FirmPot: A Framework for Intelligent-Interaction Honeypots Using Firmware of IoT

笔记本： 学术论文
创建时间： 2022/3/10 15:28 **更新时间：** 2022/3/13 10:50
作者： 1936069022@qq.com
标签： FirmPot: A Framework for Intelligent-Interaction Honeypots Using Firmware...

FirmPot: A Framework for Intelligent-Interaction Honeypots Using Firmware of IoT Devices

FirmPot：使用物联网设备固件的智能交互蜜罐框架

摘要：模仿物联网设备行为进行威胁分析的物联网蜜罐正变得越来越重要。现有的蜜罐系统使用安装了特定版本固件的设备来监控网络攻击。但是，蜜罐经常收到针对与自身不同的设备和固件的请求。当蜜罐对此类请求返回错误响应时，攻击终止，监控失败。

为了解决这个问题，我们引入了 FirmPot，一个使用固件自动生成智能交互蜜罐的框架。该框架有一个针对蜜罐生成优化的固件模拟器，并通过机器学习来学习嵌入式应用程序的行为。生成的蜜罐通过一种机制继续与攻击者交互，该机制从对攻击请求的模拟响应中返回最佳响应，而不是错误响应。

我们在基于开源 OpenWrt 的无线路由器的嵌入式 Web 应用程序上进行了实验。结果，我们的框架生成了模仿八家供应商和十种不同 CPU 架构的嵌入式 Web 应用程序的蜜罐。此外，与现有的相比，我们的交互方法改进了与攻击者的会话长度。

索引词——物联网蜜罐、固件、机器学习

一、引言

物联网设备在增加功能方面取得了重大进展。例如，物联网设备的固件通常嵌入网络应用程序，允许用户从互联网上检查设备的状态 [1]。嵌入式 Web 应用程序具有特定于供应商的实现。因此，通过静态或动态分析发现它们的漏洞具有挑战性，并且它们在没有任何安全措施的情况下使用 [2]。

为了解决物联网设备的安全问题，研究人员重点分析了使用蜜罐的攻击方法。蜜罐通过伪装成合法设备来监控攻击者的入侵。如果攻击者利用零日漏洞，研究人员可以为物联网设备的开发人员和用户传播信息。

现有的蜜罐系统使用安装了特定版本固件的物联网设备。但是，漏洞的存在取决于固件版本。原因是固件更新部分修改了实现。实施中的修改可能会消除现有的漏洞，也可能会引入新的漏洞。

知道了这个特性，攻击者会在执行漏洞代码之前检查设备的固件版本。一旦攻击者确认蜜罐不是目标系统，攻击就终止。
我们的目标是继续与攻击者互动，以有效地观察针对物联网设备的攻击。为了实现我们的目标，蜜罐必须应对攻击者的系统调查。但是，一些系统检查很简单，例如验证目标系统的响应不是错误。因此，一个蜜罐对接收到的请求返回一个正确的响应而不是一个错误可能会绕过他们的检查。
我们提出了 FirmPot，这是一个通过模拟固件来生成智能交互蜜罐的框架，用于监控针对物联网设备的攻击。 FirmPot 通过模拟在 docker 容器上启动的固件来收集 Web 交互，并通过机器学习来学习请求和响应的对应关系。此外，生成的蜜罐会从模拟的响应中返回对接收到的请求的最佳响应。使用这种机制，即使在模拟阶段没有收集到请求，我们的蜜罐也不会返回错误响应，从而增加了攻击继续进行的可能性。
我们实现了该框架并在基于开源 OpenWrt [3] 的无线路由器上进行了实验。结果表明，该框架可以自动生成模仿八家供应商和十种 CPU 架构的 Web 应用程序的蜜罐。此外，框架生成的蜜罐被放置在互联网上，并捕获了攻击者对界面的操纵，例如登录尝试。此外，与现有方法相比，我们的交互方法改善了与攻击者的会话长度。
本研究的贡献如下：
1.我们找到了一种基于OpenWrt的固件仿真方案，适合蜜罐生成。
2. 我们提出了一种使用机器学习与攻击者进行有效交互的方法。
3. 我们建立了一个框架，只需提交固件即可自动生成蜜罐并发布源代码，以支持未来的研究1。
本文的其余部分结构如下：第 2 节介绍了物联网蜜罐的相关工作，第 3 节提供了研究背景。接下来，我们在第 4 节中介绍我们的框架。在第 5 节中，我们报告了我们的评估结果，在第 6 节中，我们讨论了我们的结果和未来的工作。第 7 节解释了伦理考虑，第 8 节总结了本文。
二、相关工作
物联网蜜罐是一种对物联网设备进行威胁分析的技术。它们主要根据交互程度分为两类：高交互蜜罐和低交互蜜罐。除了这两个类别之外，本节还介绍了另一个交互级别，智能交互蜜罐。
A. 高交互蜜罐
高交互蜜罐通过提供攻击者完全控制的系统来收集网络攻击的高级信息。此类物联网蜜罐包括将物理设备部署为蜜罐的 SIPHON [4] 和公开在虚拟环境中运行的固件映像的 Honware [5]。

