小的PRNGs，例如AKARI。对于加密原语有许多一般的安全要求，例如向前保密和向后保密以及PRNGs的特定模型，或测试PRNGs输出随机性的套件（Marsaglia随机测试套件、NIST测试套件）。但是，如果原函数的周期较短，则可以存储输出位，检测重复，一个实例就是对A5/1的攻击。因此，长的周期是实现前向保密的必要性。

PRNG可以建模为一个确定的状态转化函数 取值范围为 n=|M| 大小的有限状态空间。将单个状态看作节点，状态与其唯一后继状态之间的转化看作边，结果就表示为一个有向图，其中，，其中每个节点只有一个输出边。生成的图的结构反映了函数的行为信息。对于非双射过度函数，该图通常由几个弱连接的组件组成，每个组件都包含一个循环，并且通常包含几棵树，树的根部在循环之上。树往往很粗糙。图1描述了组件的结构。

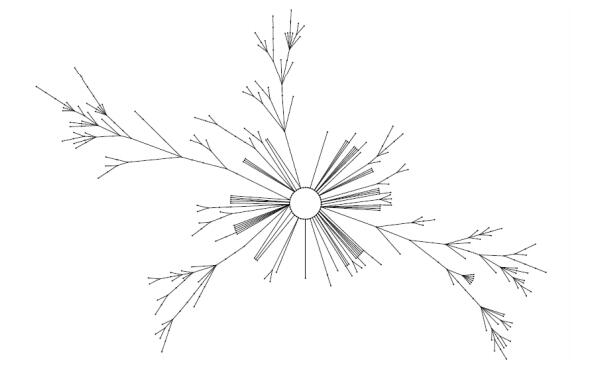


图1.典型的状态转化图的连通组件

图的属性包含了连接组件的大小和数量、循环的长度、树的最大深度。为了识别所有连通分量，必须对整个状态空间进行分析，例如通过深度优先搜索（DFS）。可以从组件的唯一后边缘开始检测循环，如果状态空间过大而无法全部检测，这可以分析状态空间的一部分，接受可能错过一个或多个组件的事实。尽管如此，这种方法任然可以提供状态结构空间的相关的有价值的信息。典型的做法就是随机选取一个起始节点，然后按照起始点所在路径进行下去直到完成一个周期。影响分析时间的路径的数量不需要非常大，因为预期的组件数量很小（见下文），通过状态图的少量样本路径已经可以接触到所有较大的组件，并提供相关周期长度。注意组件的大小远远大于其自身周期长度，这种组件中的短周期会影响到状态图中的许多子状态。

在[12]中给出了一种可以规避DFS带来的巨大内存消耗的的状态空间的分析方法，在这个方法中，每个节点必须被标记位访问状态，但当时显然不可行，解决办法就是在遍历树时只存储某些节点，如果仅存储自开始节点后的第2,4,8,16…步到达的节点，那么所需求的内存（）大小取决于步数，因此非常低。图2描述了周期检测的过程。低内存的要求带来了执行时间的开销，这个算法可能需要两倍于最佳情况的时间开销。

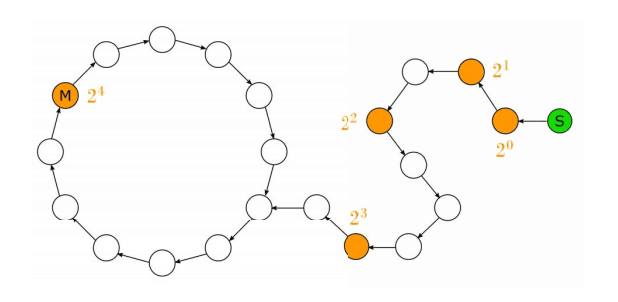


图2.周期检测

执行时间可以通过花费更多的内存空间来降低，但是平均内存开销不能大于1bit/node。一种解决方式是存储第k,2k,3k…步后到达的节点，然后检擦每个步骤是否再次到达以存储的节点。这将开销减小到了最多k个额外的步骤。但这要求每一步都会用到的一个大小为搜索数据结构，m为路径长度。因此查询时间必须为常量，例如使用低利用率的hash表。参数k可由给定的路径长度m和内存空间大小M确定，虽然后者可由使用的计算机直接确定，但是路径长度只能猜测。对于一个随机选取的状态转移函数来说，其所期望的路径长度为[13]，但是并不能保证路径长度会如此，因此如果数据结构耗尽了内存空间，就必须丢弃所有其他存储的节点来动态调整。这个数据结构可以用来缩短采样其他路径的时间：保留先前路径存储的节点，如果到达这些节点之一就停止跟随另一条路径。从图1可以看出，因为这个树的结构，路径早晚都会相遇，至少，如果两条路径在同一个组件中，对于第二条路径来说可以避免在周期周围进行重复的步骤。由于预期的树路径和周期长度都是具有可比较因子的，平均长度应该减少了两倍。因为弱连通分量的数量预计很小（[13]），许多路径都会在同一个组件中，因此通常需要付出代价来增加k以便于存储到目前为止所有采样的路径的节点。

算法的运行时间与路径的平均长度和起始点的数量成比例。由于少数的起始点就足够了，因此路径长度即使在左右，也可产生合理的分析时间。如果路径长度为约为，则将限制在以内，这允许分析一个状态空间大小为的PRNG或hash链。这些样本的结果树和循环结构可以提供有关算法安全性的有价值的见解。而且可以在置信区间内利用从属于组件的起始点的分数猜测连通分量的大小。请注意，有类似的双射函数方法，特别是期望运行时间为的Knuth算法[14]，可以通过每个节点使用1bit来实现线性化，或者通过如上所述的通过使用存储节点的方法来改进。

算法

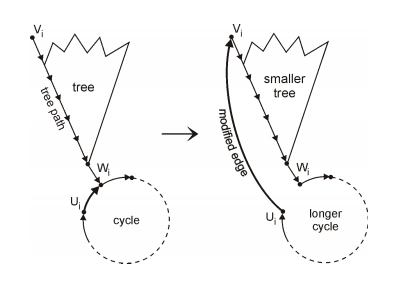


图3.打破循环增加循环长度

对于一个短周期的PRNG，我没增加周期长度的策略是在某个节点处切断周期通过修改状态转移函数使这个点的下一个状态转移到树深处的某一个节点。如图3所示，循环节点的输出边（）被修改，树节点变为的新后继，得到的循环长度是原循环长度加上到的树路径长度。

这可以针对具有显著大小的组件来做，甚至可以对一个组件进行多次操作，来得到循环长度更大的增加。由于预计的组件数量很小，而且仅有小部分具有显著大小，修改边的数量也会很小，并且可以存储在查找表中。PRNG的状态修改函数可以如算法1那样实现。

**算法1** 修改状态转移函数。

**前提：**s 是PRNG当前状态，TRANS 是原始的状态转移函数

**function** MODTRANSITION(s)

**if** s 是特别状态 **then** 访问查找表

= 访问查找表得到的s的新的后继

**else**

= TRANS(s)

**return**

注意，在查找表中花费的时间也会增加转化函数的执行时间，会导致功耗的增加。因此，我们将测试部分为了两个部分（大部分时间会失败，因为只有小部分的特殊状态）：一个非常快速的测试，但大多是情况会失败，以及另一个后续测试，它会进行精确的检查但很少执行。第一个测试使用易于测试的属性，例如状态表示的某些位具有某种模式。我们称这些节点为候选者。测试的唯一限制是转化函数改变的状态必须是候选者，但是没有严格的限制。

困难的任务是找到这些数量少的特殊状态和这些状态延伸出来的新的边。而且在增加周期长度的情况下不损害输出的统计特征以及其他属性。虽然这是个只需完成一次的离线任务，但是它需要对状态图进行采样，即如果平均路径长度是，选择的开始节点个数为，则可能需要执行次状态转移函数。为了在可接受的时间内完成，需要并行算法。

如果我们忽略所遇到的组件没有候选者这一情况，那么可以通过候选图的构造来选择特殊状态，即可从所选起始节点到达所有候选节点的图。候选图中的两个候选节点 ，之间的边表示状态图到 的唯一路径，并且使用该路径的长度进行标注。当计算候选图的时，可以使用DFS计算周期长度和树的深度，通过改变边缘使得周期长度增加最大化来选择特殊状态。重复执行操作直到候选图的连通分量的所有候选状态处于周期内或者实现最大数量的特殊转化。有关从候选图中确定特殊状态的详细信息参见[1]。

候选图的计算遵循一个简单的规则：跟踪源自起点的路径，如果两条路径相遇，则只跟随其中一条路径。当我们遍历路径时，我们会记录每个访问过的候选节点，这有了如下算法2.

**算法2** 计算候选图的并行算法

**前提：**S 起始节点集合，m =| S |，ISCAND 检测是否为候选状态

**function** COMPCGRAPH(S)

**for** i =0 to m //并行循环

**repeat**

**repeat**

()

**until** ISCAND()

ADDEDGE()

**until**  已经被其他路径访问

每个线程遵循一个候选者到另一个候选者的路径，然后检查该候选者是否已被其他线程访问过，如果是则该线程停止，另一个线程继续，如果不是则线程在下一步继续遵循此路径，如果俩线程在同一轮中到达候选者，则线程号小的继续。注意我们隐藏了一些细节。第一，构造候选图必须添加节点和带距离的边。第二，一个线程将会完成周期，并且遇到自己先前访问过的候选节点，这也必须检测到。最后，无法保证路径包含另一个候选者，因此，还需要其他的方法来检查是否已经达到一个周期。

并行算法将图中的每棵树划分为与线程遵循路径对应的链。链开始于，结束于一个被多个路径访问的候选点。换句话说，当开始于路径到达候选点时它已经被其他线程访问过了。图4用五个开始点说明了这个问题，图中开始于A和B的路径在候选点X相遇，随后仅遵循来自A的路径。同理，来自起始点C，D，E的路径在候选点Z相遇，随后遵循来自C的路径，直到在候选节点Y遇到来自A的路径，随后围绕循环。

