Keyword Extraction for Fine-Grained IoT Device Identification

Ashley Andrews
Electrical & Electronic Engineering
University of Bristol
Bristol, United Kingdom
ash.andrews@bristol.ac.uk

George Oikonomou
Electrical & Electronic Engineering
University of Bristol
Bristol, United Kingdom
george.oikonomou@bristol.ac.uk

Simon Armour
Electrical & Electronic Engineering
University of Bristol
Bristol, United Kingdom
simon.armour@bristol.ac.uk

Paul Thomas
Electrical & Electronic Engineering
University of Bristol
Bristol, United Kingdom
paul.thomas@bristol.ac.uk

Thomas Cattermole
Computer Science
University College London
London, United Kingdom
thomas.cattermole.17@ucl.ac.uk

***Abstract*— Internet of Things (IoT) devices are becoming more widespread in networks and are shown to have security considerations as an afterthought. Identifying IoT devices can help users locate security vulnerabilities in their networks. Previous studies have used machine learning and rule-based methods to try and identify unknown devices from passive network traffic. The first issue with these approaches however is that the device must have been seen on a training dataset beforehand; otherwise it cannot be identified. The second issue is that trying to achieve granularity on device identification down to firmware level from passive network traffic has not been researched before, and is a key factor in identifying vulnerable devices. This paper contains a novel technique to solve those two problems. The technique automatically identifies unknown devices from passive network traffic without using a machine learning approach that finds and weights keywords found in each packet per device. These keywords then allow device identification down to a specific firmware version. The approach in this paper achieved 71% accuracy for identifying firmware versions and 74% and 78% for models and makes respectively, across a test dataset of 44 devices.**

Keywords*— Internet of Things (IoT), Machine Learning (ML), Passive Network Traffic Analysis, Firmware Versions, Device Identification, Models, Keywords

I. Introduction

The 'Internet of Things' (IoT) has a broad definition, but the ITU-T has defined it as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies" [1]. When discussing IoT devices, security is an afterthought, with time-to-market and production costs taking priority over prudent security practices [3], and therefore there are many fairly trivial attack vectors for IoT devices. For example, an innocuous coffee maker was shown to be vulnerable to a modified firmware attack [4]. There is value in accurately identifying all devices on a network. This enables users to understand their estate and whether there are devices that should not be connected. More granular information, e.g., identifying exact models and firmware versions, further allows the user to identify devices that may be vulnerable.

While fingerprinting approaches can be used as a reactive measure once a device has been shown to be acting maliciously [5], [6], ideally they should be used preventatively, before a device has done any damage.

Meidan et al. [2] defines an IoT device by the following 4 properties:

1. Type,
2. Make/Manufacturer,
3. Model name/number,
4. Firmware version.

Previous studies have focussed on identifying devices using properties 1 to 3 but neglected property 4. We discover to what extent it is possible to identify devices fully, and therefore this paper contributes in the following ways:

- We propose a novel device identification method based on extracting keywords from passive network traffic that does not rely on machine learning techniques;
- We demonstrate that our method can detect device firmware versions with 71% accuracy;
- We highlight the challenges involved in detecting device firmware versions.

II. Previous Work

A. Prior Art for Device Identification in General

There are two broad methods for identifying devices on a network: active and passive. An active method probes a device by sending it packets to see how it responds. A passive method analyses the traffic sent to and from a device. Although there is literature [9] and products on the market [7], [8] that use active methods, the ideal solution is to use passive methods. This is because passive methods do not add load to the network and they do not risk crashing a device by sending it packets it does not know how to respond to. For these reasons, we choose to apply a passive method for device identification in this paper.

For a passive method, there are four types of analysable traffic:

- Device broadcast/multicast traffic, inside the LAN. This traffic is generated by a device and goes to every single device,
- Device to device traffic, inside the LAN,
- Device to the internet traffic, through a router with NAT,
- Device traffic outside the NAT boundary.

It is worth noting here however that not every device generates all these types of traffic. IoT devices are typically resource constrained and made to do a specific task, and therefore may not need to generate lots of traffic. Another

consideration is the size of the dataset used in the study. A larger dataset with good results is likely to be more transferable between devices and network setups.

There are a few techniques that can be used to try and classify a device on a network from passive network traffic. For any method the following three properties should apply [10]:

- The fingerprint has to be unique for the device;
- The features need to be hard to forge;
- The fingerprinting method should remain stable even when devices move from one network to another.

OUI lookups on the MAC address of a device are also used to identify devices [12]. This approach however is naïve as it will determine the manufacturer of the Network Interface Card (NIC) as opposed to the manufacturer of the device.

