JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Design of Firmware Updation System in IoTDevices using DNS

[†]Praveen Misra, Rajeev Yadav

Abstract*—The Internet of Things (IoT) has completely changed computing thanks to the plethora of applications that have been developed around different kinds of sensors. This comprises a staggering array of items of all sizes and forms, ranging from intelligent microwaves that autonomously cook meals to self-driving cars with sophisticated sensors that identify objects in their route. With predictions of close to a Trillion smart devices, there is already a significant amount of activity in IoT-based product lines, and this engagement is anticipated to increase in the decades to come. The security of these resource-constrained and frequently remotely placed IoT devices becomes crucial given the expected exponential expansion in their number. The majority of IoT devices have factory-installed firmware. IoT devices' firmware is typically not updated during the course of their lifespan. As a result, they are more vulnerable to attacks, which could cause serious issues. The potential threat for IoT devices can be significantly decreased by regularly updating the firmware. One of the building blocks of the internet is the domain name system, which is primarily used to store and obtain IP addresses from web addresses. Different record types supported by DNS, such as A, AAAA, etc., hold various forms of data about the requested domain name. The OX record type is one of the recently suggested record types. The goal of the study is to determine whether the OX record type and its numerous accessible fields are appropriate for securely updating the firmware of IoT devices. The globe is gradually switching from the outdated IPv4 protocol to the new IPv6 protocol as the number of gadgets connected to the internet rises. As a result,this project is mostly being deployed for IoT devices that use IPv6.

***Index Terms*—Domain name system, firmware updation, Internet of Things, IPv6, security in IoT devices, global identifier system, firmware vulnerability, smart devices, constrained hardware security**

I. INTRODUCTION

The IoT devices are referred to as restricted devices since they have limited processor, memory, and power resources. The Internet Engineering Task Force (IETF) assigns limited devices a class, ranging from Class 0 (lowest) to Class 2 (highest), based on their computational capability. Some IoT devices might be able to use conventional internet protocols, but others may not be able to do so at all due to their resource limitations. In order to better accommodate their limits, these limited nodes design an IoT protocol stack as shown in Figure 1. Constrained devices on a limited network must use a gateway node in order to connect to the network's hosts that uses a complete TCP/IP stack. Smart gadget IoT gateways can be created on smartphones. As a gateway between the limited network and the Internet, there are also specialized devices. The TCP/IP stack and the IoT stack are connected by these gateways.

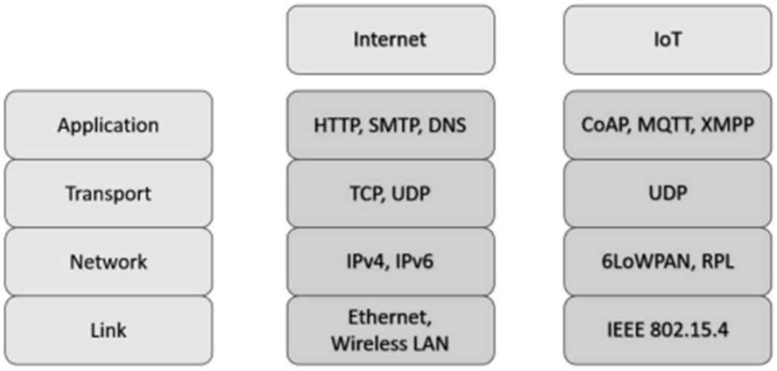

Figure 1: IoT Stack

We have IEEE standard 802.15.4 at the bottom of the IoT protocol stack. It offers a few-meter-long, low-speed Wireless Personal Area Network (WPAN). Devices can communicate with one another and with neighboring devices by using IEEE 802.15.4 without the assistance of additional infrastructure.
A low powered network based on IEEE 802.15.4 may send and receive IPv6 packages thanks to the IPv6 via reduced power WPAN (6LoWPAN) protocol. By enabling IPv6 header compression and encapsulation, this is made possible. Each node in the 6LoWPAN network is capable of sending data to another node, which is known as mesh routing. Since they can interact by utilizing other nodes as

