网络流量分析模块 WriteUp

团队名称：红景天战队

团队成员：程逸飞（chengyifei@iie.ac.cn）、孙裴帅（sunpeishuai@iie.ac.cn）、谢江（xiejiang@iie.ac.cn）、徐红博（xuhongbo@iie.ac.cn）、徐家佳（xujiajia@iie.ac.cn）

指导老师：桑亚飞（sayafei@iie.ac.cn）

单位：中国科学院信息工程研究所

目录

一、 引言

本次 DataCon2022 比赛中，我们选择了网络流量分析模块进行分析。该模块分为挖矿流量检测、智能蜜罐环境构建两部分。基于赛前准备和比赛过程中学习的背景知识，我们分别针对这两部分进行 writeup 总结记录。

本次 WriteUp 主要有以下章节，第二部介绍相关背景知识，第三部分介绍挖矿流量检测，第四部分介绍智能蜜罐环境构建（Level1 和 Level2），第五部分为总结与展望。

二、 背景知识

本次 Datacon 竞赛的网络流量分析赛道共给出了两个方向的挑战赛题，分别是挖矿流量检测和智能蜜罐构建。在比赛开始前，我们进行赛前准备的时候，就已经对这两个领域的相关背景和方法进行了调研分析，因此在题解中我们也将这部分内容展现出来，算是对我们比赛过程中解题思路的一个初步梳理。

2.1 挖矿流量检测

总体上来说，对挖矿流量的检测要根据实际针对的挖矿行为所采取的流量通信方式，可以分为三类，如图 2.1 所示。

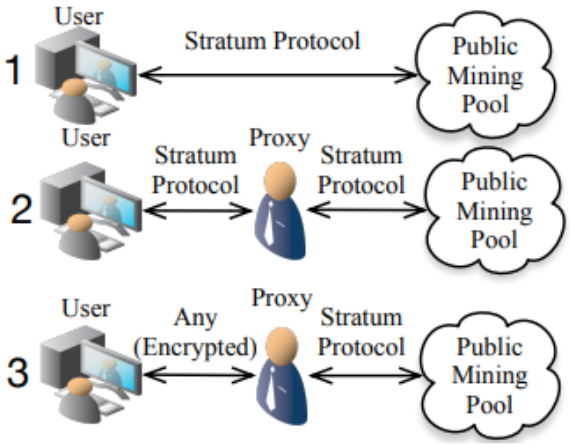


图 2.1 挖矿流量通信方式

- ✓ 挖矿用户与矿池直接通信，使用 Stratum 协议(或类似的明文 TCP 协议)进行明文通信。

- ✓ 挖矿用户借助代理转发与矿池间接通信，代理两端均使用明文协议。
- ✓ 挖矿用户借助代理转发与矿池间接通信，代理与用户使用加密通信，与矿池直接使用明文通信。

在实际分析场景中，所获取的流量大多来自于用户侧，并且主要关注流量行为本身，对于服务端的具体情况没有太大的关注度；另一方面，赛事方也对流量矿池进行了匿名化处理。因此，我们姑且将其归纳为加密和明文两类情况，不关注是否为代理转发。

2.1.1 明文挖矿流量分析

2.1.1.1 流量特征

基于明文通信的挖矿流量一般在 tcp 报文中会留下较为明显的通信特征字符串或关键词，而不同币种在挖矿时使用的明文通信协议不同，所对应的关键字集合也会有细微差别。

以常见的门罗币(XMR)挖矿流量为例，其使用的 Stratum 协议是基于 JSON-RPC2.0 封装的 TCP 通讯协议，在挖矿主机和矿池的交互过程中支持挖矿主机登记、任务下发、账号登录、结果提交和挖矿任务难度调整这五种行为，在 tcp 报文中各类行为都对应不同的关键词。

除了 Stratum 协议以外，XMR 还加入了 xmr-stak，xmrig，claymore 等协议，基础框架都是基于 JSON-RPC，只是在挖矿算法配置上有修改

2.1.1.2 威胁情报关联

相近网段 IP、挖矿木马和矿池关联 IP、json 文件 hash、关联的挖矿钱包地址等。

2.1.2 加密挖矿流量分析

为了对抗监管检测，许多矿池在提供挖矿接口时都封装了加密协议，使得传统的明文流量分析策略失效。此时需要借助负载以外的间接流量特征。

- ✓ 文献[1]中总结了包括矿工通信时间、加密数据包大小、加密流标志位占比等流量特征。
- ✓ 文献[2][3]等使用机器学习方法，基于通用的加密网络流量指纹特征，训练分类器模型进行识别。除了上一条中提及的以外，还包括总体的包大小统计特征、包时间间隔统计特征、序列特征、握手协议指纹等。
- ✓ 相较于需要训练集的监督学习方法，文献[4]提出了一种利用挖矿行为本身固有模式的无监督的机器学习检测方法。通过计算加密挖矿流量序列与实际区块产出序列的相似度进行识别(图 2.2)。

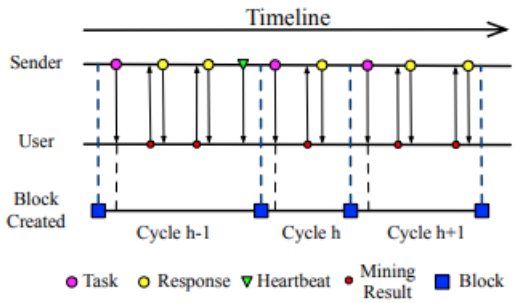


图 2.2 加密挖矿流量序列相似度识别

2.2 智能蜜罐构建

传统意义上的蜜罐其价值在于被扫描、攻击和攻陷，主要作用就是欺骗黑客，从而诱导攻击并最终捕获攻击。

对蜜罐的分类标准一般是蜜罐的真实度，即交互程度。可以分为中低交互和高交互蜜罐：

中低交互蜜罐一般不会提供一个真实的操作系统或服务，要么模拟服务主要特征功能，要么提供更多的交互信息从而诱导更多的攻击。目前较为常用的中低交互蜜罐开源产品有 Hfish、T-pot、Kippo 等。

高交互蜜罐完全提供真实的操作系统和网络服务，从而引诱更多的黑客攻击信息，但同时又加大了部署和维护的复杂度。例如 Chameleon，直接提供 DNS、HTP Proxy、HTTP、HTTPS、MySQL 等真实交互环境。

另一方面，在对蜜罐进行性能测试时，则需要从防御者的角度提供一定数量的模拟攻击或漏洞验证(POC),这类方法一般针对只模拟服务主要特征功能的低交互蜜罐。目前国内使用较多的开源 poc 库有长亭科技的 xray[5]和知道创宇的

fscan/WebScan[6]等。如果想要引导这类 poc 命中乃至进行下一步攻击，作为低交互蜜罐可以在服务端部署相应规则脚本一一处理，也可以使用自动化工具直接生成漏洞规则，例如专门针对 xray 库生成可供扫描漏洞的开源库 yarx[7]。当然，此类策略对于 0day poc 的攻击没有处理能力。