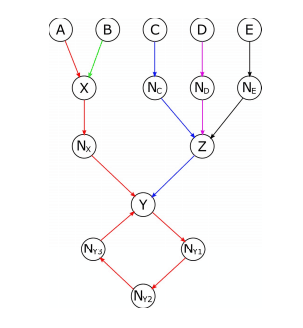


图4.路径的并行采样

算法的运行时间由最长的链决定，即从起始点出发在该组中找到的最长的候选点序列乘候选点之间的平均距离。如果存在n个可能的状态和c个候选状态，则候选状态间的平均距离是n/c，假设转化函数足够随机使得他们的在图中的分布类似随机分布。然而，标准差很高，所以在一轮中的最大距离可能远大于平均距离这导致线程负债不均衡和空转。为了避免这个问题，每个线程在每轮中处理几条路径，以便运行时间更接近于平均值。此外，这个问题出现很自然，因为需要处理的路径的数量大于可用线程的数量，即使考虑像GPU这样有着数千个硬件线程的大规模并行环境。之前在并行程序[15]中使用的每轮跟踪多个路径并随后查询被访问候选者的技术，尽管有着不同作用：通过捆绑多个查询以减少查询，消息传递机制中的高通信成本可以摊销。但这在本次研究中没起到作用，因为图表足够小，可以保存于共享内存中。但是在GPU中全局共享内存的访问很慢，所以不频繁的协调访问可以提高性能。

如果一段时间后跟随路径的数量变小，则可以执行负载平衡以尽可能地保持负载平衡。 一旦跟随路径的数量变得小于线程数，就必然发生负载不平衡。 如果最大链长和大多数链长之间的差异很大，则这种负载不平衡会遭到破坏。 如图4中的小例子所示，在两轮之后，仅留下五条链中的两条，并且在三轮之后仅剩下一条链，余下的一条链继续另外四轮。

如果这样的长链相对于其他路径进展得更快，则可以改善运行时。请注意，这是可能的，因为线程在每一轮中跟随几条路径，因此线程可以将t-2路径前进到下一个候选状态，而将一条路径前进到下两个候选状态。因此，后一条路径的进展速度是其他路径的两倍。由于线程现在只处理t-1条而不是t条路径，因此需要在线程上分配不同的路径，这没有问题。不幸的是，长链仅在算法结束时才知道，因此不清楚哪条路径会比其他路径进展得更快。

为了继续改善负载平衡，我们从自然界借用一个类比：当一只蚂蚁遇到其他蚂蚁的路径时，它往往会沿着这条路走，并且通过放置更多的信息素来加强这条路径。蚁群算法已经成功地用于解决与图论相关的问题，例如，在[16]。这里，如果路径遇到已经被另一条路径访问过的候选者，则它通过将其自己的时隙“捐赠”到另一条路径来加强该路径，这是可行的，因为不需要进一步跟踪第一条路径。因此，如果另一条路径是a 长链，它会前进得更快。虽然这种简单的启发式方法对于没有其他路径结束的长链没有帮助，但由于度为1的随机图中树木的粗糙结构，这种情况不太可能发生。显然，这种捐赠不能以线性的方式持续很长时间，因为每一个线程在一轮中最多遵循m / p路径，因此从超过m / p的其他路径获得捐赠的路径的加速将减慢一轮。由于在运行时不能确定最佳加速度，所以必须通过实验找出最有利的加速形式。此外，捐赠可以是传递性的，即从其他路径获得捐赠的路径，可将自己的时间连同从其他路径获得的时间捐赠给另一条路径。第四部分的实验将说明这种简单的策略足以将自然启发的类比转化为真正的优势。

实验

我们使用了一个模拟器来快速评估我们的并行算法于直接并行实现的路径加速的优势。模拟器读入我们之前在研究[1]中生成的候选图，然后逐个逐轮次的模拟线程。因为候选图结构已经可以使用，因此算法2中的检测是否为候选者的循环可以一部完成。因为后续节点之间的距离存储在图中，所以每一轮的执行时间（就每个线程调用TRANS的最大次数来说）可以被给出

模拟器应用于具有个随机选取的起始节点的路径的图，使用加密散列函数MD5作为转换函数并且将输出限制为64bit。候选集定义为节点集，其中第4位到第25位设置为1.

模拟器分别以100，500，1000个线程运行，以测试不同的m/p比率。我们使用两个简单的捐赠函数：捐赠时间段（最多m/p）的线性或对数。此外我们使用直接实施，不发生捐赠。我们还给出了顺序运行时间以便比较。注意，在运行时，我们指的是在算法期间线程遵循的从一个候选点到另一个候选点的边数。为了不模拟负载平衡，我们跳过了对函数TRANS调用的更精确的度量。

表1.不同线程数和捐赠策略的并行算法模拟运算

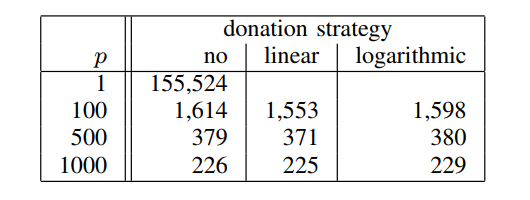


表1给出了不同配置的结果。我们可以看出对于100个线程，线性捐赠策略减小了运行时间的4%，这看起来很小但值得注意的是，并行算法中完成的轮次小于100轮，因此在时隙捐赠显示效果前需要一段时间。此外对于需要数小时顺序运行时间的算法，在并行算法中可以减少几分钟的运行时间。因为在捐赠效果显示之前已经处理了更多的边缘，所以对于较大的线程数，提升效果较小。对于对数捐赠策略来说仅在100个线程时有着小小的提升，其他情况略慢于没有捐赠策略的算法。所以应该选择线性的捐赠策略。我们还注意到我们的算法有很高的并行效率即使对于1000个线程来说任然有70%，这显示了对于GPU这样的大规模的并行计算引擎的可扩展性。虽然我们希望将并行效率表示为关于线程数p的函数，但这是不可能的，因为它还取决于状态图的结构。

我们还用逻辑映射作为一个混乱的PRNG例子测试了算法，其中a =3.99，x由双精度IEEE754-兼容算法获取，但是图形太小太平，它仅包含10000个起始点的20295个循环外的边。因此，无法观测到不同捐赠策略之间的运行时间的差异。但是算法的并行效率很高，对于100个线程有99%，1000个线程有88%

总结

我们提出了一种并行算法，用于寻找有希望的候选状态，以修改伪随机数发生器的状态转移函数。 使用这些候选者可以显着地增加周期长度，这对于那些因为资源限制无法增大自身状态空间的设备来说很有用，例如如嵌入式设备中的性能和能量限制。 该算法固有的负载平衡问题可以通过使用蚁群策略来解决：由于遇到另一条路径而未被进一步遵循的路径将时间贡献给可以更快地进展的另一条路径。 这再一次说明了大自然如何激发与安全相关的算法的改进。 得到的并行算法即使对于大线程数也表现出规则的结构和高效率，因此适用于像GPU这样的大规模并行计算引擎

我们未来的工作将包括进一步的实验，以微调时间段的“捐赠”来最大限度地提高性能，并将我们的模拟转移到GPU上的真实实现。在这种大规模并行处理设备上的性能也将能够处理更大的状态空间。 我们还希望将我们的工作扩展到例如流密码之类的类似的原语。Spritz [17]可能是一个很好的候选者，因为它也可以将我们的工作从非双射性扩展到双向过渡函数。

深度学习实施中的安全风险

**摘要：深度学习算法的巨大进步掩盖了其软件实现中的安全风险。本文揭露了流行的深度学习框架（Caffe，TensorFlow和Torch）中的一组安全漏洞。与深度学习模型的小代码量相反，这些深度学习框架很复杂，而且在很大程度上依赖于众多的开源软甲包。本文通过研究这些漏洞对语言识别，图片分类等常见深度学习应用的影响来说明漏洞带来的风险。通过利用这些框架的实现漏洞，攻击者可以发起拒绝服务攻击造成深度学习应用的崩溃或挂起，或者通过控制流劫持攻击导致系统受损或者认知逃避。本文的目的是提醒软件的实现和呼吁社区协作努力提高深度学习框架的安全性。**

介绍

近几年因为深度学习的成功应用AI成为人们关注的焦点。GPU，深度学习算法以及大规模数据集的发展使得深度学习算法能够解决许多领域的实际问题，从图像分类到医疗保健预测，从自动游戏到逆向工程，许多科学以及工程领域对深度学习有着极大的热情。

这些对新的机器学习算法的热情应用激发了多个深度学习框架的发展例如Caffe，TensorFlow和Torch。这些框架可以快速开发深度学习应用程序。框架为神经网络层提供了通用的构建模块，使用这些框架，开发者可以专心于模型的设计和应用特定的逻辑，而不必担心输入解析、矩阵乘法和GPU优化的编码细节。

本文中，我们研究了上述三种流行的深度学习框架的实现。我们根据框架发布的示例程序收集了他们的软件依赖项。这些框架的实现都很复杂（通常有数10w行代码），而且通常还建立在大量的第三方软件包上，例如图像和视频处理，以及科学计算库。

复杂的实现通常会导致软件漏洞，这是软件行业面临的共同挑战。深度学习框架同样面临着这样的问题，通过我们的研究，我们发现多个实现中的缺陷，其中我们发现的15项已经得到开发者的确认，并且已经获得CVE编号。缺陷的类型涵盖了多种常见类型的软件错误，包括堆溢出，整数溢出和使用后的释放。

我们对这些漏洞造成的威胁和风险进行了初步研究。对于构建在这些深度学习框架上的各种深度学习应用，我们尝试了一系列攻击，包括应用程序输入、训练数据和模型中的数据格式错误。这些漏洞潜在的后果包括拒绝服务攻击，逃避攻击和系统损害。本文简要的总结了这些漏洞以及我们预料的构建在这些框架上的深度学习应用的潜在的风险

通过我们对常见的深度学习框架的研究，我们做出了以下贡献：

* 本文介绍了深度学习应用的攻击面的研究
* 通过这篇文章，我们揭露了这些框架实现中的多个漏洞
* 我们表明了安全风险有可能出现在数据处理管道和深度学习模型本身
* 我们还研究了这些漏洞的影响和描述了建立在这些脆弱的框架上的应用的潜在安全风险