relays, this makes it simple to add servers anywhere within the network.
The User Datagram Protocol (UDP), a lightweight protocol, is implemented at the transport layer of the Internet of Things protocol stack because limited nodes have found it to be acceptable. The ideal protocol for restricted devices is UDP due to its speed, despite the fact that TCP is more dependable than UDP. Because UDP doesn't have error correction and has a simple

configuration, fast communication is made possible. Some of the missing TCP functionalities in the application layer are implemented by some IoT protocols, like the Constrained Application Protocol (CoAP). [1]
Several application layer protocols in the Internet of Things protocol stack, including CoAP, MQTT (aka Messaging Queuing Telemetry Transport), and Extensible Messaging and Presence Protocol, have been created to be usable by restricted devices (XMPP). Although these protocols operate in various ways, they can all be employed by compact and light devices.
Most Internet users require the Domain Name System (DNS), an UDP-based application protocol. Even though its functionality was different at the time, DNS has existed since 1983 on the ARPANET. A hostname, such as a website address, is resolved by DNS, which then converts it to the IP address at which the webpage is maintained or situated. The act of entering a hostname into a web browser, such as www.example.se, is a common example that most individuals are familiar with and can relate to. A DNS query is a request sent by the web browser. There are a number of various ways to submit a query; a user is not required to type a website link into a web browser.

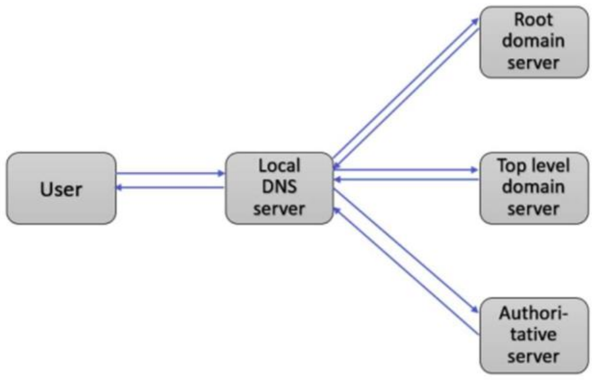

Figure 2: DNS query flow

When a web URL is entered into a browser, a DNS query is issued. This is what will be demonstrated in the example that follows. Initially, the neighborhood DNS server receives the request. The user's computer is frequently close to this server, either on the same network or a few router hops away. The neighborhood DNS server won't be capable of locating this hostname on its own if it is asked to do so for the first time. A main domain server receives the request instead of the neighborhood DNS servers. The complete status of the request is depicted in Figure 2.
The server addresses for top-level domains (TLDs) can be resolved by the root domain servers. 13 main name servers are scattered throughout the globe. However, for security and dependability concerns, these servers do have replicas. The "A" root server's address is "a.servers.in," and the root domain servers are numbered "A" through "M". All root domain servers must adhere to the naming scheme. Five billion inquiries a day are sent to DNS root server "A."
The leading name server information is returned by the root server in response to the local DNS server's request for it from the root domain servers. Domain extensions like ".com," ".edu," ".org," and ".se" are examples of TLDs. When this happens, the source node will resolve the ".se" portion and relay the answer to the neighboring DNS server. Now, the ".se"-TLD server canreceive the request from this server and respond. The "example.se" portion of the address is resolved by the TLD server, and this information is then forwarded to the neighborhood DNS server.

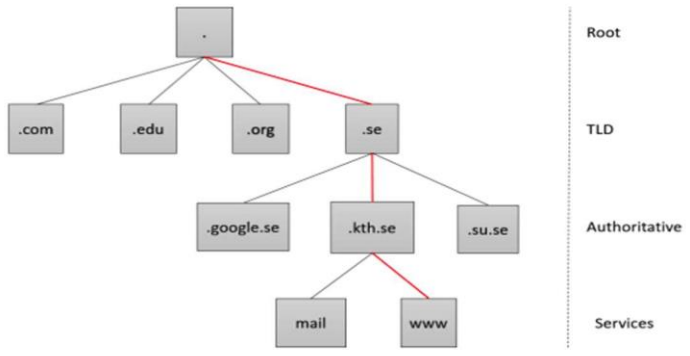

Figure 3: DNS hierarchical architecture

The request can now be sent to the authoritative server of example.se by the neighborhood DNS server. Every organization that makes its hostname publicly available has an authorized Name server that resolves the addresses of the organization. To fully resolve the requested host name www.example.se, one last step must be carried out. When a DNS query is made, www.example.se's