三、 挖矿流量检测

3.1 概述

根据题目要求，我们需要审计某校园网网关出一段时间的出入流量数据（通过 pcap 文件形式给出），并且尽可能准确地（对漏检率和误检率都有要求）识别出其中的挖矿流量，最终形成行之有效的检测规则以便扩展到更多的应用场景。题目的挑战在于（1）已对流量中的域名和 ip 进行了匿名化处理，无法通过威胁情报关联的方法来锁定恶意 ip；（2）已对流量中的大部分 tls 握手信息进行了随机化处理，无法通过分析 tls 握手阶段的明文信息锁定恶意挖矿程序；（3）校园网内部 ip 数量多，流量类型分布多样，准确识别挖矿流量的难度进一步增加。

我们首先对题目流量数据进行统计性分析，使用 wireshark 的“会话审计”功能共展示出 1754 个 ipv4 会话二元组（源 ip、目的 ip），进一步地由这些 ipv4 会话构建了 5950 个 tcp 会话五元组（源 ip、源端口、目的 ip、目的端口、协议类型）。此处我们重点对 ip 会话包数量进行考察，除去极少数包数量超过 5000 的会话，绘制直方图如图 3.1 所示。本题流量数据包中超过 80%的 ip 会话包数量都在 200 以内，整体分布呈尖峰重尾特性，符合流量分布一般规律。而挖矿流量大部分阶段都是处于规律传递挖矿信息应答包的模式，除非当前会话连接时间特别长，否则其整体的会话包数量通常也在数百个以内，因此从本题的数据中识别挖矿流量有如“大海捞针”。

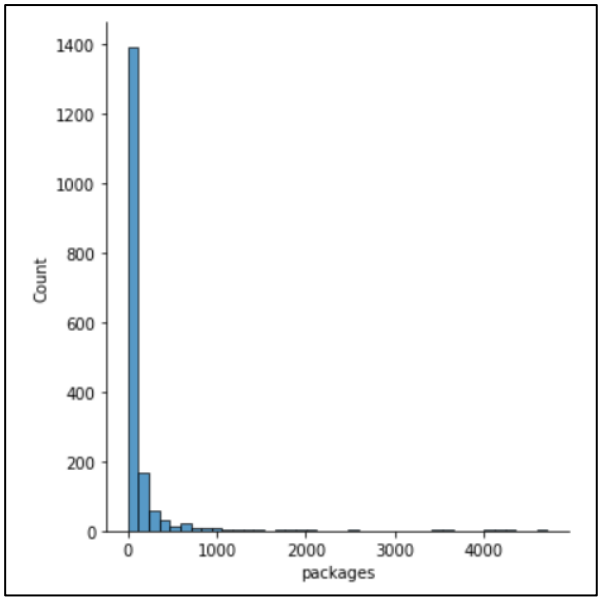


图 3.1 ip 会话包数量直方图

由后验知识我们可以得知题目中的挖矿流量包含了传统挖矿流量和加密挖矿流量，而这两种挖矿流量的识别方法有着较大的差别，因此下文将分别对普通挖矿流量识别和加密挖矿流量识别两个方面展开分析。

3.2 传统（明文）挖矿流量识别

传统挖矿流量通过特定的协议进行明文任务信息传递，stratum 协议是目前最常用的矿机和矿池之间的 TCP 通讯协议，因此可以使用深度包检测的方法从 stratum 协议内容出发来制定挖矿流量匹配规则。

3.2.1 协议分析

矿机和矿池间的一般通信过程主要分为矿机登记、任务下发、账号登录、结果提交与难度调整等部分，如图 3.2 所示。

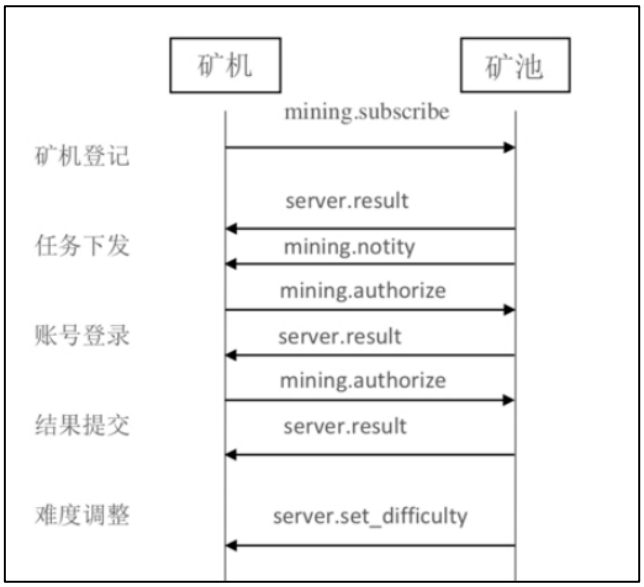


图 3.2 stratum 协议

其协议细节如下：

- ✓ 任务订阅：矿机启动，首先以 mining.subscribe 方法向矿池连接，用来订阅工作。矿池以 mining.notify 返回订阅号、ExtraNonce1 和 ExtraNonce2_size。
- ✓ 任务分配：该命令由矿池定期发给矿机，当矿机以 mining.subscribe 方法登记后，矿池应该马上以 mining.notify 返回该任务。
- ✓ 矿机登录：矿机以 mining.authorize 方法，用某个帐号和密码登录到矿池，密码可空，矿池返回 true 登录成功。该方法必须是在初始化连接之后马上进行，否则矿机得不到矿池任务。
- ✓ 结果提交：矿机找到合法 share 时，就以”mining.submit“方法向矿池提交任务。矿池返回 true 即提交成功，如果失败则 error 中有具体原因。
- ✓ 难度调整：难度调整由矿池下发给矿机，以 mining.set_difficulty 方法调整难度，params 中是难度值。矿机会在下一个任务时采用新难度，矿池有时会马上下发一个新任务并且把清理任务设为 true，以便矿机马上以新难度工作。

3.2.2 规则制定

根据上述分析和相关资料，我们总结出以下关键词可以用于传统（明文）挖矿流量识别的深度包检测，如表 3.1 所示。

表 3.1 深度包检测挖矿流量关键词

jsonrpc	nonce	XMRig	seed_hash
job_id	argon2	params（易混淆）	algo（易混淆）
login（易混淆）	submit（易混淆）	mining.xx	client.xx