Another fairly new approach is looking at 'context aware' devices, i.e. adding sensors to devices and using readings such as temperatures as new features [13], but this use case is more towards a device acting abnormally as opposed to identifying it, as setup would require finding the device and adding sensors.

These final two methods are therefore not considered, giving rise to just three main techniques in use by studies:

1) Rule-based Approach

A rule-based approach is where pre-determined rules are set up based on previous knowledge of how a device has been seen to behave. The benefit of a rule-based approach is that a rule can be quite specific and is based on well-known ground truth of how a device behaves. The issue here however is the sheer number of IoT devices available that would all need specific rules, and also the fact that as devices get new firmware updates over time the rules may no longer be valid. Feng et al. [14] looks to solve this by automatically generating rules for IoT devices without human effort or training, by extracting passive keywords and comparing this to entries on various e-commerce sites. The granularity for this study does not go down to version however, only the type, make and model.

2) Supervised Machine Learning Approach

The main approach used in various studies is to apply supervised machine learning algorithms, such as a random forest, to a dataset after extracting various features. The features used can be statistical (e.g., number of packets in a time window) [15 - 18] or non-statistical (e.g., port numbers used) [10], [11], or a combination of both [19]. Accuracy results from this method are generally high (> 90%) for identification down to device models. The limitations of these studies however are that a device must have been seen before in the training set to be correctly identified. With the vast array of IoT devices available, this is generally not feasible for granularity more than a 'device type' when dealing with a lot of devices. This approach is therefore not a focus of this paper.

3) Unsupervised Machine Learning Approach

The other approach that can be used is an unsupervised machine learning approach, using algorithms to try and find common patterns in the data that can be used for classification, such as [20] and [21]. This does rely on having good features that are able to distinguish between devices that may do similar things.

B. Specific to Keyword Extraction

Previous studies into keyword extraction can give an indication to the protocols and fields that are likely to be unencrypted, and how they can be used. The main traffic used in these studies includes broadcast traffic such as mDNS and SSDP [12], [22 - 24], DHCP [12], [22- 24], HTTP (User Agents & URI) [14], [24 - 26], DNS [24, 26] and packet content [27]. These protocols are all shown to give unencrypted information that is useful for analysis. When extracted, keywords should be filtered to make them both more useful and unique [12], [24], [25]. Another technique is to combine two protocols to identify a device between manufacturers. For example, an Apple iPhone and an iPad could have the same mDNS messages and very different SSDP [22].

One machine learning algorithm using Natural Language Processing (NLP) that can be used is Term Frequency Inverse Document Frequency (TF-IDF). This is used to give an indication of how unique a word is within a 'document', in relation to all documents, so therefore has a use in device identification. Many studies use TF-IDF on different keywords including the TLS Handshake [28], HTTP [14], [25] and DNS [26]. The results show this is a valid technique although the accuracy within the results (down to vendor and model) can vary quite a lot. Although [23] generates data using active scanning, the techniques can still be analysed. TF-IDF is run across mDNS, UPnP and HTTP to give a bag of words to feed into a Random Forest (RF) algorithm. The main issue with TF-IDF and other ML algorithms is the time taken to generate the training data.

The technique introduced by Xu et al. [24] uses keywords extracted from protocols which are then manually input as searches to e-commerce sites. The results are hard to re-produce, and the granularity in best cases is a model. Khandait et al. [27] also states how similar devices have the same keywords within the payload. It is worth noting in all these studies that granularity down to a version is not considered except in [12] where an OS version is given for some but not all devices (mainly mobile) and does not identify but rather states each device has a unique feature vector for future ML techniques [29]. Previous studies also do not always identify every device accurately. Wang et al. [25] states how if the same base firmware is used on devices, the algorithm cannot tell the difference.

The technique in this paper fills these gaps in the literature, getting device identification granularity down to a version while still being accurate and efficient in terms of processing time.

III. Method

Our proposed system collects traffic data from the IoT lab using a sniffer and then analyses protocols in order to extract keywords to identify devices. The architecture of the system is shown in Fig. 1.

A. Lab Setup

For this study, although general IoT traffic datasets exist [16], [30], [31], no datasets existed publicly that contained