authoritative DNS server answers with the website's IP address. The neighborhood Domain name server receives the response and passes it

to the user, who can then retrieve the webpage. As shown in Figure 3, the operation of this system is hierarchical.
The IP-address can be retained in the memory location of the local DNS server once the entire query has been answered and it has received the IP-address. The resolution has a Time to Live (TTL). In case there are network changes, such as a hostname changing its IP address, TTL, which is measured in seconds, is required to maintain the data current. A fresh DNS query must be made when the TTL expires.
There is a hierarchy to DNS, as shown in Figure 3. DNS is able to deal with the scale issue thanks to its hierarchical structure and huge number of servers. The server would become overloaded and unable to handle all of the requests if there was just one server responsible for handling them. As previously indicated, the answer is to partition the flow in a hierarchical way.

II. IoT and DNS

IoT devices frequently use the DNS protocol to find these remote services and exchange data with them. For example, IoT devices may analyze sensor information through one or maybe more external services housed on the Internet. IoT deployments consequently take place across two co-evolving and interacting ecosystems: (1) the DNS with its resolver operators, authoritative dns server operators, and custom domain providers, and (2) the IoT with its device manufacturers, IoT device operators (such as drone operators), and providers of the remote services with which these devices interact.

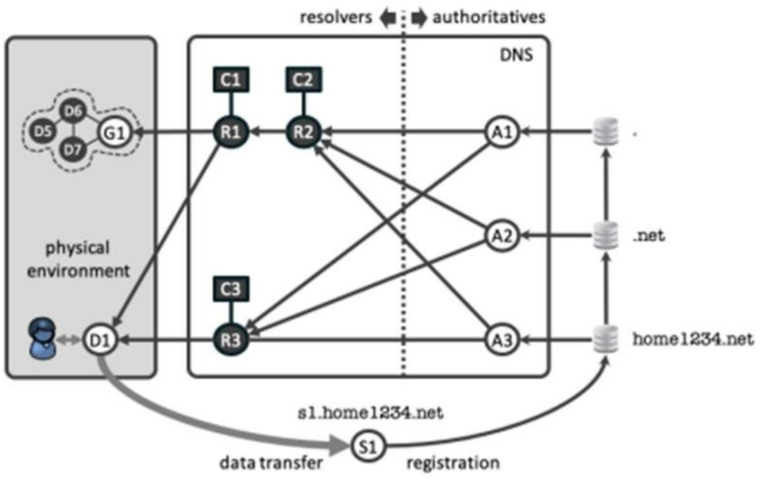

Figure 4: IoT with DNS resolution [2]

III. DNS Challenges in IoT

The DNS in the IoT has a number of difficulties. The three topics on which this thesis focuses each have obstacles, which are discussed in this chapter. The functionality challenges are discussed in Section 1, security challenges are discussed in Section 2, and availability challenges are discussed in Section 3. [2]

1. Functionality

Any device that is a part of the IoT must have a unique identification in order for it to develop and become a global infrastructure. Units in the IoT have limited capabilities, as shown in Figure 1, Structure of the IoT. The main issue that ought to be taken into account is the constraints brought on by using these limited technologies. These restrictionscanbe in the form of processing speed,memory capacity,or power supply.
The restricted devices are consequently smaller than conventional devices. Being portable is a benefit of having smaller devices. The gadgets would need to be configured or set up each time they were moved, which would be troublesome. Zero configuration can be used to resolve this problem.