高交互的问题在于，随着蜜罐数量的增加，可扩展性会降低。原因是物理设备和虚拟机消耗计算资源。此外，高交互蜜罐存在漏洞利用的风险，例如 Honware 陷入 DDoS 攻击 [5]。为了防止蜜罐被利用，用户必须定期执行繁重的任务，例如状态监控和系统刷新。

B. 低交互蜜罐

低交互蜜罐包括模拟单一协议的蜜罐，例如 U-PoT [6]，以及模拟特定设备的蜜罐，例如 ThingPot [7]。另一方面，一些蜜罐扫描互联网以模拟各种物联网设备的多种协议[8]、[9]。

低交互蜜罐只支持系统的部分功能，不支持整个系统。例如，很少有蜜罐具备物联网设备提供的认证功能。这个问题源于缺乏适用于不同设备类型和供应商实现的物联网设备的通用仿真方法。此外，由于蜜罐在接收非模拟请求时的固定行为，攻击者很容易检测到蜜罐。

C. 智能交互蜜罐

智能交互蜜罐已被 IoTCandyJar（罗等人的工作）添加到交互级别。 [8]。智能交互蜜罐的概念是与攻击者进行交互，以最大限度地提高捕获攻击的可能性，而不是像在高交互或低交互中那样准确地模拟特定设备的行为。

IoTCandyJar 从互联网扫描收集的 IoT 设备的众多响应中选择攻击者的期望。如果选择了预期的响应，则攻击者假设蜜罐是他们的目标并发送恶意命令。 IoTCandyJar 使用马尔科夫决策模型从头开始学习攻击者期望的响应。因此，他们的蜜罐需要一些时间才能适当地响应请求。结果，该模型需要两周的时间来学习足够的知识以持续与攻击者交互。

三、背景和动机

本节解释了我们框架中的挑战。

A. 固件仿真的挑战

我们框架的第一步是模拟物联网设备的固件。我们的目标固件由基于 Linux 的内核和文件系统组成。存储在文件系统中的应用程序的特点是可以访问硬件。出于这个原因，QEMU 经常用于模拟 CPU 和外围设备。

QEMU 有两种方法：全系统仿真和用户模式仿真。大多数现有的固件仿真器使用全系统仿真，它仿真整个系统，包括内核。典型的仿真器 Firmadyne [10] 通过引导过程启动固件的每个应用程序。然而，Firmadyne 的仿真成功率较低，而 Honware [5] 和 FirmAE [11] 使用经验方法对其进行改进。

这些仿真器通常针对 MIPS big-endian、MIPS little-endian 和 ARM little-endian。但是，我们发现了 Pow3erPC 和 MIPS64 固件映像，如第 V-A 节所示，因此这些仿真器不太通用。此外，郑等人。发现全系统仿真比用户模式慢大约十倍，只

仿真目标程序[12]。但是，用户模式仿真无法访问内核函数。因此，我们的第一个挑战是在蜜罐生成中找到速度和功能的最佳仿真。

B. 网络仿真的挑战

嵌入式 Web 应用程序是一种用于管理设备的服务。它提供静态内容（如 HTML）和脚本生成的动态内容。因此，攻击者将仅具有静态内容的 Web 服务视为诱饵系统。为了再现网络的本质，Musch 等人。 [13]通过爬行收集静态内容，通过模糊测试收集动态内容，这会故意改变响应。然而，他们只是模仿了登录前的页面。大多数针对嵌入式 Web 应用程序的攻击都针对登录后可访问的配置页面。因此，没有基于登录界面的蜜罐在观察到的攻击中受到限制。综上所述，我们的第二个挑战是模仿嵌入式 Web 应用程序的所有页面和功能。

C. 智能交互的挑战

当今的许多攻击都是由自动化脚本执行的。这些脚本具有针对众所周知的漏洞的预定义路径和利用代码。因此，模仿特定设备的蜜罐与受到攻击的设备不匹配，并且经常对接收到的请求返回错误响应。在收到来自目标系统的错误响应后，攻击者确定它不是易受攻击的并终止攻击。

但是，一些攻击脚本会根据响应中包含的简单信息来确定是否继续攻击。例如，TP-Link 固件的概念验证代码 23 通过简单地检查状态代码是否为“200”以及正文是否包含“成功”等词来确定设备类型。这个例子表明攻击者期望一个成功的响应并通过接收它来继续攻击。

已经提出了几种方法来分析接收到的请求并智能地响应以处理此类情况。小等人。 [14] 建议通过自然语言处理生成对接收到的请求的响应。但是，它不适合生成相当大的供应商相关响应，例如嵌入式 Web 应用程序发出的响应。八木等人。 [15] 定义了一种算法，将请求的路径转换为蜜罐的路径结构。但是，他们的方法没有考虑查询参数和请求正文。我们的第三个挑战是准备一种机制来返回基于接收到的请求中的全部信息继续攻击的最佳响应。

四。固件

本节提出了 FirmPot，这是一个自动生成模仿嵌入式 Web 应用程序行为的智能交互蜜罐的框架。 FirmPot 的概念不是完全像实际系统那样工作，而是从预先模拟的响应中返回最好的响应接收到的请求。它基于响应中的简单信息触发攻击的事实。在我们的实验中，我们选择了无线路由器的固件映像。