分层实施深度学习应用

深度学习框架可以实现机器学习应用的快速开发，深度学习框架配备了预先实现的神经网络层，使得开发人员可以专注于应用程序逻辑。开发人员可以在深度学习框架上设计，构建和训练特定于场景的模型，而无需担心输入解析、矩阵乘法和GPU优化的编码细节。

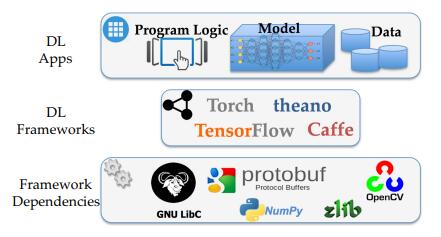


图1.深度学习应用分层方法

深度学习应用的实际实现各不相同，但基于深度学习框架的那些应用通常由3层软件组成，图1显示了典型的深度学习应用层。顶层包含了应用程序逻辑、深度学习模型和训练产生的相关数据。中间层是深度学习框架的应用，例如张量组件和各种过滤器。顶部两层之间的接口通常由实现中间层的编程语言指定。例如接口编程语言的选择为C++，Python，Lua分别对应Caffe，TensorFlow和Torch。底层是框架使用的构建模块，这些构建模块是完成例如音视频处理以及模型表示（protobuf）之类的任务的组件。构建模块的选择取决于框架的设计，例如，TensorFlow有自己的音频和图像处理实现，构建在第三方软件包（librosa，numpy）上，而Caffe选择直接使用开源库（OpenCV，Libjasper）来解析媒体输入。即使对于深度学习应用开发人员来说底层和中间层通常是透明的，这些组件已让是深度学习应用的基本部

分。

表1.深度学习框架以及依赖库

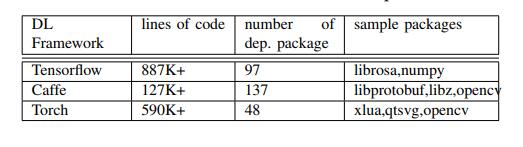


表1提供了一些关于深度学习框架实现的基本统计信息。在我们的研究中，我们分析的TensorFlow和Caffe版本为1.2.1和1.0。还包括Torch7。由于默认的Torch包仅支持有限的图片格式，我们选择研究结合OpenCV的Torch7，它支持各种图像格式例如bmp，gif和tiff。

我们通过代码行数和软件依赖包数量两个指标来衡量深度学习框架的复杂性。我们通过使用Linux上的cloc工具来计算代码行数。如表1所示，这些实现的代码库都不小。TensorFlow超过887k行代码，Caffe超过127k行代码，Torch超过590k行代码。此外他们都依赖于众多的第三方包，Caffe有超过130个依赖库（Linux ldd 测量），TensorFlow和Torch分别依赖于97个Python模块和48个Lua模块，上述结果通过import和require模块计数得到。

分层方式是软件工程常用的方法，分层不会直接引入风险，但是复杂性通常会增加漏洞风险。框架或者它的构建组件的任何缺陷都会影响到基于它的应用程序。本文接下来的部分将介绍一些关于实现缺陷的初步发现。

漏洞和威胁

虽然有很多关于深度学习和人工智能应用的讨论，但这些应用的安全性受到的关注较少。 为了说明与深度学习应用程序相关的风险和威胁，我们先介绍机器学习应用程序的攻击面，然后考虑实现漏洞导致的风险类型。

1. **攻击面**

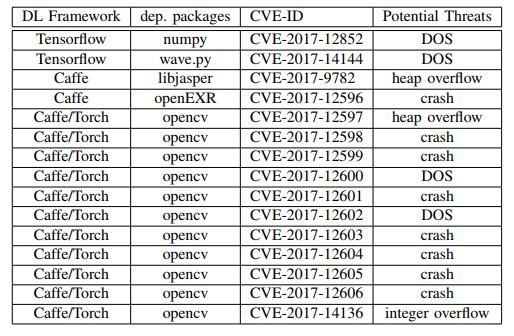
在不失一般性的情况下，我们以MNIST手写数字[20]识别为例来考虑深度学习应用程序的攻击面。 针对深度学习应用程序的攻击类型有很多种，包括DoS攻击，远程攻击，分类规避，模型反转攻击和成员资格推断攻击。 虽然这些攻击在攻击目标方面各不相同，但是针对深度学习应用程序（如MNIST）的攻击者发起点（攻击面）的来源主要来自以下三个角度：

* **异常的操作输入**：当前的许多深度学习应用一旦完成训练，通常就输入数据进行分类或识别。对于从文件或者为了读取输入的应用，攻击者可能会构造错误的输入。这适用于从文件读取输入的MNIST图像识别应用程序。对于从直接相连的相机或传感器获取输入的应用程序来说攻击面会显著减少。但是在这些情况下任然不会消除输入错误的风险，后文我们会有讨论
* **异常的训练数据**：如果训练数据来自外部源，则用于图像识别的训练示例可能会被污染或打上错误的标签。这也称为数据中毒攻击，数据中毒攻击[1,3]可能不依赖于软件漏洞，但是实现中的缺陷可能会使数据更容易中毒（或者更难检测中毒）。例如，我们发现框架和常见的桌面应用程序（图像查看器）中的图像解析过程不一致。这种不一致使得监控训练过程的人员无法注意到隐匿的数据污染
* **异常的模型：**如果开发人员使用他人开发的模型，深度学习应用也可能被攻击。尽管许多开发人员都从头开始设计和构建模型，但许多模型都可供没有足够机器学习知识的开发者使用。这种情况下，这些模型就成了可供攻击者操作的潜在的资源。与数据中毒类似，攻击者可以在不利用任何漏洞的情况下威胁那些使用外部模型的应用程序。但是，实现缺陷（例如模型解析代码中的漏洞）可以帮助攻击者隐藏异常的模型

注意，攻击面根据每个特定的应用程序有所不同，但我们认为这三种攻击面覆盖了攻击者威胁深度学习应用的大部分攻击空间。

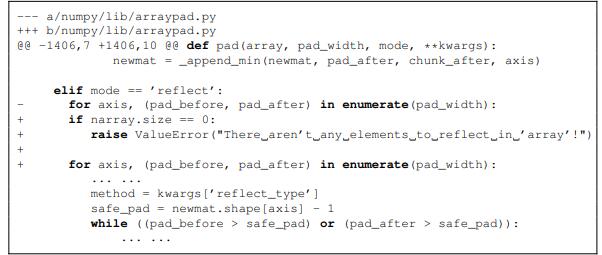
1. **威胁类型**

表2.DL框架和依赖项的CAVE

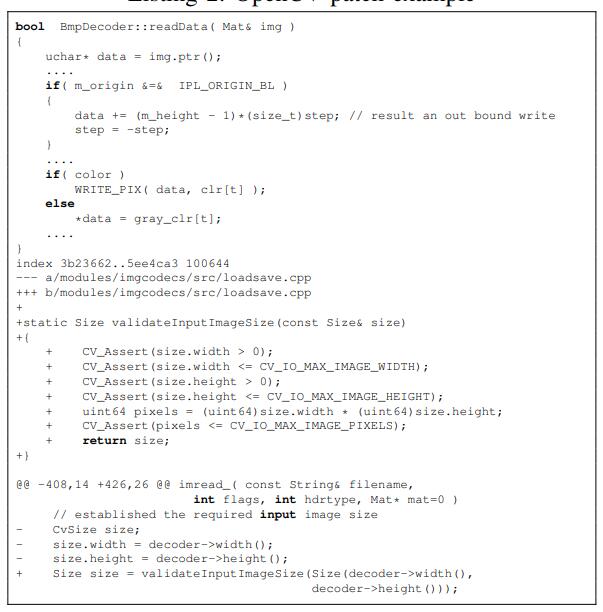


我们研究了几个深度学习框架并找到了超过10个新的实现缺陷，我们将这些缺陷报告给了开发者，所有的缺陷都被认定为新的bug并都已经被修复。表2总结了部分被分配了CVE编号的缺陷。这些实现缺陷使得应用程序容易受到各种各样的威胁，由于篇幅的限制我们只给出了异常输入造成的威胁。我们的分析基于应用从文件或网络获取输入这样的假设。

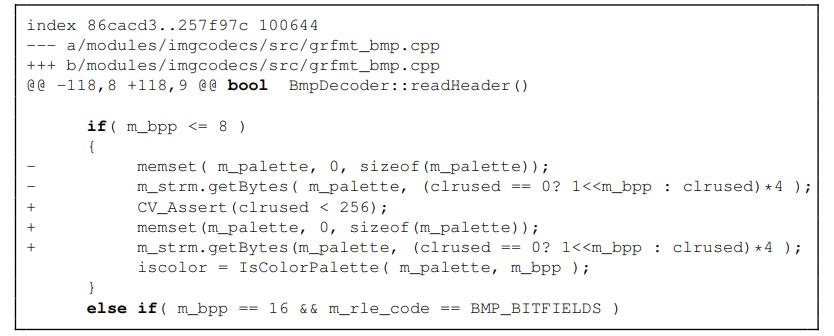
* **DoS攻击：**我们在深度学习框架中发现最常见的漏洞是导致程序崩溃，进入无限循环或耗尽内存的软件BUG。这写BUG导致的直接威胁是对运行在框架上的应用的拒绝服务攻击。下表显示了TensorFlow框架构建模块numpy Python包中发现的BUG的补丁。Numpy包用于矩阵乘法和相关处理，它通常被用于基于TensorFlow的应用。这个特殊的错误发生在包含一个不会因为开发人员意外的输入而终止的while循环的pad()函数里面。当调用者传入一个空向量时会引发这个缺陷，因为循环条件中的变量safe-pad会被设置为负值。由于这个BUG，我们展示了流行的样TensorFlow样本程序，例如Urban Sound Classification[15]，将悬挂在特制的声音文件中。