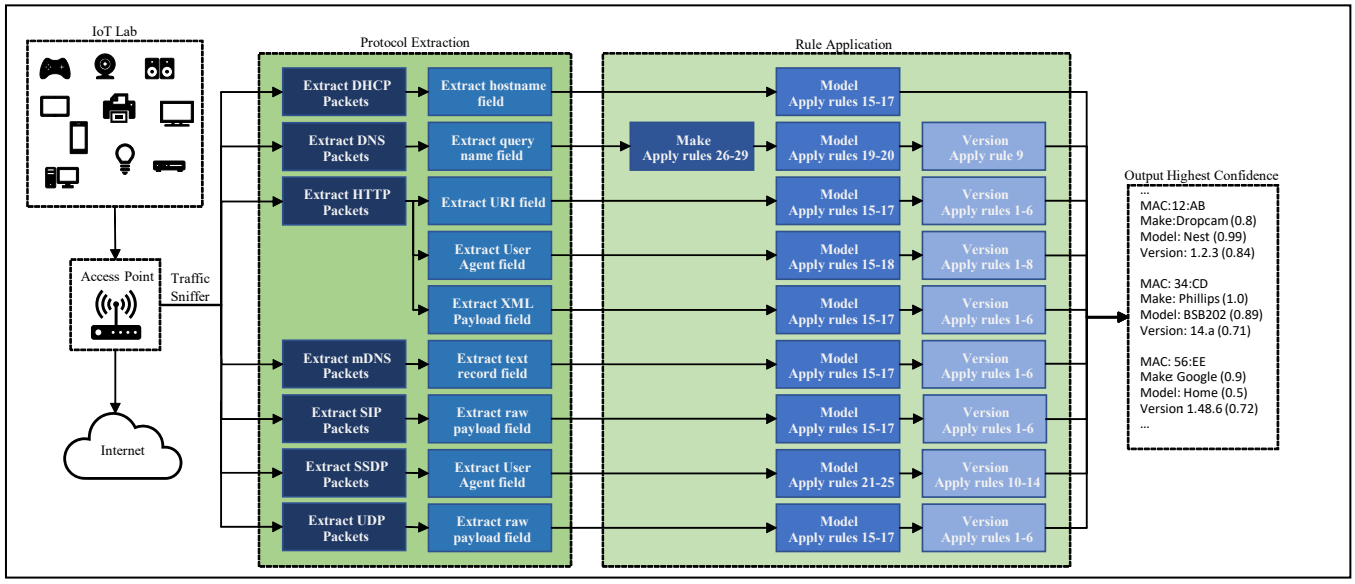

Fig. 1. Architecture of the system. Normal traffic from the IoT lab is collected by the Sniffer, keywords are extracted from specific protocol fields, and a number of rules are applied depending on the classification type. The system outputs classification predictions along with their confidences.

device firmware versions with corresponding timestamps. [16], [31], are also based only on the device setup phase, limiting their usefulness for this study. Due to these restrictions, a lab had to be set up with a range of devices, and some manual monitoring performed to periodically check device firmware versions. This did however allow for a larger and more varied dataset in terms of the number and types of devices (this lab contained 44 devices) than is available publicly, allowing a richer dataset and results. With IoT devices, 'auto updates' are normally turned on, hence traffic for multiple versions will be captured. The lab captures all network traffic 24/7 and includes devices in idle mode and also those stimulated by user interaction. Datasets and device lists are available upon request to the authors.

B. Method Description

Seven different protocols were used that are either broadcast traffic or protocols with fields known to be unencrypted, as we are looking to pick out keywords from the traffic, building on previous work. Our approach here is not a computationally expensive machine learning technique. We only look to process unique packets within the dataset and the unique packets are then parsed for the protocols listed in this section to extract any keywords, filter them using a regex for likely model or version words and then apply rules shown in TABLE II to give a 'confidence score' with weightings towards certain words. The weighting values in TABLE II. were created through a combination of domain knowledge for how version and model strings typically look and experimentation. These are explained in the 'reasoning' column in the table. A count is also provided to see how many times the string appears in each unique packet. This way we can get an overview of the dataset and save processing power by not using computationally expensive machine learning techniques. It also gives the benefit of not needing to have a dataset to train on, and therefore the technique can deal with unseen devices. The following explains how each protocol is parsed to give the output results.

1) *DHCP (Hostname)*

DHCP hostname is normally set by the user, and so the results of this protocol are mainly to reinforce other predictions. Each DHCP record is extracted and then rules 15-17 are applied to attempt to identify models only.

2) *DNS (Query Name)*

Extracted DNS results are split by the '.' delimiter, and filtering is applied to remove words common to the English language and any digits. The Top Level Domain (TLD) is then used as a potential make with rules 26-29 applied. Potential models are also considered using rules 19 and 20. The specific case of Amazon devices only use rule 9, as they can show version information in DNS requests.

3) *HTTP (XML Payload)*

The XML is first parsed to split into keyword pairs based on the XML tags, so the output is similar to mDNS. The regex is then applied for makes, models and versions to obtain suitable potential keywords. These are then input to rules 1-6 to weight versions and 15-17 to weight models.

4) *HTTP (Request URI)*

For the request URI, unexpected characters are first filtered out. The remaining URL is split by the '/' delimiter, and each one parsed for versions using rules 1-6 and 15-17 for models – the same as HTTP XML. The difficulty here is knowing whether a version string is current or related to an update package check.