A. 高级概述

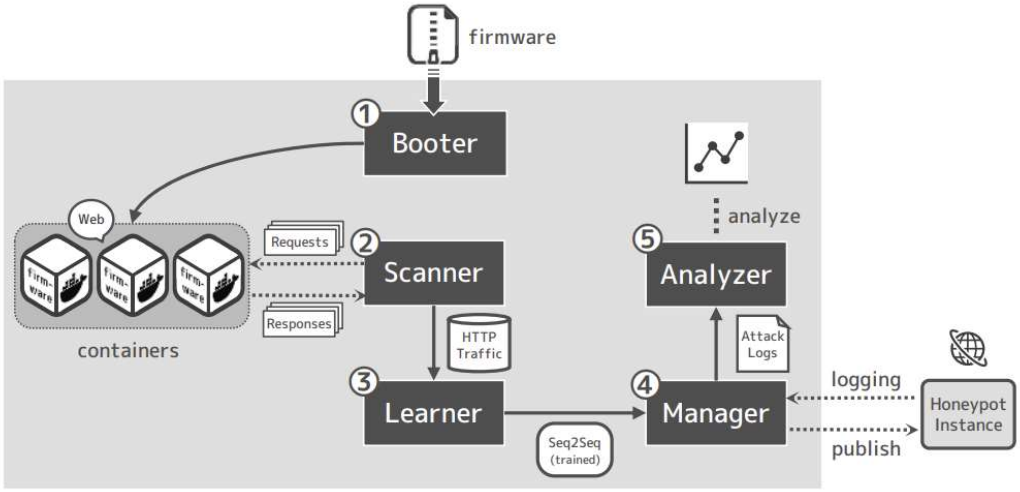

Fig. 1. A high-level overview of FirmPot.

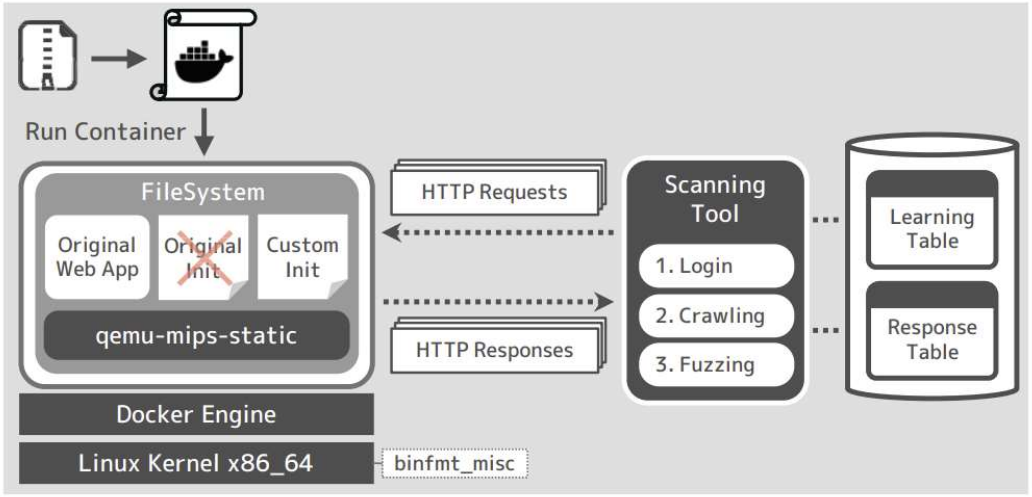

Fig. 2. The left half shows the structure of the container created by the booter, and the right half shows the collection of web communication by the scanner.

图 1 显示了整体框架。用户准备好固件镜像并提交给框架，然后框架自动开始生成和运行蜜罐。该框架有五个组件：引导程序、扫描程序、学习程序、管理程序和分析程序。引导程序、扫描程序和学习器模块用于模拟固件并学习其行为。 manager 模块负责准备一个蜜罐实例并将其发布到 Internet。此外，管理器监控蜜罐的中断并定期将攻击日志收集到本地机器。分析器模块负责分析从管理器接收到的日志。例如，它输出接收请求的统计信息以及与客户端的会话长度。分析仪的分析结果显示在第 V-C 部分。

B. 引导者

引导程序为我们的第一个挑战提供了蜜罐生成中的最佳模拟器。该模块的目标是在 docker 容器上运行嵌入在固件中的 Web 应用程序。 docker 容器是一个轻量级的虚拟环境，其中网络和文件系统与主机分离，并由指定配置和启动行为的 docker 映像创建。我们的框架创建了一个 docker 镜像，其中包含从固件镜像