之后我们就可以使用 wireshark 的查找功能，设置对“分组字节流”中的“字符串”格式进行查找，如图 3.3 所示，查找“jsonrpc”便可锁定可能存在的挖矿流量。

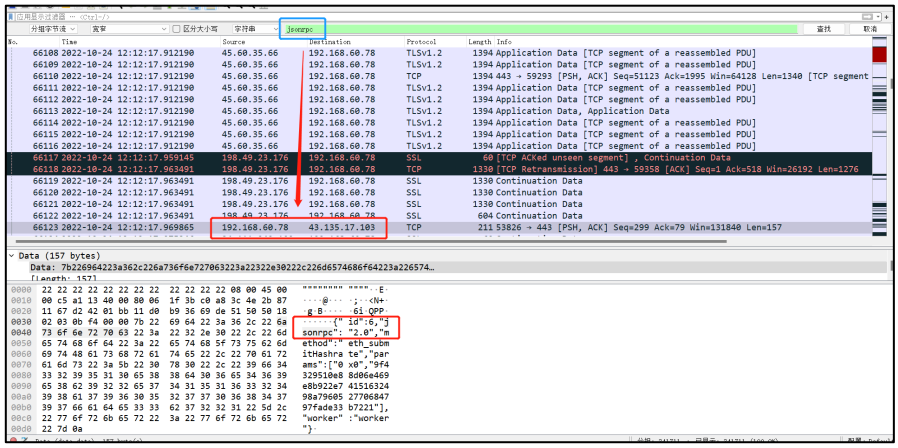


图 3.3 通过关键词识别挖矿流量示例

3.3.3 识别结果

通过上述方法，共识别出五个传统（明文）挖矿主体 ip，传统（明文）挖矿流量 189 条。我们还将识别出的挖矿流量中的关键字构建了词云图（图 3.4）。经过后验知识比对，上述方法在本题构建的环境下对于传统（明文）挖矿流量识别的精确率和召回率均达到了 100%，且该方法使用对明文流量的深度包检测，精确把握挖矿流量特征，泛用性强，能够很好地迁移应用到其他场景之中。

图 3.4 挖矿流量关键字词云

3.3 加密挖矿流量识别

由于传统（明文）挖矿流量的自身就具有明显的特征字段且关于它的识别研究工作已经相对成熟，所以在第一阶段的识别工作中我们同大多数队伍的情况一样，并没有遇到太大的困难。在完成了传统（明文）挖矿流量识别的阶段之后，才真正迎来了本赛题的难点。这一阶段我们的解题思路主要分为三个方面，我们首先打算从流量统计特征相似度的角度入手，通过寻找一些公开可获得的加密挖矿流量样本，分析已有样本和加密挖矿流量样本的相似度，但是苦于无法找到合适的对照样本，我们做相似度比较的时候只能选择了阶段一中的明文挖矿样本，这也导致最终效果并不理想；第二方面，我们完成了基于矿池服务器列表的检测识别方法，虽然主办方已经将 ip 和域名进行了匿名化我们在这种方法中一无所获，但是这种检测识别方法在真实环境中构建了威胁情报，并进一步检测挖矿 ip 确是行之有效的，对通用的挖矿流量检测场景帮助较大；最后我们重新思考了加密挖矿流量所应具有的关键统计特征，并总结提取了简单有效的加密挖矿流量识别规则。

3.3.1 统计特征相似度匹配（失败）

在没有已知的加密挖矿流量样本的前提下，我们考虑加密挖矿流量只是对原始挖矿流量在传输层封装了加密协议，对于挖矿协议自身可能暴露的统计特征影响不大，因此我们采用第一阶段中已经识别出的挖矿流量样本作为对照。使用 cicflowmeter 工具，我们可以方便地提取出流级别的丰富统计特征，之后我们将所有待检测样本与已知挖矿流量进行了余弦相似度计算。由于每次待检测样本是 1 个，而已知挖矿流量是多个，因此我们考察它们之间的最大相似性和平均相似性这两个指标（图 3.5），并对前二十的结果取交集作为最终结果。通过比赛得分我们得知这种方法识别的结果一个都没对，细致分析其失败原因我们认为只考虑流级别统计特征是不够的，对于重要的时序特征（如规律心跳特征）、tcp 协议层面特征等综合分析时检测识别挖矿流量的关键。

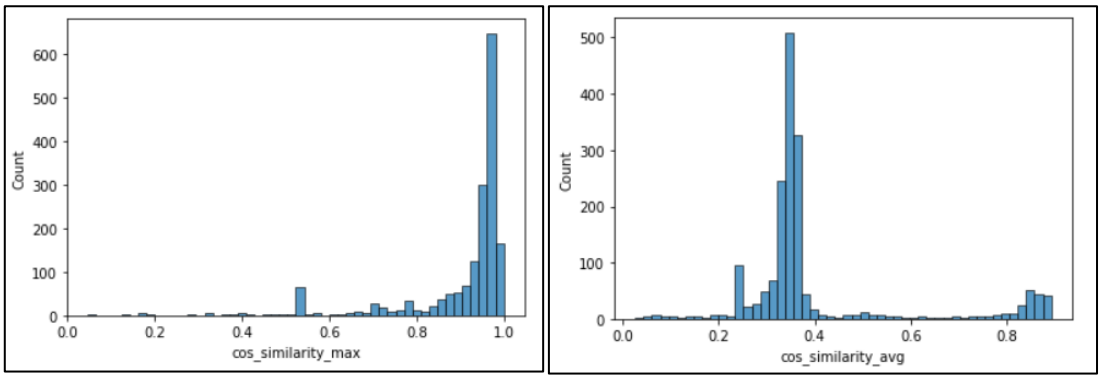


图 3.5 统计特征相似性指标

3.3.2 基于矿池服务器列表的检测方法（失败）

对于一般的加密挖矿流量，其出向通信 ip 一般都是矿池子域名对应的 ip。因此我们通过网上公开信息列举矿池及其子域，并将它们转换为 IP 地址的方法，利用矿池服务器域名和 ip 就可以检测加密挖矿流量[1]。我们使用 python 爬虫的方式首先爬取了 https://miningpoolstats.stream/网站上公开可获得的目前主流加密货币矿池网站，之后通过在线的子域名查询网站和域名 ip 映射网站，自动化获得了上万个曾在 pcap 捕获日期内映射到各矿池服务器子域名的 ip，通过与当前流量文件中的 ip 进行撞库的确获得了三个对应上的 ip。我们知道主办方已经将文件中的服务器域名和 ip 进行了匿名化，这种方法的尝试命中了只是巧合，但是我们使用的这种主动式的矿池服务器列表检测方法在实际检测加密挖矿流量工作应该是行之有效的。