5) *HTTP (User Agent)*

HTTP User agents should follow a set standard in theory, but in practice this is rarely the case. For HTTP user agents, the format should be "<Product>/<Product Version> (<system-information>) <platform> (<platform-details>) <extensions>". This is the basis for the parsing and subsequent rules being applied. For versions we apply rules 1-8, and for models 15-18.

6) *mDNS*

As mDNS records should be keypairs as per the RFC [32], the algorithm first gets these key/value pairs from the dataset. From these, the regex is applied to the output to obtain words that could be related to models or versions. For versions, rules 1-6 are applied, and 15-17 are applied for models.

7) *SSDP (Server & User Agent)*

In theory, SSDP packets should also follow the RFC: and the server and user agent fields be in the format "<OS>/<version> UPnP/1.1 <product>/<version>" and optional "(<model>)" at the end. The parser therefore attempts to split the string based on this and checks each of the 4 elements in turn. The uPnP element is removed, and the weightings applied are based on how well the string matches

the RFC. Rules 10-14 are applied to versions, and 21-25 applied to models.

8) SIP & UDP (Raw Payload)

Session Initialisation Protocol (SIP) is a protocol used mainly for voice and video applications. For this protocol and UDP, the payload is just raw bytes. It initially needs to be processed to find only human-readable strings as these could be keywords. These strings are then passed to rules 1-6 for versions and 15-17 for models.

IV. Results & Discussion

A. Overall results

The overall results were verified by looking at the outputs that gave keywords and comparing them to the known ground truth. For the devices, not every protocol was transmitted for every device, and some devices are very quiet on the network generating very little or no traffic. For cases where no keywords were found, we mark these accordingly and they are not used for the final accuracy score.

For visualisation of the results, we use Kibana to output the keywords per MAC address in a word cloud using the confidence score as the relative size. If the keywords were clear and obvious we mark them as correct, as shown with the PlayStation 4 device running firmware version 8.03 in Fig. 3. If there is not one clear result but the correct value is within the keywords, we mark the result as 'inferred', as shown with the Google Home device running firmware version 1.54.250118 in Fig. 2. We class both these results as being an acceptable correct output in our results. The summary results are shown in TABLE I. and are visualized in the graphs in Fig. 4. Of the 44 devices in the test dataset, 32 makes, 32 models and 22 versions were correctly identified. The overarching theme here is that combining information from each protocol can give a rich amount of information. The following sections will provide an analysis of how each protocol performed for the identification process, with examples specific to our dataset.

Fig. 2. Kibana output for the inferred Google Home device running firmware version 1.54.250118.

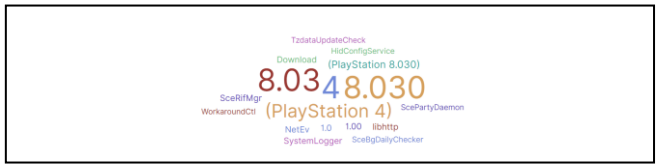

Fig. 3. Kibana output for the PlayStation 4 device running firmware version 8.03.

TABLE I. Results

	Correct	Inferred	Incorrect	No Keywords	Accuracy
Make	29	3	9	3	0.78
Model	27	5	11	1	0.74
Version	12	10	9	13	0.71

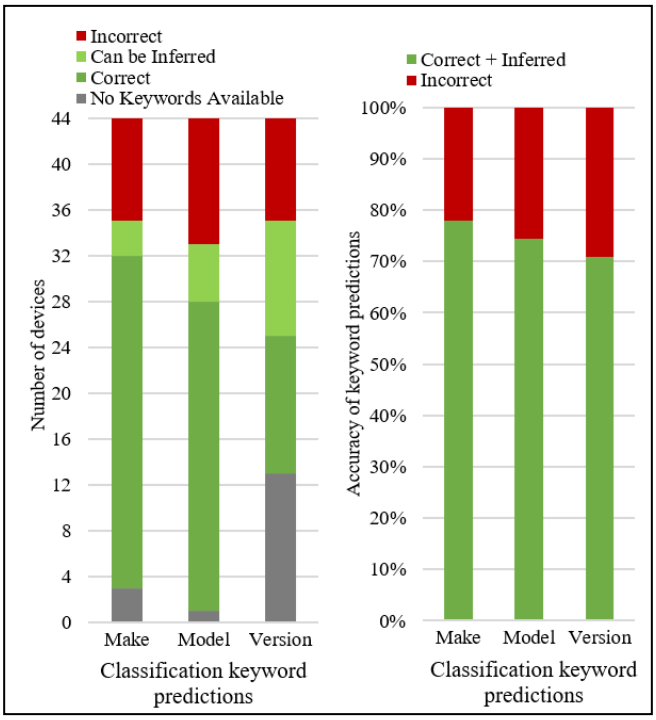

Fig. 4. Overall results showing keyword predictions across dataset (left) and accuracy of predicted keywords (right).