和 QEMU 用户模式模拟器中提取的文件系统，如图 2 的左半部分所示。在 Linux 内核中使用 binfmt misc 启动 QEMU 用户模式模拟器取决于二进制格式允许容器在与主机不同的 CPU 架构上运行可执行文件。
要启动的容器从 docker 网络接收其 IP 地址，并执行引导程序模块将为每个固件映像创建的自定义初始化脚本。此自定义脚本遵循 OpenWrt 项目提供的开源 Linux 发行版的启动过程4。在OpenWrt启动过程中，先启动/sbin/init，然后调用/etc/init.d/rcS脚本。 /etc/init.d/rcS 执行 /etc/rc.d/ 目录下的各种供应商创建的脚本，例如密码初始化、TLS 配置和 Web 应用程序启动。但是，执行所有这些脚本会使系统崩溃。原因是其中一些脚本访问用户模式模拟器无法处理的网络接口和硬件。因此，我们的自定义脚本不会运行包含与网络或外围设备相关的词的脚本，例如“wifi”、“switch”、“led”等。这种技术允许我们在不访问网络设置或外围设备的情况下启动 Web 应用程序，从而避免崩溃。
由于此方法基于 OpenWrt 规范，因此似乎只支持部分固件。但是，很多厂商都使用 OpenWrt 进行开发，我们发现我们的模拟器可以运行各种固件镜像。我们模拟器的另一个优点是能够从单个 docker 映像启动多个容器。在下一个扫描阶段，我们通过并行化容器减少了蜜罐生成所需的时间（参见第 V-B 节）。
C. 扫描仪
扫描仪解决了我们模仿整个 Web 应用程序的第二个挑战。该模块负责获取模拟行为所需的各种请求和响应之间的对应关系。扫描仪包含一个扫描工具和一个数据库，如图2右半部分所示。该扫描工具包含三个组件：用于验证用户身份的登录组件、用于探索静态页面的爬虫组件以及用于获取动态变化响应的模糊测试组件。扫描工具发送的请求和 Web 应用程序返回的响应存储在数据库中。
登录组件将指定的用户名和密码输入到表单中，然后单击提交按钮。成功登录后获得的 Cookie 用于抓取和模糊测试。
爬行组件以两种方式搜索页面。一种是在检索到的网页中搜索锚链接并递归地重复访问检索到的链接。另一种方法是使用从固件映像中提取的文件系统的 Web 应用程序目录中的文件路径信息。通过结合这两种方法，爬取组件在 Web 应用程序中检索尽可能多的页面。
fuzzing 组件通过改变请求头来收集动态内容。该组件创建许多具有不同组合和头字段值的请求，并将它们发送到 Web 应用程序。模糊测试请求的增加导致训练数据的增加。
数据库中有两个表用于存储收集到的请求和响应：学习表，用于下一个学习阶段，以及响应表，用于蜜罐与攻击者交互时使用。

表一所示的学习表包含请求和响应之间的对应关系。该表包含请求的方法、路径、查询、标头和正文，以及相应的“响应ID”（res id）。响应 ID 是对应于单个响应的单个数字。响应ID和响应的对应关系记录在表II所示的响应表中。记录的每个请求或响应的标头和正文中的日期、时间和 IP 地址将被删除。

TABLE I
EXAMPLE OF A TABLE USED FOR LEARNING.

method	path	query	headers	body	res_id
POST	/login.html	a=1	Conn···	Pass···	1
GET	/favicon.ico	-	Acce···	-	2
GET	/index.php	b=2	Cont···	-	3

TABLE II
EXAMPLE OF A TABLE USED FOR HONEYPOT RESPONSE.

res_id	status	headers	body
1	200	Set-cookie···	<html >Welcome···
2	404	Content-le···	Page Not Found.
3	503	Last-modif···	503-Service Unava···

D. 学习者

学习者实现了我们的第三个挑战，一种对攻击请求返回正确响应的机制。该模块的目标是从扫描程序收集的数百个响应中选择一个可能会继续与攻击者交互的响应。但是，由于攻击者的方法是一个黑匣子，我们不知道正确的响应。因此，我们专注于人工智能聊天机器人的机制。聊天机器人通过学习问题和答案之间的对应关系来实现人类水平的交流。我们的想法是创建一个聊天机器人，将 HTTP 协议请求和响应视为对话，并选择对攻击请求的自然响应。

我们的建议使用基于检索的模型，该模型从预定义的响应中提供最佳响应。具体来说，我们创建了一个多对一的序列到序列 (Seq2Seq) 模型，该模型具有一种注意力机制，可以根据学习表中记录的接收请求预测响应 ID。