* **逃避攻击：**当攻击者构造应归类于特定类别但被深度学习应用错误的归类于不同类别的输入时，就会发生逃避攻击。机器学习研究人员花费了大量的研究工作得到通过对抗学习的方法产生逃避输入[8,10,19]。当面对脆弱的深度学习框架时，攻击者可以利用软件BUG来实现逃避的目标。我们在深度学习框架中发现多个内存损坏错误，这些错误可能导致应用程序产生错误的分类输出。攻击者可以通过以下两种方式来实现逃避攻击：1）通过漏洞对分类结果进行覆盖，使攻击者可以修改特定的内存内容。2）劫持应用程序控制流以跳过或重新排序模型执行。下面的表显示了一个越界写入漏洞和相应的补丁。在readData函数中数据指针可以设置为任何值，然后可以将特定的数据写入data指定地址，因此它可能会覆盖分类结果。



* **系统受损：**对于允许攻击者劫持控制流的软件BUG，攻击者可能会利用软件BUG并远程破坏托管着深度学习应用的系统。当深度学习应用作为一个从网络获取输入的云服务时会出现这类情况。下面列表显示了在OpenCV库中找到的简单的缓冲区溢出的补丁。OpenCV库是一个计算机视觉库，旨在提高计算效率，并且非常注重实时应用程序。OpenCV支持深度学习框架例如TensorFlow，Caffe，Torch/PyTorch。缓冲区溢出发生在grfmt\_bmp.cpp中的readHeader函数中。变量m\_palatte代表了一个大小为256\*4个字节的缓冲区。但是clrused的值取自输入图像，可以被攻击者设置为任意值。因此一个格式错误的BMP图像可能导致来自getBytes()函数的缓冲区溢出。通过我们的调查，这个漏洞可能导致任意的内存写入并且我们基于精心设计的图像输入成功强制示例程序（Caffe中的cpp\_classification）生成了一个远程shell。我们发现有另一组研究人员[17]同时也在研究机器学习应用程序的漏洞和层级影响。虽然他们为了系统攻击而探索OpenCV的想法与我们的努力有着相似的目标，但他们没有找到或发布OpenCV开发人员确认的漏洞[9]。相比之下，我们的研究结果已得到相应开发人员的证实，其中许多已根据我们的建议进行了修补。 此外，我们还开发了一个概念验证开发，利用我们团队发现的漏洞成功地证明了远程系统损害（通过远程获取shell）



开发样本

为了阐明深度学习框架中的软件安全问题的风险，我们精心设计了多个输入样例来演示各种威胁。以下的这些图像文件用于演示逃避攻击，本地和远程利用的威胁，表3显示了MD5校验和值和这些图像文件的注释。

表3.样例图片信息

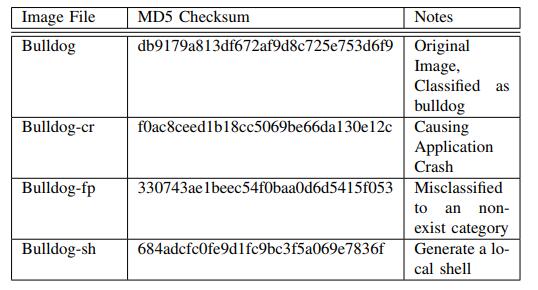


图2展示了这4种图片，左上角是从互联网上获取的原始图片，其他三个是导致流行深度学习应用发生故障的原始图片的变体。产生的效果包括拒绝服务攻击（此特定示例崩溃），逃避攻击（图像分类错误）和权限升级（获取本地或远程shell）

值得注意的是虽然原始图像（左上）和其他图像之间有轻微的视觉效果差异。图像失真是由图像元数据操作引起的（例如改变整数溢出的调色板数量）。并不是由图像数据本身的变化引起的。没有任何输入威胁是对图像特别的干扰结果，只是对抗学习[5,11,13,16]中常用的而已。

这些图片的细节和他们对流行的深度学习框架的影响见附录。

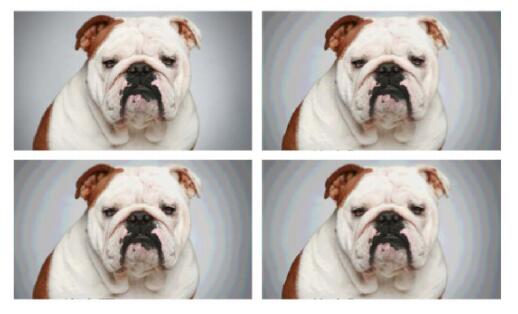


图2.原始图像（左上）和特定图像

讨论和未来工作

上一节介绍了深度学习框架实现上的软件漏洞，这些漏洞只是影响应用安全的其中的小部分因素[2,12]，还有其他例如应用输入源和训练数据格式需要考虑的因素。除非非常严格的检查输入数据并且训练数据的格式良好，否则会带来安全风险，我们在这里简要的讨论了一些相关问题。

1. **封闭环境中的应用程序安全风险：**许多样例深度学习应用程序都是被设计用在封闭环境中，这种环境下应用程序都是直接从和自身紧密联系的传感器获取输入。例如在相机上运行的机器学习实现仅获取相机内建的相机传感器生成的数据。可以说，风险相对于从网络或用户控制文件获取输入的应用更小。然而，紧密相连的传感器任然不能消除异常输入的威胁，例如存在传感器完整性被破坏的风险，如果传感器和运行在云服务器上的深度学习应用通信，攻击者可以反转通信协议直接攻击运行在云服务器上的后端程序。
2. **深度学习应用中的漏洞检测：**我们利用传统的BUG检测方式，特别是模糊测试，找到了本文中提到的软件漏洞。我们希望所有其他传统的静态或动态分析方式适用于深度学习框架的实现。但是，我们发现基于覆盖的模糊测试工具并不适用于深度学习应用，特别是发现模型执行中的错误。拿MNIST图片分类器作为例子来说，因为所有的输入都经过相同的计算层，所以几乎所有图片都覆盖一样的执行路径。因此，由于在这个案例中路径覆盖的反馈效果较差，基于覆盖的模糊测试工具不容易发现诸如被0除的简单错误。
3. **由逻辑错误和数据操作导致的安全风险：**我们的初步工作集中在导致程序崩溃，控制流劫持或拒绝服务的“常规”软件漏洞。考虑是否存特定于深度学习的BUG类型并且需要特殊的检测方式将会是有趣的。逃避攻击和数据中毒攻击不依赖于软件缺陷，例如内存损坏。如果应用程序的输入和深度学习模型的实际操作不匹配，就足以创造逃避攻击。检测深度学习应用的逻辑错误的另一个挑战是难以区分不充分的训练和有意的操作，后者的目的在于将一组特定输入错误分类。我们计划研究一种方法去检测这种错误。

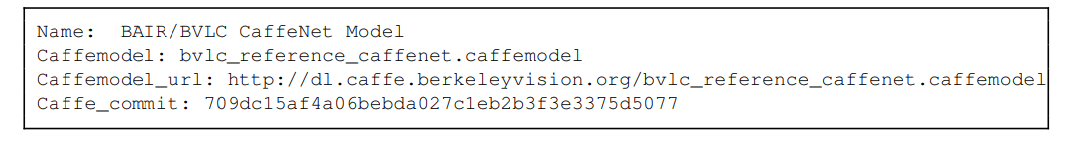
总结

这项工作的目的是为了提高软件实现错误引起的安全威胁的意识。深度学习框架是非常复杂的软件。因此，它有实现上的BUG几乎是不可避免的。这篇文章概述流行的深度学习框架实现上的漏洞和相应的风险。我们在深度学习框架和他们使用的库中发现了多个漏洞，潜在的风险类型包括拒绝服务、逃避检测和系统损坏。尽管封闭式应用在完全控制输入方面的风险较小，但仍不能完全规避这些攻击。考虑到深度学习应用程序的不透明性，这种应用程序掩盖了训练数据时的隐藏逻辑，导致实现缺陷引起的安全风险很难被检测到。我们希望我们在这篇文章中的初步研究可以提醒研究人员不要忽视常规威胁并且切实的寻找检测深度学习应用中软件缺陷的方式。

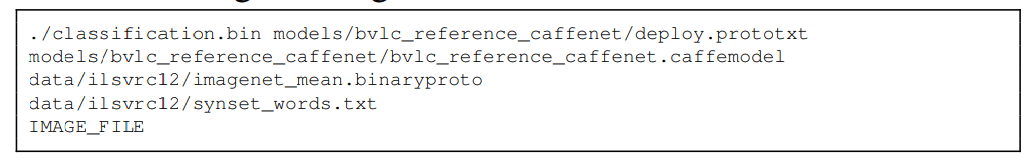
附录

这个附录提供了关于我们实验配置的简要信息。所有软件实现和深度学习模型都从网络上获取，我们的团队没有对模型和样例应用做任何修改。

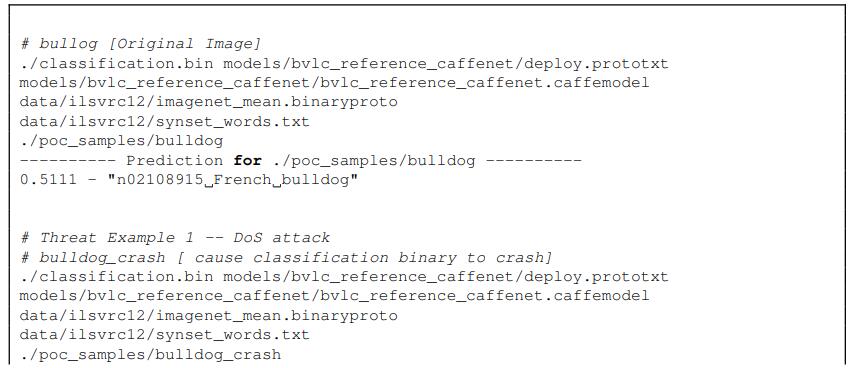
1. **软件版本和模型信息：**Caffe的软件包和相应的图片分类示例于2017年10月25号直接从官方的GitHub中签出。OpenCV使用的是来自以下URL的最新稳定版本https://github.com/opencv/opencv/archive/2.4.13.4.zip。我们在概念验证中使用了BVLC CaffeNet Model，该模型是基于原始的Caffe软件包中的说明提供的指导进行训练的结果。为了避免模型设置中的任何错误，我们直接从BVLC的官方GitHub页面下载了模型文件，有关模型的详细信息见下表。