3.3.3 基于流量特征分析的检测方法

前面的尝试都失败了，此时我们对加密挖矿流量特征层面进行了再思考。文献[2]中将常见挖矿流量通信特征总结了一些几点。

- ✓ 数据包一般很小，数据包大小通常在 40 到 120 字节之间。
- ✓ 目标端口小于源端口。
- ✓ 大多数流都包含 ACK&PUSH 标志位。
- ✓ 双方通常不会主动发送 FIN 标志位断开连接，即大多数流都不包含 FIN 标志位。

✓ 大多数流不包含 RST 标志。

通过加密挖矿流量行为的分析和总结，我们对该规则进行了改进扩充，增加了五条规则（1）通信期间存在明显的心跳特征（如图 3.6 所示）；（2）会话包数量小于 200；（3）持续时间大于 5 秒；（4）包负载长度为零的情况不能超过会话包总数的五分之一；（5）包负载长度达到需要分片的的次数不能超过 5 次。

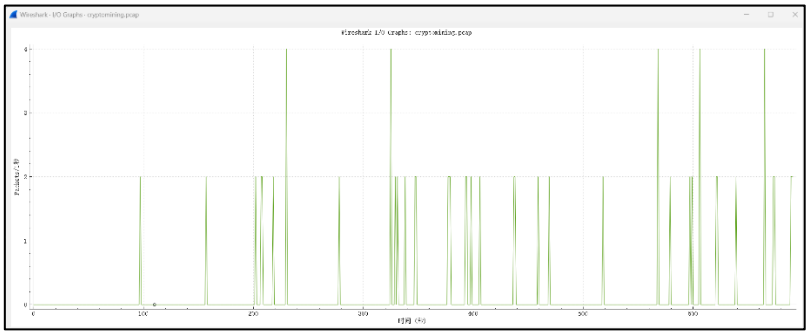


图 3.6 加密挖矿流量心跳特征图

通过以上规则我们进行了流量的筛选，最终筛选出 4 个可疑 ip（如图 3.7），顺利确定出了加密挖矿流量，精确率和召回率均达到了 100%。

	ip	会话持续时间	包数量
大可能	23.223.18.120	58秒73毫秒120	26
中大可能	120.79.19.176	26秒44毫秒610	45
大可能	120.79.12.36	6分48秒481毫秒	64
极大可能	39.106.134.208	9分52秒746毫秒	88

图 3.7 加密挖矿流量 ip 确定

综上所述我们对于挖矿流量检测的最终规则如表 3.2 所示。

表 3.2 加密挖矿流量检测规则

1. 通信期间存在明显的心跳特征
2. 会话包数量小于 200
3. 持续时间大于 5 秒
4. 包负载长度为零的情况不能超过会话包总数的五分之一
5. 包负载长度达到需要分片的的次数不能超过 5 次
6. 数据包大小通常在 40 到 120 字节之间
7. 大多数流都包含 ACK&PUSH 标志位
8. 目标端口小于源端口
9. 大多数流都不包含 FIN 标志位

10. 大多数流不包含 RST 标志

3.4 挖矿流量检测总结

综上所述，我们设计并实现了多层次完善的挖矿流量检测方法，分为三阶段，首先对于明文流量进行深度包检测匹配关键字，其次构建矿池服务器列表威胁情报（在实际场景下有意义），最终利用表 3.2 所提规则进行流量特征层面的分析检测。该层次化方法对于不同类型的挖矿流量都具有良好的识别能力，通过本次比赛的验证，该方法取得了精确率和召回率均为 100%的效果。

四、 智能蜜罐环境构建

蜜罐环境构建由两部分够成（Level1 与 Level2），接下来分别进行介绍。

4.1 智能蜜罐环境构建-Level1

4.1.1 概述

题目挑战内容：选手需要在提供的虚拟环境中构建一个智能靶标环境；其中该环境要求开放指定端口以接收 poc，并对接收到的 poc 进行正确的响应以获得分数，每次正确响应都会得到相应分数，满分为 100 分。

本部分主要是对蜜罐构建首先进行初步探索，熟悉蜜罐处理攻击者请求的各个流程。

4.1.2 解题的四个阶段

4.1.2.1 第一阶段 “石器时代”

在印象中，我们了解的蜜罐都是像 Hfish 这种的实际部署有缺陷的服务，然后对攻击进行记录的产品。因此，在比赛刚开始的前几天，我们队一直在向这个方向努力。Hfish 这种类型需要实际部署的蜜罐一般均需要比较高的权限去创建文件夹和服务。受限于比赛环境的权限限制，解题一开始的进展很缓慢，我们逐

渐尝试了 Hfish、honeyhttpd、honeypots 等蜜罐，并经历了从高权限蜜罐->python 库类型低权限蜜罐(如 honeypots)的尝试。并且在部署的过程中，我们意识到单端口限制也是一个需要解决的问题，如果实际部署服务，必然要对 8888 端口的请求进行转发。由于一次请求智能接收端只会接受最前的那一次回复，因此转发的过程中必然要对请求进行判断，那么手动对接收到的请求进行判断或许是解决此题目绕不过去的坎。

在成功部署了一款 python 类型的蜜罐 honeypots（如图 4.1），并利用其开启 ssh 服务之后，我们进行了比赛的第一次有效提交，然而提交却超时了，获得了 0 分。至此，我们发现，实际部署的思路完全偏离了出题人意图。于是我们迅速调整了比赛做题方向，开启蜜罐方向的第二阶段。

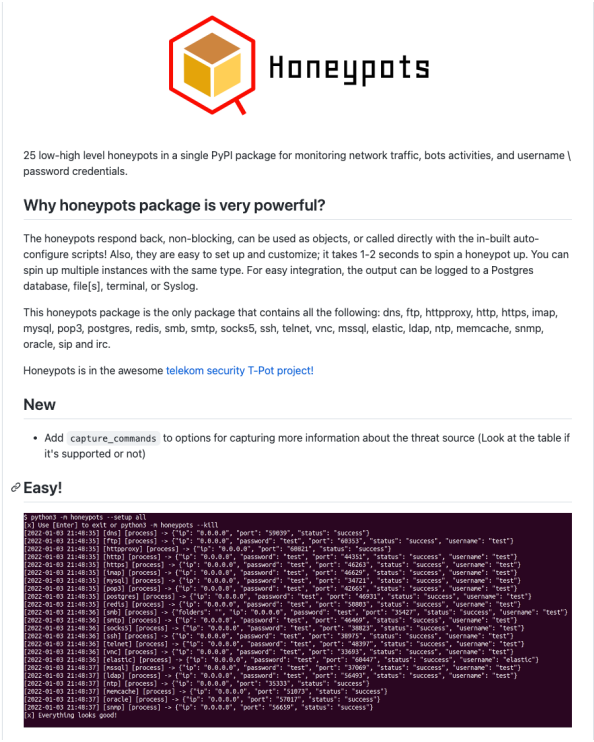


图 4.1 python 类型的蜜罐 honeypots

4.1.2.2 第二阶段“农耕时代”