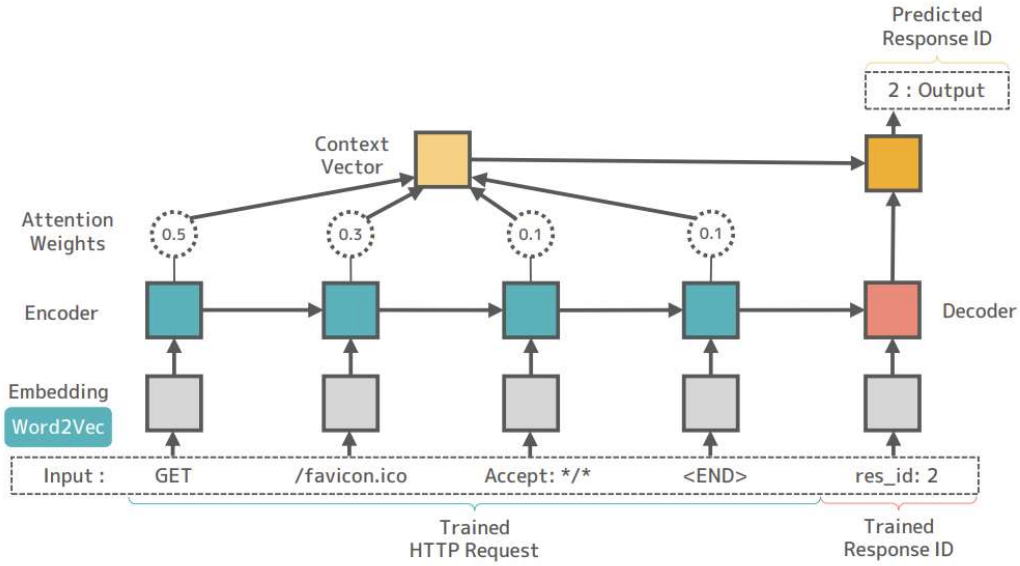

Fig. 3. Structure of our interaction model.

该模型的架构如图 3 所示。最初，学习器模块创建一个字典，将单词映射到数值。对于请求的方法、路径、查询和正文，它将每个字符串映射到单个值，对于请求头，它将每个字段的键值集映射到单个值。

接下来，通过 word2vec 算法从所有数字请求中训练一个词嵌入。经过训练的词嵌入将相似的词转换为相似的向量。矢量化请求由 Seq2Seq 中基于 GRU 的循环神经网络 (RNN) 编码。此外，注意力机制为每个输入词分配一个注意力权重，用于预测基于解码器的 RNN 的输出。

经过训练的 Seq2Seq 模型用于与蜜罐实例中的攻击者进行交互。然而，现实世界的蜜罐经常收到包含未学习单词的请求，称为词汇表外 (OOV)。因此，我们准备了 Patel 等人提出的算法。 [16] 用于处理 OOV。该算法根据 word2vec 学习到的字符串计算 OOV 的向量。因此，该模型将 OOV 视为与已经学习的单词相似的单词。通过加入这个算法，我们实现了一个可以智能响应非模拟请求的蜜罐。

E. 蜜罐实例

我们的框架最终将生成的蜜罐实例包含 Web 服务器、经过训练的模型和响应表。该实例配置了 IP 地址和开放端口，并在 Internet 上发布。

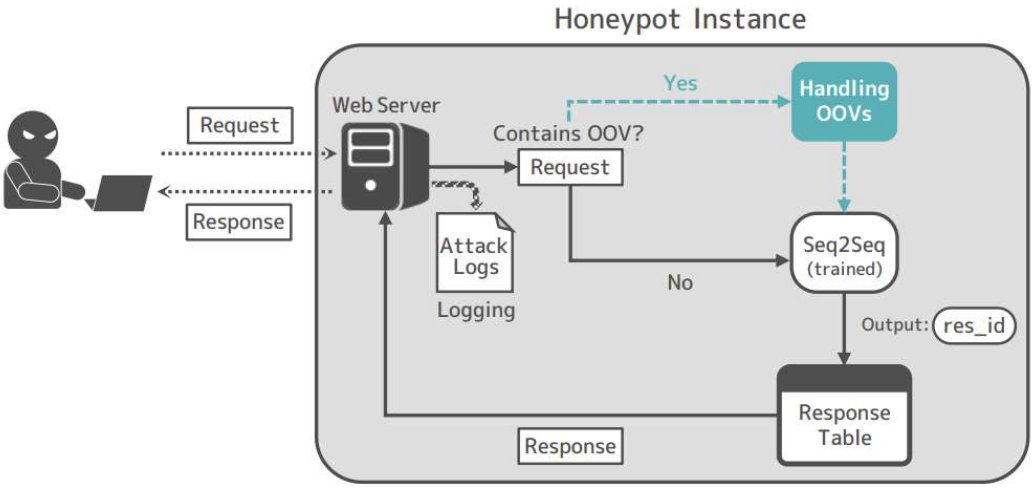

Fig. 4. Response mechanism for the honeypot instance.

图 4 显示了蜜罐如何与客户端交互。首先，检查 Web 服务器收到的请求是否包含 OOV。根据结果，应用适当的预处理并将其输入到训练的模型中。然后，使用训练好的模型预测的响应ID作为主键，在响应表中查找对应的响应。 Web 服务器向客户端返回选择的响应，其中 IP 地址和主机名替换为蜜罐的信息。最后，记录收到的请求和预测的响应以供分析。

五、绩效评价

本节展示了对我们框架的评估。实验装置如表 III 所示。

TABLE III
EXPERIMENT SETUP.

Hardware		Software	
OS	Ubuntu 20.04.3 LTS	QEMU	4.2.1
Kernel	Linux 5.8.0-48 x86_64	Docker	19.3.13
CPU	i7-10700K@3.80GHz	Binwalk	2.2.1
GPU	GeForce RTX3060	SeleniumWire	2.1.2
Memory	32GB	TensorFlow	2.3.1

TABLE IV
VENDORS OF SUCCESSFULLY EMULATED FIRMWARE IMAGES.