B. Identifying Versions

Version extraction was the main focus of this study, as a novel area that had not been explored before. The experiments showed that this was a difficult problem for the following reasons. First, it was shown that RFC standards for protocols were not always followed. Also, some devices intentionally hide version strings, possibly based on Mozilla recommending not giving too much version information away in user agents [33]. It was also shown that there was no commonality on methods to identify different versions on the same device, and also some versions strings in the packets looked similar, but not exactly the same. The second point is related to the actual version string. There is generally no common nomenclature for version strings for devices (although typically it would be "major.minor.patch" format) and therefore it is up to a manufacturer to decide on a scheme. The device may also be running modified firmware based on an underlying embedded Linux or Android operating system, and therefore two versions (the kernel and the device) would both be valid. The version string also may not be for the device but for the protocol, and therefore a technique would need to be 'context aware' whether it is likely to be for the device or the protocol. The third is related to the protocols and packets generated by a device. There is no widely accepted protocol that is used to specify a device version. Also, this technique relies on devices generating some plaintext traffic, which some devices do not. That being said, the following outputs were shown per protocol, showing this technique to be beneficial. It was found that in the case of certain Amazon devices, the version string was shown in a DNS request. The string 'andr-22-amazon-aeokn-104666user669702420' was shown, with '669702420' being the version. This is likely from an update so could be a version the device wants to be, but in any case it shows the device is *not on latest* firmware, or has just become latest. This was not on every Amazon device, however. For the

mDNS protocol, routers, switches and NASs gave verbose data including exact firmware versions of devices. These would be useful to secure, as in the case of switches and routers a vulnerable device could allow an attacker either into the network, or the ability to propagate further into the network. With NASs, they are likely used to store sensitive data and so would also want to be secured. The one false positive however was D-Link devices, which gave a version which looked like it could be for the device, but was actually for the HNAP protocol, showing the difficulties when trying to solve this problem. The HTTP User Agent field was able to identify devices and versions for tablets and mobile phones with good success, and also the PlayStation 4 games console. This generally is also verbose, giving the device version, but also the underlying OS and build. This does however require some correct parsing to ensure that the browser version is not selected. In contrast, HTTP (XML Payload) was not useful to identify versions. Although version strings were picked up, it was on a very small number of devices, and the version related more to the XML and protocol versions than the device. HTTP Request URI was clearly picking up updates, so again it can be implied that devices were *at least not on latest* firmware, but possibly do auto-update. This applied to both Sonos smart speakers and the Samsung TVs. Where the RFC was followed, SSDP gave versions for a smart speaker and a smart plug. The SSDP Server field was again not overly useful as it was not specific to the device. The final protocols parsed were UDP and SIP. This allowed identification of a Ring Doorbell version (to the best of the author's knowledge as Ring do not publish their versions) and also the AppKettle device. With the AppKettle device too, an update was shown to happen as the version changes during the capture, showing the device likely had automatic updates turned on, implying good security practices. The overarching conclusion here is that in isolation the keywords can give some indication of models and versions, but when combined more devices can be identified and confidence can increase on predictions.