解决问题的关键在于首先模拟攻击者的行为和感知他的攻击请求。关于感知他的攻击请求方面，一开始我们使用 tcpdump 对打过来的流量进行了抓包，然而在没有部署相应的服务的时候，8888 端口产生的流量基本上都是未联通的包，看不到什么信息。而且由于虚拟机性能问题和环境限制，无法将流量转移到线下

进行分析。此时我们决定使用 python 轻量服务器框架库 flask，首先输出攻击者打到 8888 端口的请求信息，并将其整理到了在线表格中（如图 4.2 所示），至此，我们终于看到了攻击者的攻击详细信息。

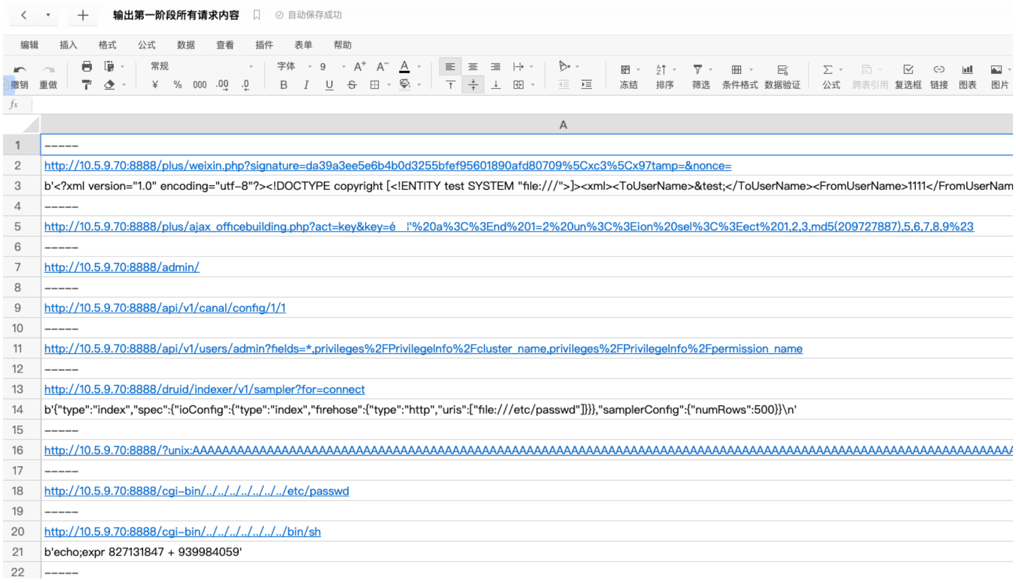


图 4.2 蜜罐环境资料在线表格

其次，为了让全队更好了解到攻击者的整个动态过程，以及我们要做的事情。我们部署了 vulhub 环境（解题者，图 4.3）和漏洞测试框架 pocassist（出题人），并在讨论会上进行了演示，完整的模拟了出题人在虚拟机上所进行的攻击发起与得分判定过程。同时，在讨论会上，我们确定了使用 flask 框架去实现这样的一个蜜罐。于是，全队根据 flask 输出请求的条数进行了分工，由 3 个人进行 poc 寻找，1 个人进行蜜罐开发，1 个人进行智能系数的研究。很快，6 分…8 分，我们的分数开始了逐步上升的过程。

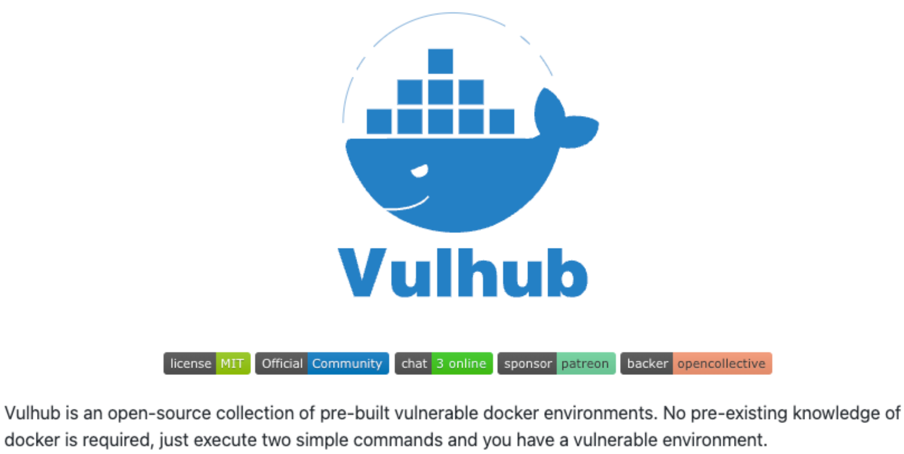


图 4.3 vulhub 环境

4.1.2.3 第三阶段“电气时代”/“自动化时代”

在寻找 poc 的过程中，我们发现，有一个 github 的工具 Xray，其中包含了我们接收到的绝大多数请求。而其他队伍貌似轻轻松松就完成了第一阶段的解答，于是我们猜测会不会存在某种自动化的反制工具。经过简单的搜索，我们终于发现了蜜罐第一题的版本答案：yarx（图 4.4）。Yarx 是 xray 的开发作者 zema1 所开发的一款反制工具。它能够根据 xray 的 yaml poc 规则全自动的生成一个满足规则要求的 Server，使用 xray 扫描该 Server 将会扫描出对应的漏洞。

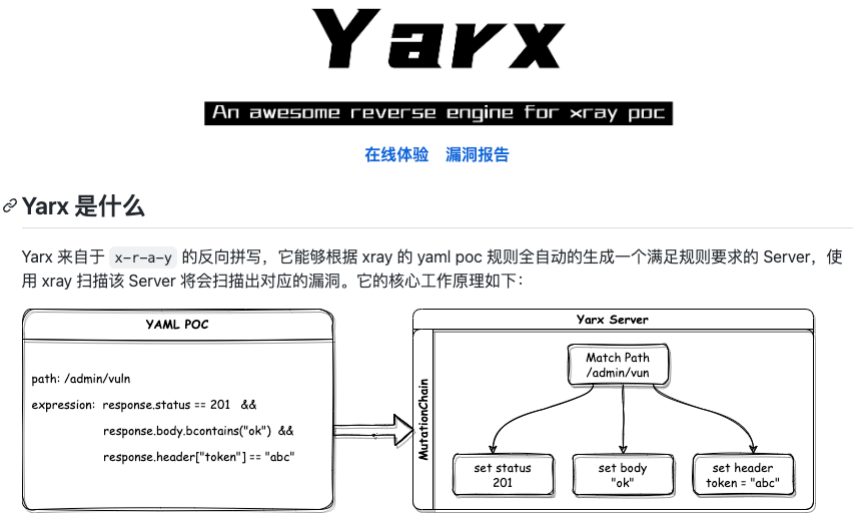


图 4.4 Yarx 标靶环境

在服务器上进行了简单的部署之后，我们的分数达到了 95 分。

4.1.2.4 第四阶段“智能时代”