Dataset	Vendor	Images	Success
FirmAE Dataset	Netgear	22	19
	TP-Link	18	18
	TRENDnet	12	10
	ZyXEL	6	5
Our Samples	GL.iNet	67	67
	ELECOM	21	20
	Linksys	4	4
	Buffalo	2	2
Total		152	145

A. 支持固件

首先，我们使用了 FirmAE [17] 提供的数据集和我们从供应商网站收集的样本。 FirmAE 数据集包含 1,124 个用于无线路由器和 IP 摄像机的固件图像，其中我们使用了所有 58 个基于 OpenWrt 的图像。我们所有的示例也是基于 OpenWrt 的。

如表 IV 所示，在来自 8 个供应商的 152 个固件映像中，我们的仿真器成功启动了 145 个固件映像。在 7 个启动失败中，一个不包含 Web 应用程序，另外两个失败是因为 QEMU 不支持固件中嵌入的某些指令。其余四个因用于为 Web 应用程序配置 TLS 的证书和密钥而失败。但是，通过手动编辑 HTTP 服务的配置文件来禁用 TLS，仿真成功。

接下来，我们使用 OpenWrt 官方发布的各种 CPU 架构的固件镜像测试了我们的模拟器。结果，我们的仿真器成功启动了嵌入在具有表 V 所示 CPU 架构的固件映像中的 Web 应用程序。总共支持 10 种不同的 CPU 架构，比现有的仿真器多 7 种 [5]、[10]、 [11] 在第三节-A 中介绍。

TABLE V
CPU ARCHITECTURES OF SUCCESSFULLY EMULATED FIRMWARE IMAGES.

CPU Arch	bit	endian
MIPS	32	big, little
ARM	32	big, little
PowerPC	32	big
Intel 80386	32	little
MIPS64	64	big, little
Aarch64	64	little
x86-64	64	little

TABLE VI
COMPARISON OF THE TIME REQUIRED BY OUR EMULATOR, PHYSICAL DEVICE, AND FIRMAE EMULATOR. THE TIME IS SHOWN IN SECONDS.

	Booting Time [s]	Scanning Time [s]	Communication Time [s]
Our Emulator	44.4	1815.2	0.058
Physical Device	85.2	1655.6	0.055
FirmAE	388.7	-	0.091

B. 执行速度

为了评估我们的仿真器的执行速度，我们使用了物理设备和 FirmAE [11]，它采用了 QEMU 全系统仿真器。物理设备是 Archer C9 V5，TP-Link。我们的模拟器和 FirmAE 运行与从供应商网站获得的物理设备相同的固件映像。

使用它们中的每一个生成蜜罐的结果如表 VI 所示。使用我们的模拟器，Web 应用程序的启动时间是最快的，显示出的差异大约是物理设备的两倍。此外，一次通信所需的时间没有显着差

异，从向 Web 应用程序发送请求到在我们的模拟器和物理设备之间接收响应。另一方面，FirmAE 的通信时间大约是其他通信时间的 1.6 倍。此外，FirmAE 在蜜罐生成过程中陷入无法通信的状态。因此，我们的仿真器适用于蜜罐生成，因为它具有与物理设备相同的通信速度，并且比全系统仿真运行更稳定。

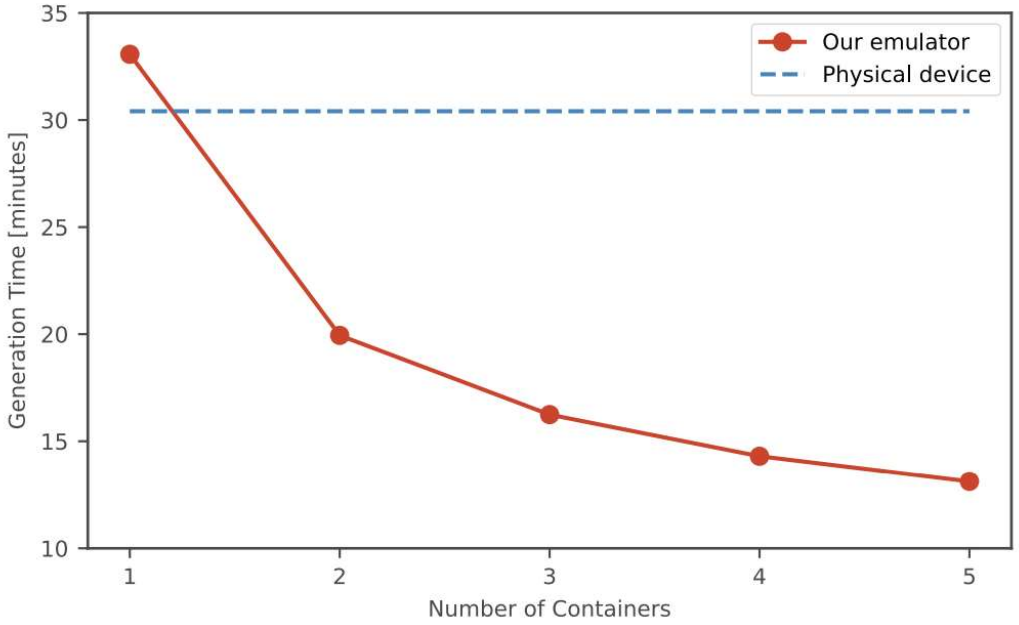

Fig. 5. Comparison of honeypot generation time between using a single physical device and 1 to 5 containers in parallel. The time is shown in minutes.