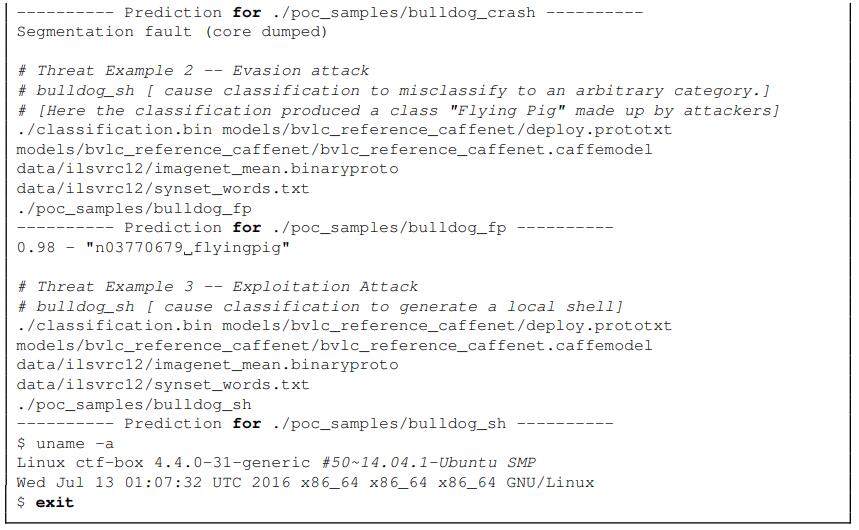


1. **命令行：**基于默认的Caffe示例CPPClassification演示了逃避和利用威胁。确切的命令行如下所示



1. **输出样例：**本文正文中提供的样本图像分类结果如下所示





一种快速检测未知格式的恶意Web内容的深度学习方法

**摘要：恶意网站内容是当今互联网上的一个严重问题，在本文中，我们提出了一种用于检测恶意网站页面的深度学习方法。相比以前关于Web内容检测的工作依赖于句法分析或模拟HTML和JavaScript来提取特征。我们的方法直接作用于利用简单正则表达式直接提取的静态HTML文件的语言无关的流上面。这使得它快到可以在防火墙和Web代理等高频数据环境中运行，并且允许它避免复杂的解析和仿真代码的攻击面暴露。与众所周知的方法不同（例如忽略空间信息的词袋模型），我们的神经网络在分层空间尺度上检查内容，这允许我们的模型捕捉局部性并且与词袋基线相比有更高的准确性。我们的提出的架构在0.1%的误报率下达到了97.5%的检测率，并且在消费级硬件上以100/s的速率对小批量网页进行分类。我们方法的准确性和速度使其适合部署到终端、防火墙、Web代理中。**

介绍

恶意Web内容是今天观察到的网络攻击的主要因素。这种有害内容分为两类。第一类涉及攻击者利用浏览器软件漏洞制作的在用户计算机上实现恶意目的网络内容。第二类是针对人类愚昧性的钓鱼网站，由诱使用户无意中泄露财务信息或登录凭据的网络内容组成。利用恶意内容的攻击在今天的网络上非常普遍。例如，在2017年12月的一周内，在威胁情报聚合器上超过100,000个以前未见过的恶意HTML文件被观测到。

检测和阻止此类内容存在多种挑战。首先，检测方法必须在用户端和防火墙中使用的消费级硬件上快速运行，以便于它们不会降低用户的浏览体验。其次，方法必须能够抵御恶意Web内容中的语法和语义变化，使得JavaScript混淆和文本随机化等对抗性逃避技术不会逃过检测。最后，检测方法必须能够在草垛中找到针如嵌入在其他良性网页内容中的表明页面是危险的小代码片段。这很重要，因为今天的许多网络攻击都通过良性网页的广告弹窗或评论内容等小组件传播。

为了应对这些挑战，本文提出了一种深度学习方法来检测恶意Web内容。我们的模型使用简单，快速，首先使用12个字符的正则表达式来标记Web内容，然后在多个分层空间尺度上检查此内容。 “分层空间尺度”在这里意味着我们不是简单地使用聚合的整个文档的标记内容作为输入，而是计算聚合的多个本地特定子区域上的表示：将文档分成二分之一，四分之一，八分之一和十六分之一。 然后我们应用两个密集层--我们称之为检查员网络--在所有这些聚合级别上去提取文档在多个空间尺度上的一个表示。

我们将我们提出的方法与许多基线进行比较，包括简单的词袋模型和更复杂的深层体系结构，并表明它以合理的计算成本实现了最佳结果。我们在时间上不相交的且以前未见过的内容上在0.1％的误报率下达到了超过97％的检测率，而且没有解析Web内容或模仿其行为的额外复杂性。实际上，我们的结果表明利用深度学习的恶意内容检测模型可以基于简单的令牌流输入来学习Web内容的高质量表示。

本文其余部分的结构如下。在第二节我们回顾了恶意HTML，JavaScript和URL检测领域的相关工作，以及深度学习和自然语言处理文献的相关工作。在第三节中，我们为层次架构选择奠定了基础的直觉，并描述了它在机械层面上的工作原理。随后，我们在第四节中描述我们的实验设置，详细介绍了我们运行的实验，以及提供我们的实验结果分析。

之前的工作

我们的工作涉及启发式Web内容检测，机器学习Web内容检测和深度学习文档分类领域的研究，下面我们将回顾这项工作。

Web检测的一个重点是仅使用URL字符串来检测恶意Web内容。[1]提出了一种用于检测恶意URL的机器学习系统。他们侧重于使用手动标注特征来最大化检测准确度。[2]也侧重于基于URL检测恶意Web内容，但是第一个使用基于手动特征工程的方法，第二个方法表明使用深度神经网络从原始数据中学习特征可以获得更好的性能。[3]使用URL作为检测信号，同时也包含其他信息，例如web链接中的URL引用，以提取他们提供的手工特征作为SVM和K近邻分类器的输入。

所有这些方法都和我们的工作有一个共同的目标，即检测恶意Web内容，但由于他们只关注于URL和相关信息，因此无法利用Web内容中的恶意语义。虽然基于URL的系统有着轻量级的优势，而且可以部署在完整Web内容不可用的环境中，我们工作重点是HTML文件，因为他们有更丰富的结构和更高的信息内容。由于这些方法使用的输入信息是正交的，因此基于HTML和基于URL的方法肯定有空间组成更有效的整体系统。

包括[4],[5],[6],[7]在内的一系列的工作尝试通过人工从HTML和JavaScript中提取特征并将其供给机器学习或者启发检测系统来检查恶意Web内容。[4]提出了一种方法，从页面的HTML和JavaScript的静态内容中提取各种特征，并将他们提供给机器学习算法。他们尝试了多种特征和学习算法的结合，并且比较了它们之间的相对优势。[5]避开了机器学习提出了基于HTML静态内容的手动定义启发来检测恶意HTML的方法。[6]也使用基于启发的系统，但他们使用JavaScript模拟器和HTML解释器来提取高质量的特征。相似的，[7]提出了一个内嵌JavaScript引擎的Web爬虫，用于JavaScript反混淆和分析来支持恶意Web内容检测。

我们在这里提出的方法与这些工作相似，因为我们侧重于HTML文件的详细分析，包括HTML，CSS，和内嵌的JavaScript。我们的工作不同之处在于，我们使用无解析器标记方法来计算HTML文件的表示，而不是明确的解析HTML，JavaScript或CSS，或者模拟JavaScript。Web内容的无解析器表示允许我们对恶意文档和良性文档做出一些语法和语义上的假设，从而允许我们的深度学习模型在学习Web内容的内部表示时有最大的灵活性。此外，这个方法最小化了攻击面暴露和复杂特征提取的计算开销以及仿真代码。

在Web内容检测文献之外，研究者在基于文档分类的深度学习领域做出了大量的贡献。例如，在一项值得注意的工作中，[8]展示了使用无监督（word2vec）和微调词嵌入的序列的一维卷积神经网络在上下文句子分类任务中相对于许多标准基线来说可以提供良好或一流的性能。[9]超越了这项工作，表明了直接从字符输入中学习表示的CNN在一系列文档分类问题上相对于其他文档分类算法有着更强的竞争力。相关的，[8]提出了一种将单词和字符级输入结合的句子情感分类模型。

我们的工作涉及到这些方法，因为我们的模型使用了一组密集的网络，这些网络在文件标记的多个细分上使用相同的参数，它可以解释为在文本上操作的卷积神经网络。我们的工作与上述工作的区别在于我们不仅仅作用于自然语言，我们的架构作用于HTML文档，实际上是HTML，JavaScript和CSS的混搭，其中包括任意源代码，攻击有效负载，和在这些格式中表达的自然语言。因为原始的HTML文档使得定义一个离散的标记词汇十分困难，所以我们不使用词内嵌作为我们的模型的输入，而是采用基于简单的，格式无关的标记化的Web文档的新颖层级表示作为模型输入。

此外，与过去基于深度学习的文档分类工作不同，我们不使用原始的字符序列作为我们模型的输入。因为典型的HTML文档的长度使得涉及对原始字符串的卷积或循环架构的推理在消费级的终端或防火墙上难以处理。我们的工作也与大多数文档分类工作不同，我们试图发现一个试图逃避检测的积极的对手（例如，在情感分类中，句子的作者并不视图规避对他们表达的情感的检测）

方法

1. **直觉与设计原理**

下面我们列出了一些激发了我们在本文中提出的模型的直觉。

1. 恶意网页通常在大量良性内容（例如，被黑客入侵的页面中的良性内容）中嵌入小部分恶意内容（例如恶意JavaScript）。因此，识别给定的文档是恶意的需要模型在在多个空间尺度上检查文档。这是因为JavaScript片段的大小范围很小，但是HTML文档长度的方差很大，这意味着表示恶意内容的文档长度的比例在示例中是可变的。
2. 明确解析HTML文档（实际上是HTML，JavaScript，CSS和原始数据的集合）是不可行的，因为他使实施变得很复杂，这可能要求高计算开销并且打开检测器自身内部的攻击面，这可能被攻击者利用。
3. 因为它带来的计算开销和它在检测器内部打开的攻击面，所以HTML文档中的JavaScript的仿真，静态分析或者符号执行是不可取的。