在第三阶段之后，考虑到题目性价比问题，我们暂时放下了蜜罐一题目的解决。然而，随着比赛的竞争越来越激烈，因此我们需要得到一切可以获得的分数，于是我们又回过头对蜜罐一进行了分析。从对蜜罐二中 poc 的分析，我们发现部分 poc 是出题人对原有 poc 进行了微小的改动，因此我们将修复后的 poc yml 文件重新补充道 yarx 的文件夹中，重新提交，即获得了满分。

4.2 智能蜜罐环境构建-Level2

4.2.1 概述

题目挑战内容：选手需要在提供的虚拟环境中构建一个智能靶标环境；其中该环境要求开放指定端口以接收 poc，并对接收到的 poc 进行正确的响应以获得分数，每次正确响应都会得到相应分数。特别的，一个 poc 可能会有多次请求，环境对每次请求进行正确响应都会获得分数，直至将该 poc 的所有请求正确响应。

在 Level1 的基础上，我们对 yarx 蜜罐平台进行扩充优化，针对不同请求类型进行针对性改写，以便于回复正确的响应。

4.2.2 智能系数分析

Level2 增加了智能系数，更全面判断标靶环境对不同 poc 的响应能力。为了更好的实现智能蜜罐的构建，我们对于智能系数进行了如下分析：

4.2.2.1 消除特征

对于攻击者来说，如何根据蜜罐开发的固有属性来发现其指纹特征是绕过蜜罐的关键。而对于蜜罐开发者来说，如何消除这些特征也是开发者如何让提高一个蜜罐智能系数的关键。如图 4.5 所示（图片来自知乎@小安），部分开源蜜罐在响应过程中，总有部分属性是固定的，因此攻击者可以利用这些响应特征对蜜罐进行识别。

蜜罐	协议	端口	请求数据	响应特征
whoisscanme	TCP	ALL	空	whoisscanme:https://github.com/bg6cq/whoisscanme
Cowrie	TELNET	23/2323	空	\xff\xfd\x1flogin:
Amun	IMAP	143	\r\n\r\n	a001 OK LOGIN completed
Dionaea	FTP	21	空	220 Welcome to the ftp service\r\n
Dionaea	MSSQL	1443	空	\x04\x01\x00\x2b\x00\x00\x00\x00\x00\x00\x1a\x00\x06\x01\x00\x20\x00\x01\x02\x00\x21\x00\x01\x03\x00\x22\x00\x00\x04\x00\x22\x00\x01\xff\x08\x00\x02\x10\x00\x00\x02\x00\x00
Kojoney	SSH	22/2222	空	SSH-2.0-Twisted\r\n
Nepenthes	FTP	21	空	---freeFTPd 1.0---warFTPd 1.65---\r\n
Nepenthes	NETBIOS	2103	空	\x82\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
Conpot	S7	102	空	Serial number of module: 88111222
Conpot	MODBUS	502	空	Device Identification: Siemens SIMATIC S7-200
Dionaea	Memcached	11211	空	version: "1.4.25" AND response:"pointer_size 64" AND response:"STAT rusage_user 0.550000"
Dionaea	SIPD	5060	空	SIP/2\.0 200 OK\r\nContent-Length: 0\r\nVia: SIP/2\.0/TCP nm;branch=foo\r\nFrom: sip:nm@nm;tag=root\r\nAccept: application/sdp\r\nTo: sip:nm2@nm2\r\nContact: sip:nm2@nm2\r\nCSeq: 42 OPTIONS\r\nAllow: REGISTER, OPTIONS, INVITE, CANCEL, BYE, ACK\r\nCall-ID: 50000\r\nAccept-Language: en\r\n\r\n
sshesame	SSH	2022	空	SSH-2.0-sshesame

图 4.5 蜜罐特征

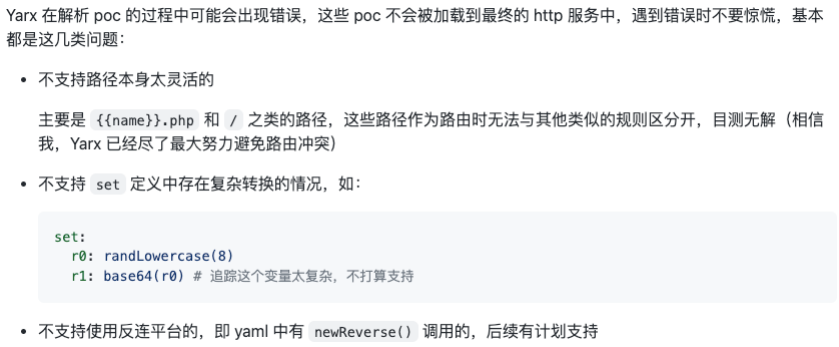
在蜜罐二赛道中，我们发现存在指向根路由的 get 和 post 请求，而这些请求貌似不对应任何 poc。于是我们线下进行了测试，发现这些默认请求返回的是 yarx 的默认加载主页，所以要消除掉这个特征，才能提高我们实现的蜜罐的智能系数。我们的解决方案是采用 flask 转发架构，首先解析从 9999 端口获得的请求，判断到是默认路由之后，返回预设的 apache404 页面，在实际测试后发现，加了这一步之后，我们的分数有所提升，因此我们认为这一手段有助于提升我们的智能系数。

此外，我们在实际测试过程中还发现了 flask 返回请求默认 header 中会加入 Server 属性，这一属性的值为：Werkzeug/2.2.2 Python/3.9.13。因此这一点也有可能会影响智能系数的判定。然而修改这一属性需要修改源码，仅仅在代码中修改 headers 只能完成追加，由于时间原因，我们最终未能实现这一修改。

4.2.2.2 实现智能解析与分发

通过对攻击者内容的详细解析，我们发现出题人有可能会对 poc 中进行些许改动，从而导致原有 poc 解析失败，如“+”改成“-”等等。且 yarx 的实现并非完美，

如下图所示，yarx 对于路由冲突的 poc 和较为复杂的变量转换是不支持的。但是通过 yarx 的解析，又可以帮助我们将对攻击请求的解析转换为简单的 poc 编写问题（因为 yarx 可以自动解析 poc），为了保留对 yarx 的使用，我们需要对其进行扩展，如图 4.6 所示。