TABLE VII
RESULTS OF THE EVALUATION WITH THE FIVE FIRMWARE SAMPLES.

No.	Firmware (Vendor)	Found Paths	Generation Time	Honeypot Size
1	Archer C9 V5 (TP-Link)	498	13 min	51 MB
2	E8450 (Linksys)	352	21 min	72 MB
3	GL-MT300N-V2 (GL.iNet)	181	9 min	17 MB
4	NBG6616 (ZyXEL)	58	4 min	37 MB
5	WRC-1167GS2 (ELECOM)	440	6 min	34 MB

接下来，我们比较了使用模拟器和使用物理设备时的蜜罐生成时间，结果如图5所示。当只有一个容器时，蜜罐生成时间比使用物理设备时长。这一结果与我们的仿真器的通信时间比物理设备稍长有关，如表 VI 所示。然而，我们的方法允许我们并行使用多个容器来生成蜜罐。结果，五个并行容器的生成时间不到使用单个容器或单个物理设备的一半。我们的方法是可扩展的，因为购买物理设备来增加它们的数量是昂贵的。

C. 蜜罐生成和观察

为了评估我们框架的性能，我们使用从供应商网站下载的五个固件映像生成了蜜罐。表 VII 显示了所使用的固件映像、扫描仪的爬行组件找到的路径数量、生成蜜罐的时间以及蜜罐实例的大小。 5个容器并行时，生成蜜罐的最短时间不到5分钟，最长不

到22分钟。这个时间取决于每个 Web 应用程序的实现。此外，实例的大小受训练模型的大小和响应表的影响。
接下来，我们于 2021 年 7 月 1 日至 30 日在大学的公共服务器上设置了 5 个蜜罐。我们观察到所有蜜罐中的数据库和配置页面的请求来自与知名爬虫和搜索引擎不同的 IP 地址。此外，还确认了具有身份验证功能的 1 号至 4 号蜜罐的登录尝试。另一方面，没有认证功能的5号蜜罐没有收到与登录相关的请求。结果表明，攻击者意识到了每个蜜罐接口的差异。
最后，我们检查了每个蜜罐和客户端的会话长度。如果客户端发送一个请求，蜜罐返回一个响应，通信终止，则会话长度为 1。会话长度越长，攻击者越有可能认为蜜罐是自然系统，但 DoS 和暴力破解除外攻击。因此，会话长度被认为是蜜罐欺骗性能的关键指标之一，即学习过程的有效性[8]。
为了比较，除了我们的智能交互蜜罐（Proposed）之外，我们还准备了基于规则的低交互蜜罐（Rulebase）。基于规则的蜜罐的实现是基于Musch等人的方法。 [13]。我们还准备了一个简单的交互蜜罐（Simple），它对所有请求都响应“200 OK”。