根据这些直觉，我们在创建我们提出的方法时做出了以下高级设计决策

1. 我们不是在HTML文档中执行详细的解析、静态分析、符号执行或内容模拟，而是计算了一个简单的词袋风格的文档标记，它对文档组成格式做了最小假设。
2. 我们使用一种表示方法来捕捉多空间尺度上的局部性，它表示了不同级别的本地化和聚合。而不是简单的使用一个平坦的，聚合整个文档的标记袋表示。这允许我们的模型在其他良性恶意内容中找到关键的恶意内容。
3. **方法**

我们的方法涉及一个特征提取器和一个神经网络模型，前者从HTML文档的中解析出一系列的标记，后者使用分层级别的聚合的特征的共享权重来做分类决策。这个神经网络包含两个逻辑组件：

1. *检测器*：它将分层空间尺度的权重和文档信息的聚合应用于一个1024长度的向量上
2. *主网络*：它对检测器网络的输出做最后的分类决策

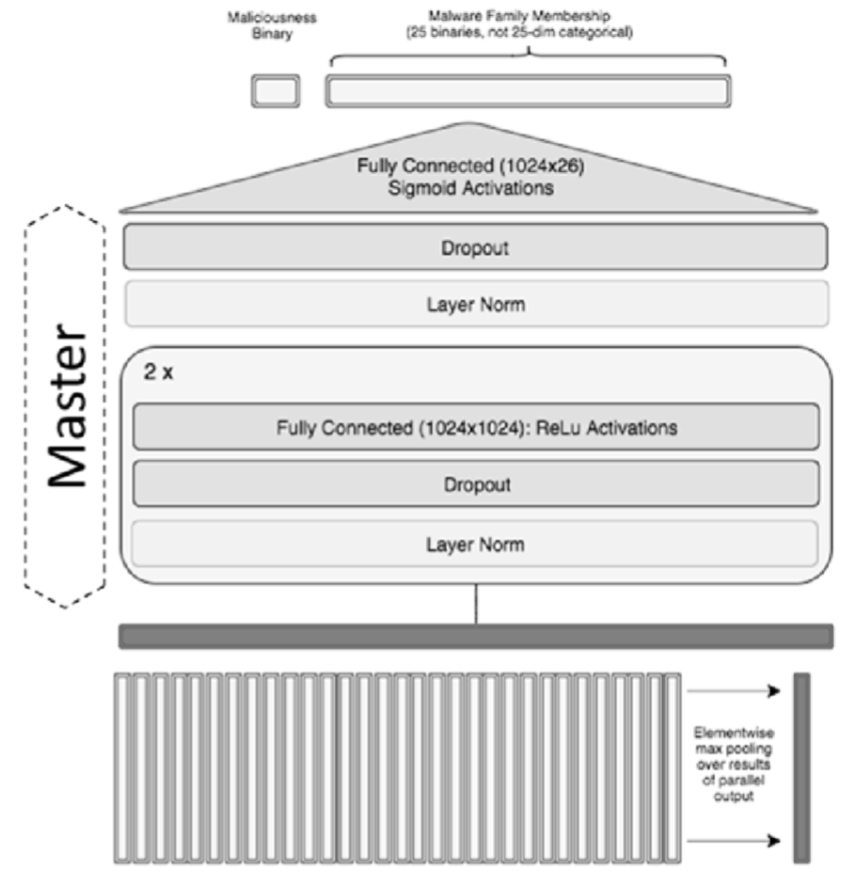
通过反向传播优化检测器和组要组件。下面我们将谈到每个逻辑组件是如何为我们整个系统做出贡献。

***特征提取****：*我们系统的工作流程开始于从HTML文件中提取一系列的字符串标记。首先，我们使用正则表达式（[^\x00-x7F]+|\w+）对目标文档进行标记，它将文档沿非字母数字的字符的边界分割。然后我们将标记流分成16个顺序等长的块——其中长度由标记的数量定义——如果文档中的标记的数量不能被16整除则余下的标记包含在最后一个块中。标记和分块函数的Python的参考实现在附录给出，函数名为TokenizeChunk。接下来，我们使用修改过的带有1024个分区的hash技巧，来为每个块创建一个词袋风格的表示。我们使用[10]中介绍的一种技术来修改分区的放置使它成为一个关于特征hash和标记长度的函数值（Python的参考实现在附录的TokenLengthHash）。这个工作流的结果我们将其标记然后16个等长的标记块，然后将每个标记块的特征散列到1024个分区中，然后组成一个16\*1024的张量表示从HTML文档中提取到的一系列标记袋，其中序列的每一个元素代表了输入文档的连续上的聚合。

***检测器：***一旦我们有了一个HTML文档的特征表示，我们将这个表示输入到如图1那样的神经网络中。如图所示，我们计算流程的第一步是对我们标记袋的序列创建一个分层表示，在哪里我们将最初的16个标记袋压缩至8个，8个压缩至4个，4个压缩至2个，2个压缩至1个，这样我们就可以获得多个空间尺度上捕获的标记表示。这个压缩过程的工作原理是将长度和步长为2的窗口平均放置在最初的16个标记袋上，然后递归执行，直到得到一个标记包。这里值得注意的是，对于标记计数进行平均，而不是求和，我们在给定文档中保持每个表示层级的范数相同。

一旦检测器完成了这种分层表示，他会继续访问聚合树的每一个节点并计算输出向量。如图1所示，检测器是一个前馈神经网络，有两个完全连接的层，每个层由1024个ReLU单元，我们使用层标准化[11]来避免梯度的丢弃和消失以便正规化检测器。我们使用0.2的丢弃率。为了计算检测器访问每个节点后的1024维的输出向量，我们从31个聚合块产生的31个输出中的1024个输出神经元中获取最大的激活。这使得文档的最终矢量表示是检查器的最后一层中每个神经元的最大输出，给定它在层次结构中的所有节点上的所有激活。 直观地说，这应该促进输出矢量捕获模式，这些模式最接近地匹配用于预测恶意内容的已知模板特征，无论它们出现在文档中的哪个位置，或整个文档的长度是多久。

***主层：***一旦检测器在目标文档上计算了其1024维输出向量，该向量就被输入到我们模型的主要组件中。如图1所示，主层被实现为带有两个逻辑全连接块的前馈神经网络，其中每个完全连接的层之前是层标准化和丢失。与检测器的情况一样，这里我们使用0.2的丢弃率。模型的最后一层由26个sigmoid单元组成，对应于我们对文档做出的26个检测决策。其中一个sigmoids致力于确定目标文档是恶意的还是良性的。其他25个sigmoids检测各种信息标签，例如是否该文档是网络钓鱼文档或漏洞利用工具包的实例。为了训练模型，我们在每个sigmoid输出上使用二进制交叉熵损失，然后平均得到的梯度以计算参数更新。在本文中，我们强调评估良好与不良sigmoid输出的准确性，也提及我们在下面这些其他输出上的表现



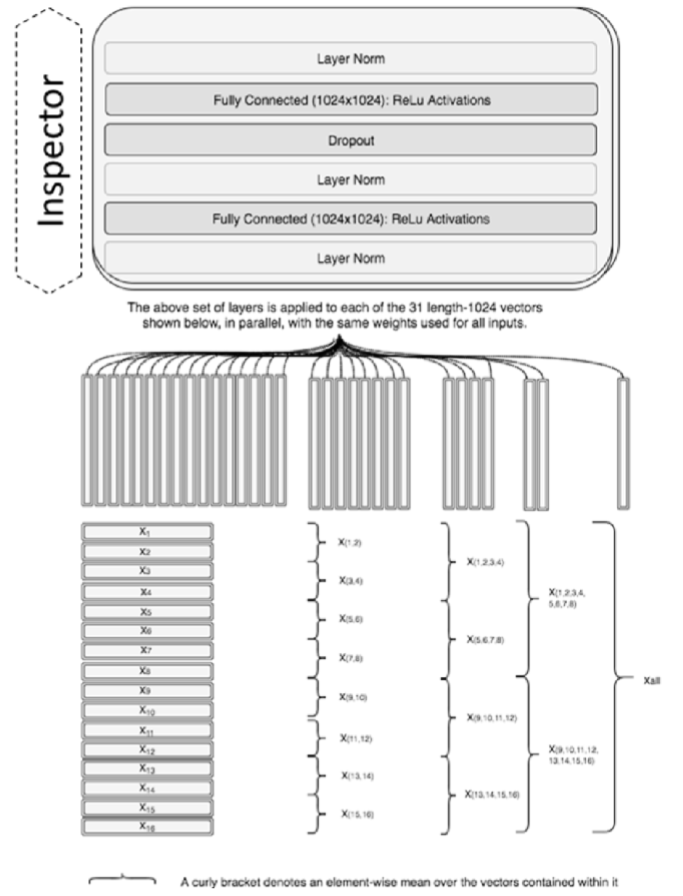


图1.分层检测器架构

评估

我们以两种方式测试了我们的方法。首先，我们将它与许多词袋样式基线进行了比较，这些基线代表了标准的文档分类方法。其次，我们以各种方式修改了架构，以测试我们的模型设计选择是否有助于提高准确性。 我们没有直接比较我们的方法和涉及复杂的Web内容解析或仿真的方法，因为这些方法的性能开销使它们超出了我们的研究目标的范围，我们的目的是为了创建一个快速的Web内容检测模型，这个模型可以对在防火墙和终端上观察到的Web内容进行操作。

下面我们描述我们的实验数据集，我们的基线，和我们的模型修改，然后介绍和讨论我们的结果。

**实验数据集和评估方法：**我们测试了从威胁情报站点VirusTotal收集的数据。VirusTotal每天接收数以万计的新HTML文件，使用来自数十家安全供应商的60个Web威胁扫描程序扫描它们。Sophos订阅了VirusTotal的付费威胁情报服务，作为此订阅的一部分，我们会收到提交给VirusTotal的每个HTML文件及其相应的扫描结果。本文中使用的实验数据集是从VirusTotal 中收集的2017年前10个月的数据，如图2所示

文件是基于SHA256唯一标识的，我们的训练/测试拆分是根据VirusTotal上首次报告文件的时间计算的。 这个过程确保 a）我们的训练和测试集是不同的（因为稍后提交的相同的HTML文件将被解释为重新提交该文件，我们会忽略它并使用期版本）b）我们的 训练和测试过程至少近似于现实世界的部署场景，下面将更详细地讨论。