C. Analysis of Identifying Makes & Models

Identifying makes and models has been the focus of many studies, but the method proposed in this paper still has merit here for the various protocols. We use the DHCP results to re-enforce other classifications, rather than classify devices in their own right for the following reason: DHCP hostname is a value that can generally be set by the user easily, and also changed easily. There is no requirement to set the name to anything related to the device, but generally this would be the case. It can still however provide some use in giving confidence to other model predictions from other protocols if there are similarities. For example, the DHCP name "Galaxy-Tab-A-80-2019" is very useful, yet 'android-d8899ec92163a09c' (a Google Nexus 7 in our dataset) is less so. DNS Results can identify a make, when weighting and counts are applied. Taking the Philips Hue device in the dataset, the top DNS results are 'philips' and 'meethue' jointly, giving an indication of a manufacturer. When combined with a model which gave many potential keywords with the same weight, such as "philips-hue", "baidu", "mqtt", putting the two together gives good confidence the device is a Philips Hue device. This method however is not great for a device that utilises a browser, such as a tablet or mobile phone, as many different URLs can be visited that are not related to a device. Although HTTP (XML Payload) was not seen much, the parser still found keyword pairs, and so good information could be determined. With weightings applied, one device had the most common entry 'SM-T290' which was the specific model. This can be combined with the DHCP results which stated it was a 'Galaxy-Tab-A-80-2019' which is a match. This again shows that combining multiple sources of data can give confidence on a device. HTTP URI was not seen much, but where it was seen and weighting applied, then models could be identified quite clearly. This was true for D-Link devices (DCH-G020, DCH-S150-A1) and again mobile and tablet devices (SM-T290). The device's hits are likely from the device reaching out to an update URL and stating what device it is. The HTTP User Agent was the most prevalent traffic found in this study. With weighting and rules applied, an array of devices could be identified, such as televisions (UE32J5500), mobile devices (iPad 4, SM-T290, Nexus 7), hubs (Amazon AEOKN and AEOSN) and smart plugs (TP-Link HS110). Although TLS is implemented for devices, there is still HTTP traffic and inside the LAN boundary this is likely to continue. mDNS keys are able to be set by a manufacturer, but in this study the devices that broadcast mDNS had model keys that matched a regex. This meant all the DLink devices (DSP-W215 etc.) as well as others (HealthyHomeCoach, Arlo VMB4000, VMB5000) were identified. This protocol was mainly found to give good information to network infrastructure devices such as routers, switches and NASs. The SSDP Server field was not overly useful, as it was normally generic. The user agent however gave more information, although it was found that device developers did not always follow the RFC, so parsing was more difficult. It did still however pick up some devices such as the Sonos ZPS3 smart speaker. The final protocols were UDP and SIP. These were both parsed in the same way, although it was difficult to get results even with good keywords contained. This was due to UDP parsing into acceptable strings, which could all theoretically be a device model. A 'User Agent' could be extracted from SIP however to give a ring doorbell device.

V. Conclusions And Future Work

This research has shown that keywords from passive network packets is a method that can identify models well, although trying to identify device versions from passive network traffic is a difficult problem to solve, for 3 main reasons. The first is related to the development process of the device itself, whether intentional or unintentional by the developers. The second point is related to the actual version string, and how there is no common format to it and no common place to find it within a network packet. The third is related to the protocols and packets generated by a device, and how this method will rely on the traffic being unencrypted. Due to these 3 main points, it is therefore accepted that this would not be a 'fits all' solution, but rather a method to complement existing machine learning techniques for device identification. This could be part of a first stage machine learning identification solution to identify some of the more obvious devices or devices not seen before, or used further in a processing chain to reinforce results. Despite these 3 challenges, the method proposed achieved an accuracy of 78% for makes, 74% for models, and 71% for versions.

As research into identifying device versions is in its infancy, the following three pieces of future work could be explored. The first bit of research would be to use Markov chains to automatically generate rules based on how other version strings look, and try to spot new ones in traffic, as well as having a method to auto-generate rules and weights. Second, the identification method from this study could be used as one stage in multi-stage device identification algorithms, either as a first stage to try to identify 'low hanging fruit' or a further stage to reinforce other results. To complement this the experiments themselves could also be expanded to follow other literature using the same (keywords) or different (statistical) extracted features and ML algorithms but with datasets of the same devices on different versions, which has not been done before, to the best of the authors knowledge. The third piece of work could be related to the behavioural analysis of devices – trying to spot devices updating, for example from viewing the HTTP URI information and using that to work out versions, or at least identifying a device trying to update, and therefore potentially vulnerable as it is not running the latest version device firmware.

REFERENCES

[1] ITU (International Telecommunications Union), Measuring the Information Society Report 2015. 2015.

[2] Y. Meidan, V. Sachidananda, H. Peng, R. Sagron, Y. Elovici, and A. Shabtai, "A novel approach for detecting vulnerable IoT devices connected behind a home NAT," *Computers and Security*, vol. 97, p. 101968, 2020, doi: 10.1016/j.cose.2020.101968.

[3] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "SoK: Security Evaluation of Home-Based IoT Deployments," *Proceedings - IEEE Symposium on Security and Privacy*, 2019, doi: 10.1109/SP.2019.00013.

[4] M. Hron, "The Fresh Smell of ransomed coffee," 2020. https://decoded.avast.io/martinhron/the-fresh-smell-of-ransomed-coffee/ (accessed Sep. 20, 2022).

[5] T. Li, Z. Hong, and L. Yu, "Machine Learning-based Intrusion Detection for IoT Devices in Smart Home," *IEEE International Conference on Control and Automation, ICCA*, vol. 2020-Octob, pp. 277–282, 2020, doi: 10.1109/ICCA51439.2020.9264406.

[6] R. Kumar and T. Kaur, "Machine Learning based Traffic Classification using Low Level Features and Statistical Analysis," *International Journal of Computer Applications*, vol. 108, no. 12, pp. 6–13, 2014, doi: 10.5120/18961-0290.