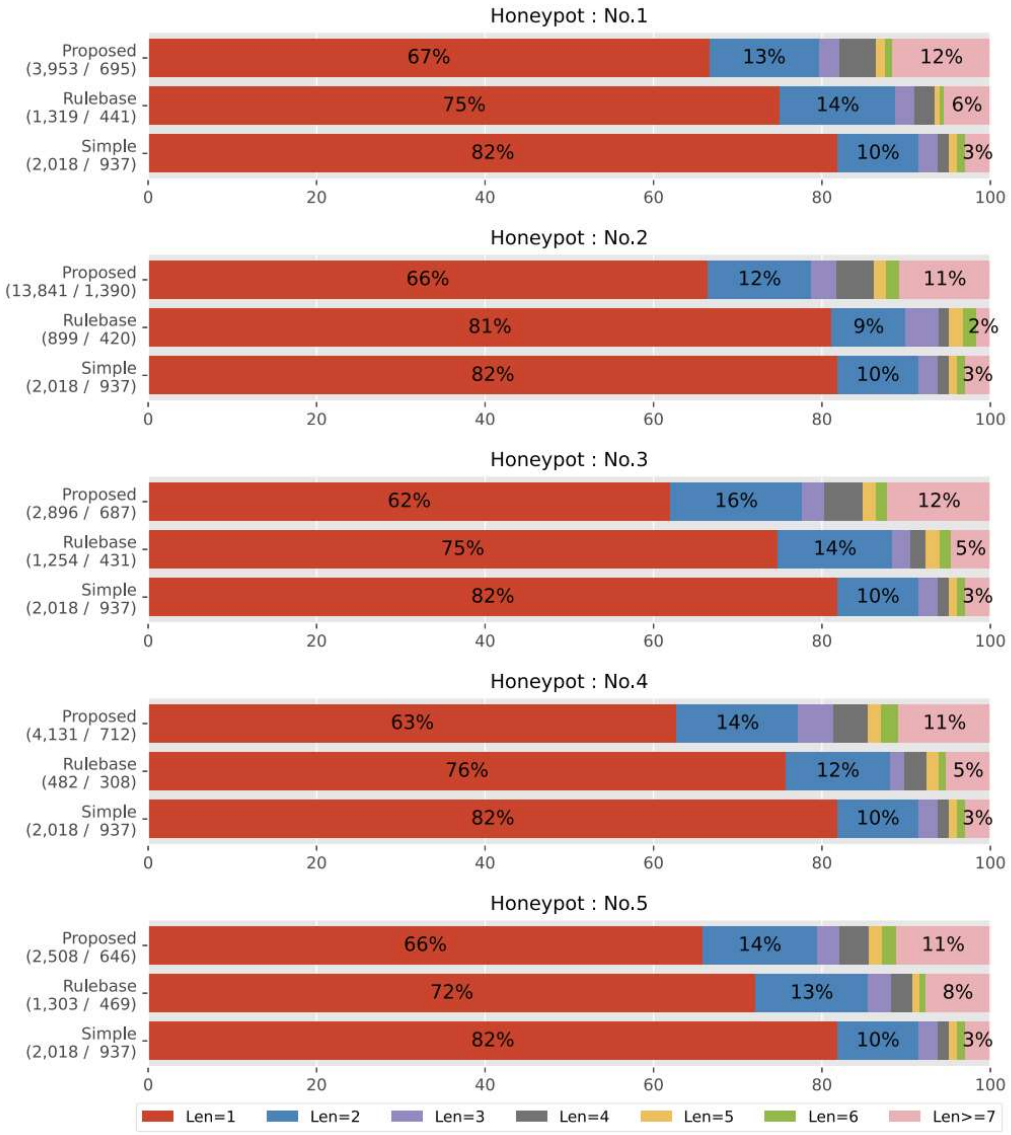

Fig. 6. Comparison in the percentage of session length with clients between honeypots. The two numbers under each honeypot method represent "(total number of requests received / total number of IP addresses observed)".

图 6 显示了我们的蜜罐的会话长度和 30 天观察的比较蜜罐。请注意，访问蜜罐的客户端数量因位置而异，因此为清楚起见，结果显示为百分比。我们的蜜罐比其他蜜罐有更多的扩展交互，尤其是持续时间超过 7 个会话的交互百分比。

为了找出通信继续进行的原因，我们在我们提议的蜜罐和基于规则的蜜罐上使用了 RouterSploit5。该工具测试系统是否响应各种现有的攻击。结果，我们的蜜罐响应了一到四次攻击，而基于规则的蜜罐对攻击没有响应。换句话说，从攻击者的角度来看，我们的蜜罐似乎是一个易受攻击的系统，可以响应攻击，并且通信可以继续进行。因此，它表明第 III-C 节中关于蜜罐返回攻击者期望的响应，从而使攻击继续进行的讨论是正确的。

六、讨论

由于在互联网上部署了我们的框架生成的蜜罐，我们观察到了攻击者的行为，例如登录。现有的没有登录功能的低交互蜜罐很难

观察到这些动作。但是，我们没有观察到诸如配置更改之类的高级攻击。这有两个可能的原因。
首先是蜜罐观测位置和周期的问题。我们的蜜罐模拟的低调 Web 应用程序需要长期观察和广域部署才能被识别为攻击目标[13]。
其次，我们的蜜罐可能已通过指纹识别被检测为诱饵系统，而攻击者正是这样做的。指纹识别是一种通过响应的不自然性来检测蜜罐的技术，并且是所有蜜罐的常见问题[5]。然而，基于 V-C 节中显示的会话长度的比较，我们可以说我们的智能交互蜜罐与攻击者的交互比低交互蜜罐更有效。此外，我们的蜜罐根据学习到的知识返回响应，而不执行攻击请求。因此，与高交互蜜罐不同，我们的蜜罐不会被破坏，也不会被攻击者劫持。它比当前用于观察攻击的高交互方法更安全。

七。道德考量

本实验和评估中使用的基于 OpenWrt 的固件映像符合 GPL 许可，可从供应商网站获取。固件镜像在蜜罐生成过程中不向互联网公开，不与外界进行通信。此外，如果在扫描阶段获得的响应包含敏感信息，则将其转换为不同的值以保证安全。今后，如果在生成蜜罐或观察攻击时发现漏洞，我们会及时向目标供应商报告并要求他们采取措施。

八。结论

本文提出了一个使用物联网设备的固件生成蜜罐的框架，以观察针对嵌入式 Web 应用程序的攻击。我们准备了一个适合蜜罐生成的固件模拟器和一个与攻击者智能交互的机器学习模型。我们的模拟器能够运行八家供应商和十种不同 CPU 架构的固件映像。此外，我们生成了模仿来自五家供应商的嵌入式 Web 应用程序行为的蜜罐，并观察了真实世界的攻击。
我们将实验限制在基于 OpenWrt 的无线路由器上。未来，我们的目标是实现一个通用的蜜罐生成框架，可以应用于其他设备的固件和通用 Web 应用程序。