Yarx 在解析 poc 的过程中可能会出现错误，这些 poc 不会被加载到最终的 http 服务中，遇到错误时不要惊慌，基本都是这几类问题：

- 不支持路径本身太灵活的

  主要是 `{{name}}.php` 和 `/` 之类的路径，这些路径作为路由时无法与其他类似的规则区分开，目测无解（相信我，Yarx 已经尽了最大努力避免路由冲突）

- 不支持 `set` 定义中存在复杂转换的情况，如：

```
set:
  r0: randLowercase(8)
  r1: base64(r0) # 追踪这个变量太复杂，不打算支持
```

- 不支持使用反连平台的，即 yaml 中有 `newReverse()` 调用的，后续有计划支持

图 4.6 poc 规则

然而 yarx 是由 go 编写的，我们对 go 并不熟悉，为了兼顾 yarx 的优点和支持智能分发，我们采用了如图 4.7 所示的处理架构。

图 4.7 智能标靶平台主要架构

Flask 负责解析请求，并根据请求内容分发给 yarx，对于简单的、不复杂的 poc，Flask 直接转发给 yarx 获取结果并回复给攻击者，对于请求较为复杂的 poc，如出现转义字符或者需要结合多个请求内容进行回复的，我们自定义 Flask 路由函数对请求进行单独解析。通过对蜜罐实现架构的转换，我们大大增加了蜜罐可处理的 poc 种类，并从基础上保障了后续分数的提升。

4.2.2.3 躲避检查点

在 poc 的处理过程中，我们发现攻击者有时会发送重复的请求，两个或多个重复的请求或许有些许差异，或许完全一致。在进行处理的过程中，我们发现，假如对于用于探测的非 poc 请求回复同样的内容，会导致分数出现急剧下降。因此，我们分析，出题人为了考察蜜罐“根据协议特征，永远返回正确的响应”特点，

设置了用于检测蜜罐智能程度的检查点。对于这种情况，我们采用 Flask 进行解析，当发现疑似检查点时，返回默认网页以差异化返回内容，以此来对检查点进行绕过。

4.2.3 标靶环境构建

Level2 中需要开启服务端口号 9999，接受 poc 数量约 300 多个，其中 GET 类型和 POST 类型 poc 分别为 200 余和 100 余个，其余为 PUT 类型。

根据对接受到的 poc 进行分析，我们发现相比于 Level1，Level2 中请求的 poc 大多针对 Level1 中的发出的 poc 进行了细节上的处理。在不影响 poc 原有功能的基础上，对其他字符进行了改动，以此测试标靶环境针对 poc 的智能处理能力。因此，我们在 Level2 中基于 yarx 平台主要进行了两方面优化，一是使用 flask 编写自定义服务器、增加转发功能，二是根据请求的 poc，针对性扩充对应的响应方式。

Flask 是一个轻量级的可定制框架，使用 Python 语言编写，较其他同类型框架更为灵活、轻便、安全且容易上手。它可以很好地结合 MVC 模式进行开发，开发人员分工合作，小型团队在短时间内就可以完成功能丰富的中小型网站或 Web 服务的实现。

因此，本次我们选择 flask 构建服务器，起架构如如 4.8 所示，开放端口后根据请求进行解析，然后进行针对性回复。

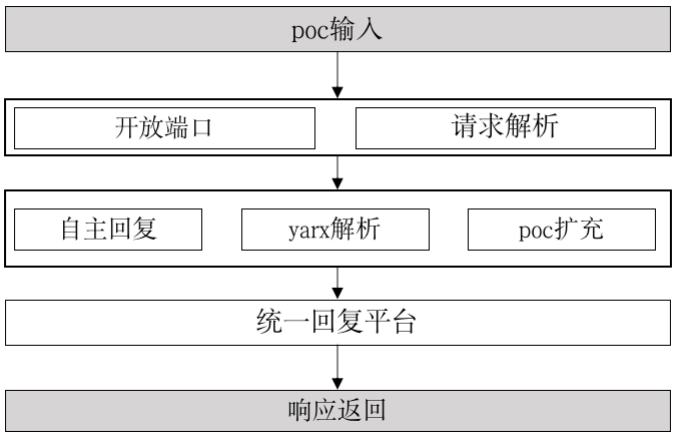


图 4.8 Level2 中基于 Flask 的自定义服务器蜜罐架构

其运行方式为——

- ✓ 自主回复：首先对请求的 poc 进行分析，如果为固定字符串的回复，比如“"root:[x*]:0:0:".bmatches(response.body)”等，则使用自定义响应函数直接回复；
- ✓ yarx 解析：针对需要计算 md5 值或者多个请求组成的 poc，我们将其转发至 yarx 平台，获得 yarx 平台的响应之后再进行分析，然后将其验证之后回复至请求方；
- ✓ 扩充 poc：利用 yarx 的规则解析功能编写更多的 poc 以扩充标靶环境的响应解析能力。

如果上述结果均未对 poc 进行正确的响应，我们的标靶平台会统一返回固定的响应，内容为常见的 poc 响应数据的集合，尽可能增加其命中概率，提高智能系数。其 poc 请求数据处理流程如图 4.9 所示。

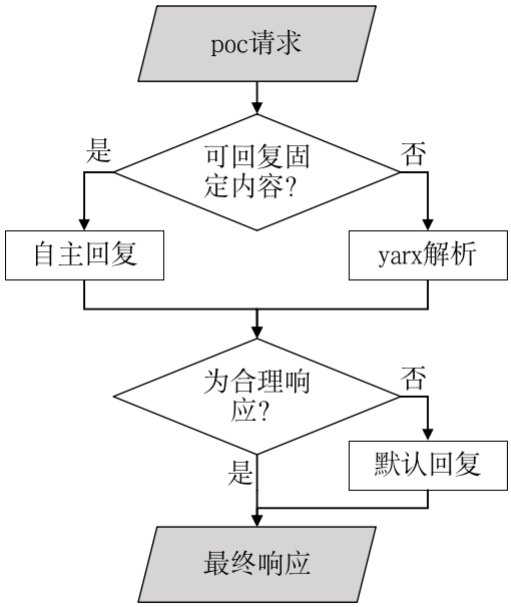


图 4.9 蜜罐针对 poc 请求的处理流程图

下面对上述三部分进行介绍。

4.2.3.1 自主回复

利用 flask 建立智能动态路由，对打过来的 poc 分析其请求类型和路径参数。针对转发过程中产生的 url 转义等问题也进行了优化。

经过统计分析我们发现，在全部 poc 中 bcontains 和 bmatchs 这两个判断函数的出现占了绝大部分情况，所以 poc 的结果判断存在一定的漏洞。因此针对较为简单的 poc 请求，比如固定响应的，就直接进行回复。如 poc 中的 GET 请求，url 为“/common/download/resource?resource=/profile/../../../../../../etc/passwd”，只需回复 response.status == 200 && "root:[x*]:0:0:".bmatches(response.body)；针对“/common/download/resource?resource=/profile/../../../../Windows/win.ini”只需回复 response.status == 200 && response.body.bcontains(b"for 16-bit app support")即可为正确的响应。因此针对此类 poc 请求我们采用固定的回复函数进行响应。