[7] "Fing", "Fing." *www.fing.com* (accessed Sep. 20, 2022).

[8] "Firmalyzer", "Firmalyzer IoTVas." *https://www.firmalyzer.com/iotvas-api/* (accessed Sep. 21, 2022).

[9] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, "Can We Classify an IoT Device using TCP Port Scan?," 2018. doi: 10.1109/ICIAFS.2018.8913346.

[10] R. R. Chowdhury, S. Aneja, N. Aneja, and E. Abas, "Network Traffic Analysis based IoT Device Identification," *ACM International Conference Proceeding Series*, pp. 79–89, 2020, doi: 10.1145/3421537.3421545.

[11] A. Aksoy and M. H. Gunes, "Automated IoT Device Identification using Network Traffic," *IEEE International Conference on Communications*, vol. 2019-May, 2019, doi: 10.1109/ICC.2019.8761559.

[12] N. Ammar, L. Noirie, and S. Tixeuil, "Network-protocol-based IoT device identification," *2019 4th International Conference on Fog and Mobile Edge Computing, FMEC 2019*, no. Section V, pp. 204–209, 2019, doi: 10.1109/FMEC.2019.8795318.

[13] N. Yousefnezhad, M. Madhikermi, and K. Framling, "MeDI: Measurement-based Device Identification Framework for Internet of Things," *Proceedings - IEEE 16th International Conference on Industrial Informatics, INDIN 2018*, pp. 95–100, 2018, doi: 10.1109/INDIN.2018.8472080.

[14] X. Feng, Q. Li, H. Wang, and L. Sun, "Acquisitional rule-based engine for discovering Internet-of-Thing devices," *Proceedings of the 27th USENIX Security Symposium*, pp. 327–341, 2018.

[15] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "IoT Devices Recognition Through Network Traffic Analysis," *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, pp. 5187–5192, 2019, doi: 10.1109/BigData.2018.8622243.

[16] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. R. Sadeghi, and S. Tarkoma, "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT," *Proceedings - International Conference on Distributed Computing Systems*, pp. 2177–2184, 2017, doi: 10.1109/ICDCS.2017.283.

[17] R. Keyte, "*Can IoT Devices be Identified Through Their Device Type Fingerprint ?*," Cardiff University, 2018.

[18] A. Acar *et al.*, "Peek-a-Boo: I see your smart home activities, even encrypted!," 2018, doi: 10.1145/3395351.3399421.

[19] A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2019, doi: 10.1109/TMC.2018.2866249.

[20] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, "Inferring IoT Device Types from Network Behavior Using Unsupervised Clustering," in *Proceedings - Conference on Local Computer Networks, LCN*, 2019, vol. 2019-Octob, pp. 230–233. doi: 10.1109/LCN44214.2019.8990797.

[21] S. Marchal, M. Miettinen, T. D. Nguyen, A. R. Sadeghi, and N. Asokan, "AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1402–1412, 2019, doi: 10.1109/JSAC.2019.2904364.

[22] L. Yu, B. Luo, J. Ma, Z. Zhou, and Q. Liu, "You are what you broadcast: Identification of mobile and iot devices from (public) WiFi," in *Proceedings of the 29th USENIX Security Symposium*, 2020, pp. 55–72.

[23] D. Kumar *et al.*, "All things considered: An analysis of IoT devices on home networks," in *Proceedings of the 28th USENIX Security Symposium*, 2019, pp. 1169–1185.

[24] Z. X. Xu, Q. yun Dai, G. Xu, H. Huang, X. B. Chen, and Y. X. Yang, *IoT Device Recognition Framework Based on Network Protocol Keyword Query*, vol. 12239 LNCS, no. 2019. Springer International Publishing, 2020. doi: 10.1007/978-3-030-57884-8_20.

[25] X. Wang, Y. Wang, X. Feng, H. Zhu, L. Sun, and Y. Zou, "IoTTracker: An enhanced engine for discovering internet-of-thing devices," *20th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks, WoWMoM 2019*, 2019, doi: 10.1109/WoWMoM.2019.8793012.

[26] R. Perdisci, T. Papastergiou, O. Alrawi, and M. Antonakakis, "IoTFinder: Efficient Large-Scale Identification of IoT Devices via Passive DNS Traffic Analysis," *Proceedings - 5th IEEE European Symposium on Security and Privacy, Euro S and P 2020*, pp. 474–489, 2020, doi: 10.1109/EuroSP48549.2020.00037.

[27] P. Khandait, N. Hubballi, and B. Mazumdar, "IoTHunter: IoT network traffic classification using device specific keywords," no. January 2020, 2021, doi: 10.1049/ntw2.12007.