我们的标记策略是使用扫描仪集成的对的给定文件的响应作为标记规则的输入，该标记规则从这些数据中导出二进制好标签和坏标签。具体而言，我们的标签规则将良性HTML文件定义为从安全供应商接收0检测的文件，恶意文件定义为接收3个及以上检测的文件。

此外，我们丢弃收到1或2个检测但未在我们的研究中使用它们的文件，因为我们认为小数量检测意味着安全行业仍不确定它们是恶意还是良性。 从图2中可以看出，这些不确定的文件占据了此时段的整体文件的小部分。

这种标记方法存在风险，即我们的方法将简单地记住安全供应商产品的认知，而不是学习真正新颖的检测功能来检测供应商社区可能错过的恶意软件。 在这项工作中，我们使用历史模拟程序对此进行测试，定义如下：

1. 我们使用时间t之前在VirusTotal上首次出现的Web内容文件训练我们的模型
2. 我们使用时间t之后两个月内首次出现在VirusTotal的Web内容文件和对应的最新的供应商标签来评估我们的模型

此评估程序减轻了评估我们检测供应商社区可能错过的恶意Web内容的能力的问题。这是因为我们测试了我们在训练中没有看到的文件，但安全供应商社区有时间通过检测规则和黑名单更新来检测，从而模拟未检测到恶意内容的问题。

换句话说，只要机器学习模型能够正确预测未来Web内容的标签，供应商社区就有时间将黑名单或写入检测规则，我们相信至少有间接证据表明我们的方法还能够检测供应商社区可能完全错过的恶意Web内容。

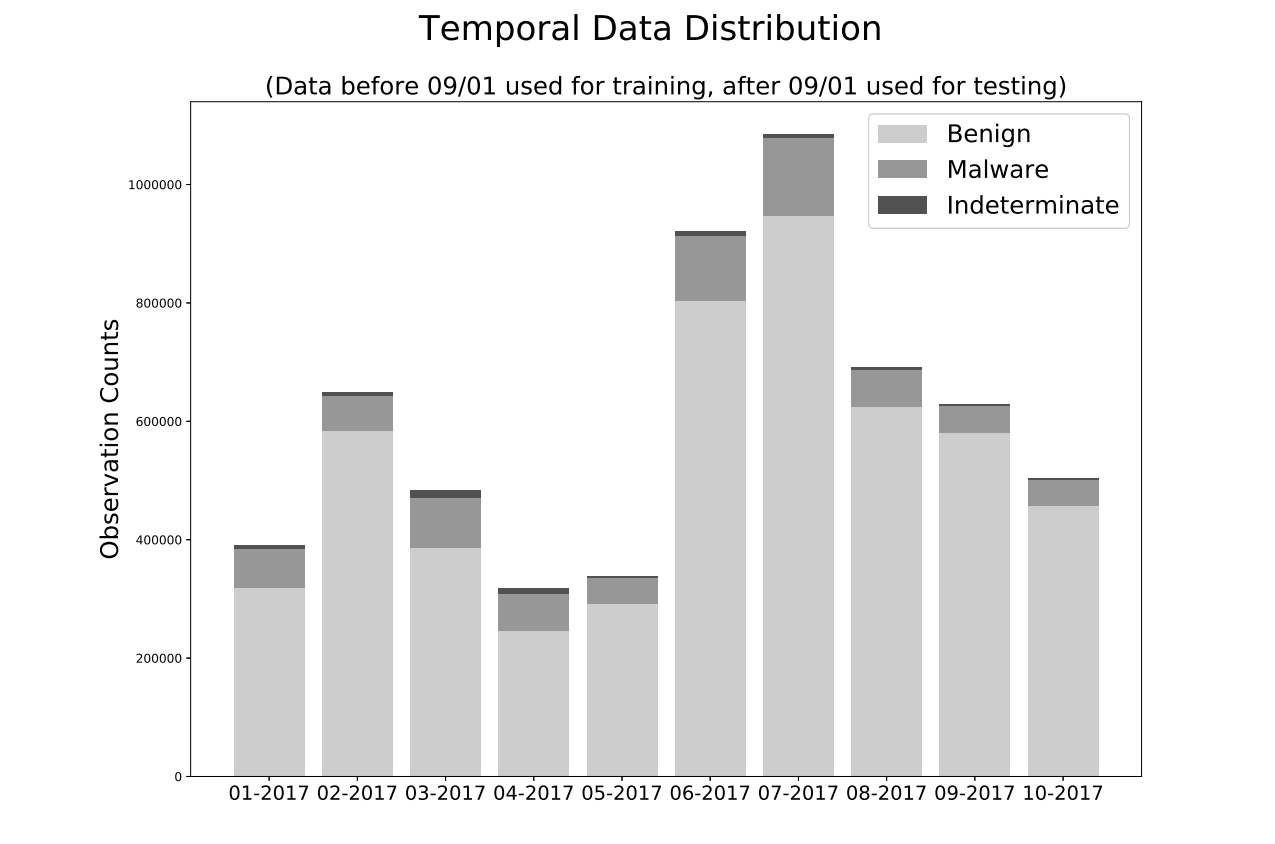
为了更直接地证明我们提出的方法的有效性，我们还亲自检查了我们的标记策略标记为良性，但我们提出的模型具有很高的恶意概率的样本，发现大多数这些假设的误报都是显然是恶意或垃圾内容。我们在下面讨论这些结果。

图2.训练和测试样本首次出现在VirusTotal上的时间直方图

实验

除了评估我们提出的模型外，我们还进行了五个额外的实验，其中三个测试我们的方法对替代模型的效果，其中两个探索我们模型的内部工作。 为了测试我们的模型架构的功效，我们进行了以下三个实验。 对于所有神经网络模型，使用Adam优化方案，64位大小的平衡批次以及基于验证集性能的早期停止来训练模型。 我们列举下面的测试方法。

**LR-BoT：**弹性网对标签标签包进行逻辑回归。在这里，我们使用我们为完整模型提取的标签，但特征将它们散列到16284长度的向量中。 我们选择在此基线实验和其他基线实验中使用16284输入，因为它与我们所提出的模型中使用的16x1024表示具有相同的维度，因此近似于一对一的比较。LR-BoT测试模型提供了一种简单的标签包线性文档分类方法作为比较点。 我们通过网格搜索确定了L1和L2罚分的λ1和λ2参数。

**FF-BoT：**使用16284长度特征的前馈架构，该特征是上面使用的散列标签包特征表示。 该测试模型提供了直接的深度学习词袋基线作为与我们所提出的方法的比较点

**XGBoost-BoT：**使用和FF-BoT相同的特征输入的梯度提升决策树（XGBoost）模型。

为了和我们的基线试验区分开，我们做了几项架构上的修改以确定我们在创建我们提出的模型时所做出的设计决策的贡献，列举如下：

1. **FlatSequential：**移除了平均池化步骤，使得检测器仅看到树的叶节点。换句话说，在输入冠军模型的31个聚合表示中，该模型只能看到16个连续的块，而没有更大的聚合窗口。此模型测试通过在分层空间尺度检查HTML文档来提供的性能增益 。
2. **FlattenedFF：**使用与我们选定模型相同的特征表示。但不在每一步都应用共享权重检测器，而是简单地将16x1024顺序标签包向量光栅化为单个16284长度的向量，并将其输入到前馈神经网络中。 相对于密集的第一层，此实验评估使用共享权重检测器的性能增益。