此外，在后期解题过程中，由于时间原因，无法进行细致的请求划分，因此我们对 bcontains 字段和 bmatchs 字段进行了搜索集合，采用预定义字段库的方式进行回复。对于 yarx 解析失败的请求，默认返回带有预定义字段库的网页，如图 4.10 所示。

图 4.10 带有预定义字段库的网页

4.2.3.2 Yarx 解析

针对较为复杂的 poc 请求，我们将其转发至 yarx 平台，利用 yarx 平台的解析能力进行响应。如 poc 中 GET 请求“/?/member/cart/Fastpay=&shopid=-1%20union%20select%20md5(2029790292),2,4,7%20%20--+“，需要解析 url 并计算 2029790292 的 md5 值。因此，我们将其转发至 yarx 平台，然后根据 yarx 的响应构建更为合理的回复方式。

4.2.3.3 扩充 poc

根据分析，我们发现基础的 yarx 平台中针对 level2 的 poc 请求响应能力有限。主要是因为原有的 pocs 仓库中数量不足，泛化能力较弱。

因此，我们也利用 yarx 的规则解析功能，自定义编写了更多了基于 xray 规则的 YML 文件，以此扩充 yarx 的解析能力。

比如，针对 GET 请求中 url 为“/comment/api/index.php?gid=1&page=2&rlist[]=*hex/@eval($_GET[1234])%3B%3F%3E“的 poc。原有的 yarx 中无法解析“_GET[1234]”因为规则库中对应的参数为“_GET[_]”。而在我们的扩充规则库中，将其改为“_GET[{{r}}]”，设为动态变量，这样 yarx 就可以具备更泛化的 poc 解析能力。

最终，我们在原有规则库中增加了约 150 个 poc 规则 YML 文件。有效提高了标靶环境的响应能力。

4.2.4 结果展示

基于上述的步骤构建标靶环境并非一簇而就，而是在比赛过程中逐步优化，不断提高的。图 4.11 为蜜罐环境在不断优化过程中提交分析结果所得到的成绩变化，可以看到随着各项服务的不断完善，得分整体也在不断上升。但总的来说还是存在提高的空间。比如可以借助更多自动化的工具和平台，借鉴其他的 poc 规则库和蜜罐环境，进一步提升处理不同 poc 的效率和完备性，相信相比于自己从头构建会更加完善和高效。但由于时间关系等原因，本次构建过程尚存在不足。

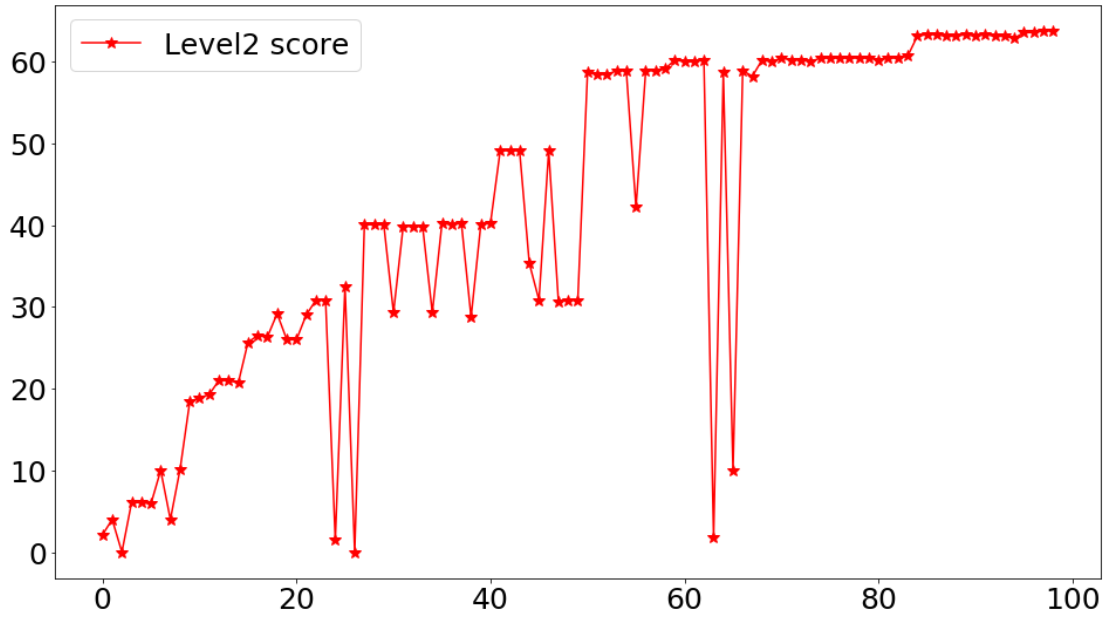


图 4.11 蜜罐环境得分趋势图

五、 总结与展望

本次 Datacon2022 大数据安全分析竞赛大大加深了我们对相关模块的了解。在我们选择的网络流量分析中包含了挖矿流量检测与蜜罐环境构建。在实践中检验和锻炼了分析数据和构建蜜罐解决实际安全问题的能力，让我们从头到尾全链路体验了挖矿流量和蜜罐的构建过程和分析过程。同时让我们认识到工具的重要性。总的来说，这是一次非常不错的学习经历。希望后续能有更多的学习机会。

参考文献

[1] Veselý, Vladimír and Martin Zádník. “How to detect cryptocurrency miners? By traffic forensics!” Digit. Investig. 31 (2019): n. pag.

[2] Caprolu, Maurantonio et al. “Cryptomining makes noise: Detecting cryptojacking via Machine Learning.” Comput. Commun. 171 (2021): 126-139.

[3] Hernandez-Suarez, Aldo et al. “Detecting Cryptojacking Web Threats: An Approach with Autoencoders and Deep Dense Neural Networks.” Applied Sciences (2022): n. pag.

[4] Zhang, Shize et al. “MineHunter: A Practical Cryptomining Traffic Detection Algorithm Based on Time Series Tracking.” Annual Computer Security Applications Conference (2021): n. pag.

[5] https://github.com/chaitin/xray

[6] https://github.com/wen-dg/qscan

[7] https://github.com/zema1/yarx

[8] https://www.reliaquest.com/blog/mining-for-better-threat-intelligence-cryptominer-pools/

[9] https://poc.xray.cool/