[28] E. Valdez, D. Pendarakis, and H. Jamjoom, "How to discover IoT devices when network traffic is encrypted," *Proceedings - 2019 IEEE International Congress on Internet of Things, ICIOT 2019 - Part of the 2019 IEEE World Congress on Services*, pp. 17–24, 2019, doi: 10.1109/ICIOT.2019.00016.

[29] N. Ammar, L. Noirie, and S. Tixeuil, "Autonomous Identification of IoT Device Types based on a Supervised Classification," *IEEE International Conference on Communications*, vol. 2020-June, 2020, doi: 10.1109/ICC40277.2020.9148821.

[30] A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2019, doi: 10.1109/TMC.2018.2866249.

[31] A. Pitcher, "IoT Sentinel Dataset." *https://github.com/andypitcher/IoT_Sentinel* (accessed Sep. 20, 2022).

[32] IETF, "RFC 6763," rfc6763. *https://www.ietf.org/rfc/rfc6763.txt* (accessed Sep. 20, 2022).

[33] Mozilla, "Mozilla developer guidance." *https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Server* (accessed Sep. 20, 2022).

TABLE II. RULES

Rule No.	*Ruleset*	*Rule*	*Weighting Applied*	*Reasoning*
1	Version	If entry does not contain '.' and hits no other rules	x 4/5	A version usually contains a dot.
2	Version	If entry is 0	x 0	A device version is very unlikely to be 0.
3	Version	If entry starts with [A-Za-z!v!V] and contains only letters	x 1/20	A version is fairly unlikely to contain only letters, but not impossible.
4	Version	If entry starts with [A-Za-z!v!V] followed by numbers	x 1/5	A combination of letters and numbers could be a version, but usually would be just numbers.
5	Version	If entry starts with [vV0] and is value 1	x 1/10	A version of exactly 1 is probably unlikely for a released product that may receive patches.
6	Version	If entry starts with 1 and contains all [.0]	x 1/10	A version of exactly 1 is probably unlikely for a released product that may receive patches.
7	Version	If entry in browser list	x 1/100	The version is likely to be related to the browser, not the device.
8	Version	If the entry is inside the value within brackets	x 1/1000	The version is likely to be part of the model.
9	Version	If entry contains '-amazon-'	x 1	Amazon updates are part of DNS requests.
10	Version	If second element in entry does not contain 'UPnP'	x 1/10	The parser is expecting 'UPnP' so the developer hasn't fully followed the RFC.
11	Version	If element 1 exists in entry	x 1/4	Element 1 of SSDP can just be OS and corresponding version usually.
12	Version	If entry contains 3 elements, and element 3 has a '/'	x 1	Element 3 of SSDP contains a version after a '/' character.
13	Version	If entry contains 3 elements, and element 3 has no '/'	x 1/2	Element 3 of SSDP should contain a version after a '/' character so the developer hasn't fully followed the RFC.
14	Version	If after processing, the entry still contains 'UPnP'	x 2/5	If 'UPnP' is found it's likely still a valid SSDP entry, so can still be included.
15	Model	If entry length > 12	x 1/2	A device model on average should be less than 12 characters in length.
16	Model	If entry contains certain characters (;:,)	x 1/3	Certain special characters are unlikely to be included in a valid model.
17	Model	If entry contains all letters or numbers	x 1/10	Normally a model would contain both numbers and letters.
18	Model	If entry is in browser list	x 1/100	The model is likely to be a browser, not a device.
19	Model	If entry is in an English dictionary, or is a digit	x 0	The model would not be a normal word or a number.
20	Model	If entry length < 4	x 1/10	The model in DNS is likely to be a string at least of length 4.
21	Model	If entry second element does not contain 'UPnP'	x 10	The packet has been picked up as being UPnP, but doesn't state that it is, so may be non-IoT.
22	Model	If entry contains 4 elements	x 1	With 4 elements in SSDP, the 4th will be a model.
23	Model	If 3 elements, and element 3 has a '/'	x 1	Element 3 of SSDP contains a model before a '/' character.
24	Model	If 3 elements, and element 3 has no '/'	x 1	Element 3 of SSDP should contain a model before a '/' character so the developer hasn't fully followed the RFC.
25	Model	If after processing the entry still contains 'UPnP'	x 2/5	The packet is UPnP but the RFC has not been followed, although it is likely a model will be before the UPnP string.
26	Make	If no rules are hit	x 1	No rules triggered therefore we can have high confidence.
27	Make	If entry contains a cloud provider	x 1/10	The device is likely utilising a cloud provider, which is not the manufacturer.
28	Make	If entry length < 4	x 1/10	A make is unlikely to be less than 4 characters in length.
29	Make	If entry is not alphanumeric	x 1/2	A make is generally alphanumeric.