结果

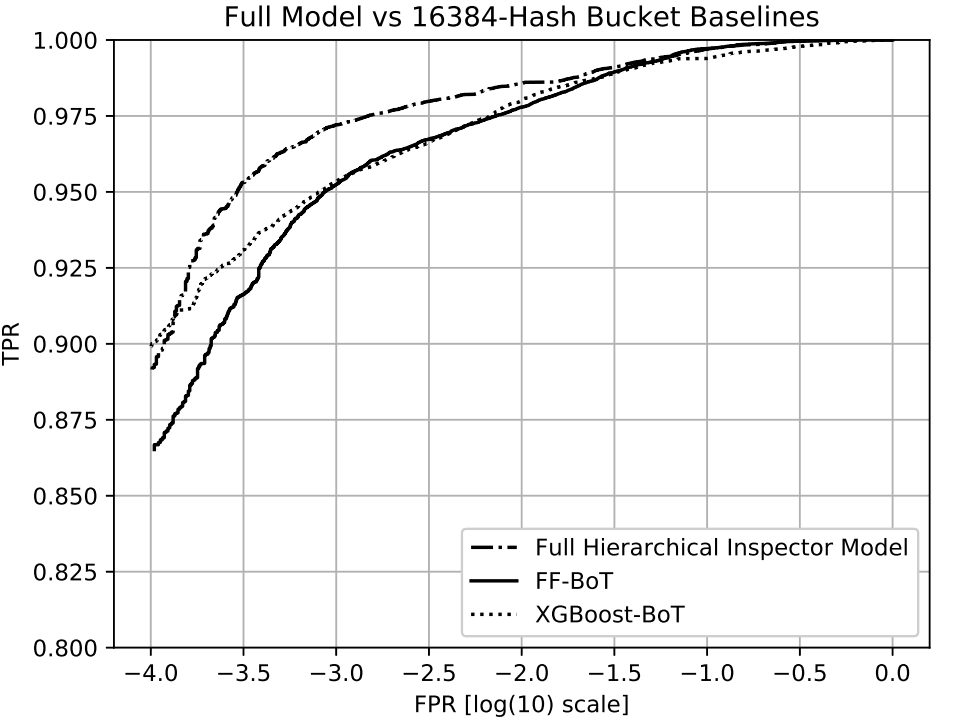


图3.我们模型与两个标签包基线的ROC曲线

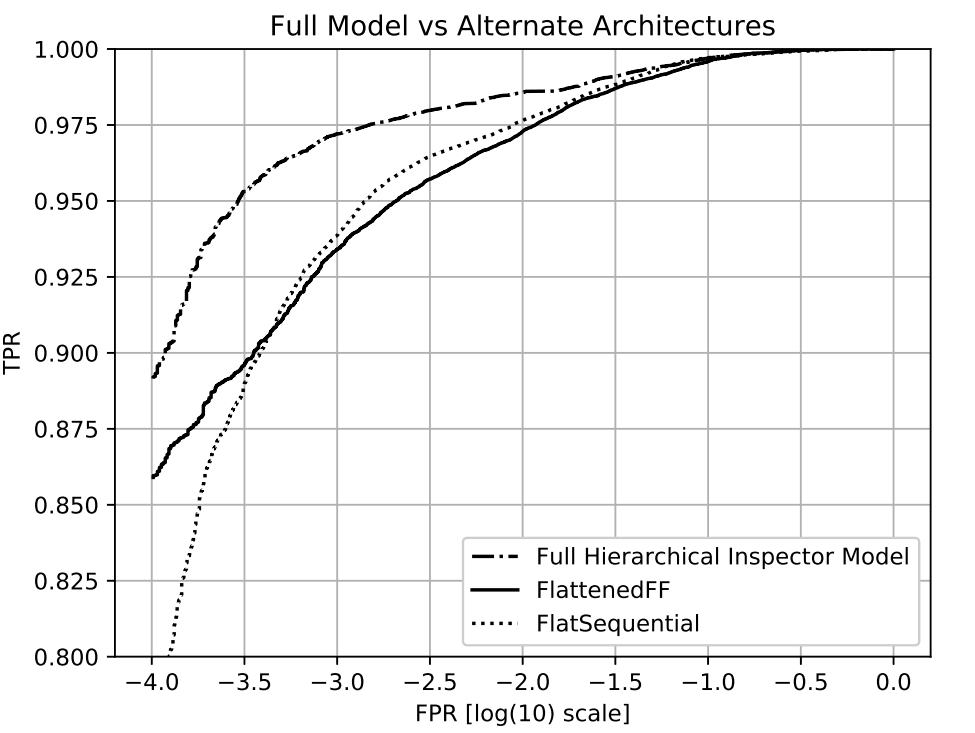


图4.我们的模型与其变种的ROC曲线

图3和图4给出了我们的实验结果作为ROC（接收器操作特性）曲线，其显示了当我们调整检测阈值时准确率（y轴）和误判率（x轴）之间的折衷。 图3将我们的基线FF-BoT和XGBoost-BoT与我们提出的方法进行了比较，图4将我们的架构修改与我们提出的方法进行了比较。 从这些图片中排除的是我们的线性基线实验LR-BoT，其实现了比我们的其余实验显著更差的结果（即，在0.1％误判率下的10％检测率）。

观察图3中的ROC曲线，我们发现我们提出的方法优于我们的基线模型。 如果我们以0.1％的固定误判率比较这些模型的相对性能，我们看到我们提出的方法，FF-BoT和XGBoost-BoT分别达到97.2％，95.2％和95.4％的检测率。 根据假阴性率，我们的方法实现了2.8％的假阴性率，而我们的基线假阴性率大约是4.8％和4.6％。值得注意的是，我们提出的模型的整体ROC曲线明显优于我们的基线。

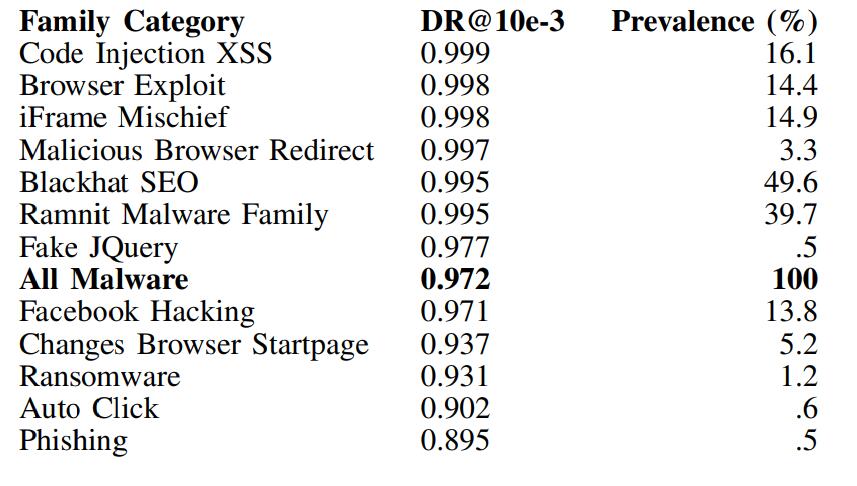
有趣的是，我们的FF-BoT基线相对于我们提出的模型表现不佳，其参数（约2000万）远远超过我们提出的模型（约400万）。这表明我们提出的方法捕获了更有效的恶意HTML文档表示，这要归功于我们的检测器在我们的层次表示中检查的每个空间上下文中使用相同的参数。

观察图4中的ROC曲线，我们看到应用前馈块的分层检测器以及我们的标签层次结构中每个节点的共享权重的方法，优于此体系结构的变体。FlattenedFF，它使用每个块的单独权重检查我们的顺序标签包表示达到了93.4%的检测率。FlatSequential与我们提出的模型相同，但没有平均池化步骤，检测率也达到93.4％。此外，测试模型的总体ROC曲线明显差于我们提出的方法。

我们提出的方法胜过FlatSequential这一事实很有意思，因为它表明在多个空间尺度上检查内容对于获得良好的准确性至关重要。类似地，我们击败FlattenedFF的方法表明，我们的检测器在检查每个空间背景和尺度时使用相同的参数对于产生高检测精度至关重要，因为FlattenedFF对每个空间背景使用单独的权重但是结果相对更差。

为了更好地理解我们的模型学到了什么，我们采用了恶意软件系列标签——之前用作辅助目标——并根据附加到哪些标签来细分真实的恶意软件样本。 基于对应于10e-3全局假阳性率的阈值，我们计算了总体检测率——97.2％——并将其与作为我们每个最普遍系列标记的恶意软件的检测率进行比较。 表1中显示的这种比较表明，我们的最终模型在代码注入，浏览器漏洞利用和iFrame操作攻击方面取得了最大的成功，但在钓鱼网站上遇到了最大的麻烦。关于患病率表中百分比的说明：这些标签并不相互排斥，这就是为什么患病率不等于100。

表1.针对不同恶意软件系列的检测率，以及出现标记的恶意样本百分比



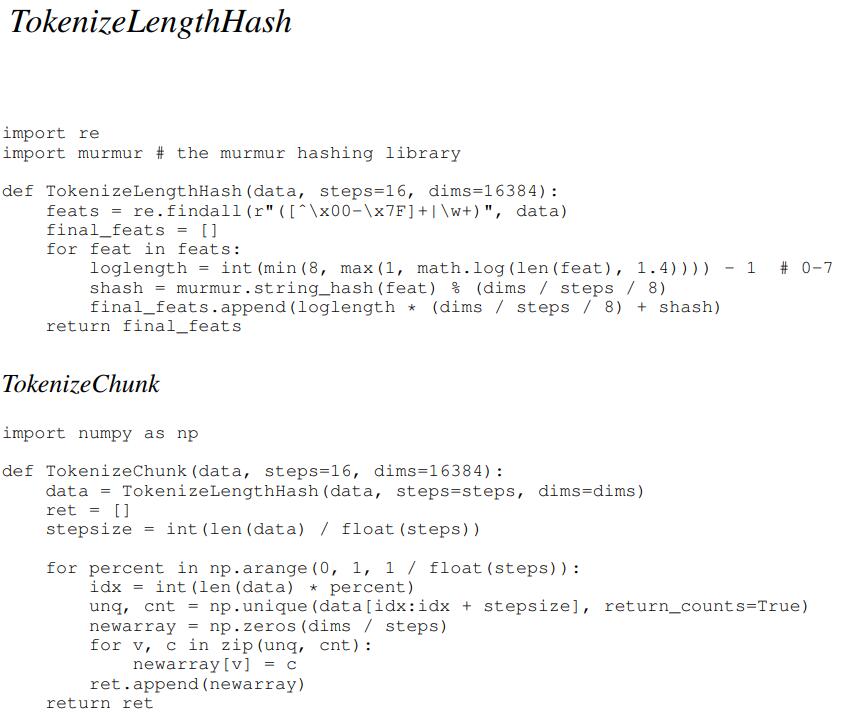
除了大规模验证之外，我们还检查了我们的模型是否与供应商的标签不一致的情况证明我们的模型实际上已经检测到安全社区遗漏的未知恶意网络内容。为了进行这项分析，我们检查了供应商社区一致标记为良性的验证集中前20个的最高得分测试示例，并发现其中13个实际上是恶意的或可能不需要的，其中7个是误报。我们发现的例子中有恶意，其中3个是来自网页内容拦截器的警报页面，不是恶意本身，但是表示恶意，3个是恶意假JQuery库，1个是向磁盘上投放虚假svchost.exe文件的Javascript，1个是 一个Facebook点击劫持页面，1个包含执行逐个下载的代码，2个是伟哥垃圾邮件。该分析验证了我们的模型能够超越标签噪声并识别以前未识别的恶意内容。

总结

最近成功应用深度学习的一个一致主题是使用域的已知结构特征（例如基于位置的特征和图像情况下的平移不变性）的价值在那个已知结构的方向上来引导和约束模型的学习能力。虽然本文的领域更专业，但它继续使用领域知识提供分层空间缩放的有用归纳偏差的相同传统，我们相信这使我们的模型更有效地处理可能大小不一的文档中的检测问题。在本文中，我们使用纯粹基于标签的静态方法以0.1％的误报率实现97.5％检测的强大性能，甚至可以识别以前未被供应商社区捕获的恶意内容，还避免了对复杂的解析或仿真系统的需求。这一结果使我们更有信心深度学习系统可以学习高质量的相当原始的Web内容输入的内部表示，其性能优于手工制作的特征。更广泛地说，深度学习方法在检测恶意Web内容方面具有广阔的前景。

附录

在这里，我们提供了函数的简短Python参考实现，这些函数生成我们提出的体系结构的输入，包括TokenizeChunk，它标记目标Web文档并将其拆分为块，以及TokenLengthHash，它为每个块使用散列技巧的修改版本创建特征向量。



第四章 论文总结