2. Security

The security issue was not taken into consideration when the Domain name system had first been created, more than thirty years ago. Later, as an enhancement, security was implemented, offering source integrity and authenticity. The first challenge is designing and maintaining a free software library that implements DNS security for IoT devices (e.g., DANE, DoH/DoT, and DNSSEC validation) [3]; transparency and regulate of remote services with which their IoT devices communicate; and other functions like traffic obfuscation to conceal the patterns of Internet of things with incredibly technical functions (e.g., the traffic pattern of light switches); and automatic resolver failover.
Such a library would need to function with the most widely used IoT operating systems and CPU architectures (such OpenWRT and RIOT) as well as the more constrained resources of common IoT devices (like constrained battery life, CPU power, or capacity for performing cryptographic operations). Additionally, it would need to provide an API so that designers of IoT devices could simply incorporate and utilize the library. The SPIN genuine visualizer for DNS query trends of IoT devices and Danish, which makes HTTPS DANE accessible on OpenWRT, could serve as starting pointsfor the creation of such a library.

3. Availability

Figure 2 shows that the system must be capable of sending requests to several servers in order for DNS inquiriesto be successful. For DNS to function as intended, it is essential that the servers are functioning properly because they can only react if they are online. Constrained devices also have restricted power sources and frequently rely on a charge of some sort. These gadgets use sleep mode because they need to preserve power. A system in standby mode cannot replyto a direct query, which is a problem with availability.

IV. Firmware Update Procedures

[4] The process of updating the firmware is prevalent in IoT devices. However, a certain approach must be taken into account in order to conduct a safe firmware (FW) updating in hardware platforms. In this part, we outline the conditions that must be met for a safe FW download as well as the difficulties that may arise.

1. Requirements on Secure Firmware Update

Before describing the solutions now known for securing the firmware update process, we will first explain the prerequisites for safe firmware update operation and potential assaults on the firmware update. To install the new FW securely and effectively, the following prerequisites must be satisfied.

a. *Request for update:* An authorized organization that manages the FW update procedure sends an external FW update request to the IoT device.
b. *Licensed flash driver:* It is essential to confirm that the entity in charge of the FW update procedure is also the one issuing orders, like putting the flashing driver into the Memory or flashing data memory.
c. *Genuine firmware:* The Firmware needs to be examined to make sure it is compatible and free of malware.
d. *Authorized components:* The approved hardware and analysis programmes provide a secure FW updateprocedure.
e. *Rollback mechanism:* To enable a reliable update in the event of a failure, a suitable rollback method must exist. A botched update or the discovery of malicious or hacked FW might both constitute a failure.

2. Attempts to Subvert Firmware Update Categories

The following categories can be used to categorize the hazards to the Firmware update operation:
a. *Reverse engineering* is the process of obtaining sensitive information by extracting the firmware from a device and analyzing it without gaining physical access to it.
b. The attacker modifies the firmware by injecting extra code to enable unauthorized actions.
c. Getting access authorisation: Some gadgets require authorization before communicating with outside devices. An attacker will be able to carry out various assaults on the target devices if he can obtain that authorization.
d. Installing unauthorized firmware: An attempt will be made to install an unauthorized FW (whether its a harmful or stolen lawful FW) to the device by a potential attacker or a legitimate party with malicious intentions. The attacker may launch further assaults after installing the illegal images on the target device.
e. Unauthorized device: A prohibited device may impersonate a legitimate one in order to obtain a genuine copy of the FW. This might lead to privacy problems, financial losses for the maker of the gadget, and the possibility of additional malicious activities.

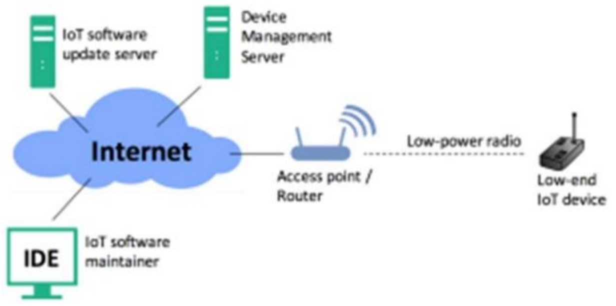

Figure 5: Prototyping IoT Firmware Update Concept [3]

In this essay, we take the scenario depicted in Figure 5 into consideration for additional research. In this instance, an IoT device is linked to an online device management server over a low-power wireless network [5]. An authorized IoT software operator should be able to do the following during the life of this IoT device:
a. Produce authorized, integrity-protected firmware upgrades;
b. Reboot the device after causing it to fetch a firmware image and check its validity and integrity;
c. If there is a change in ownership or a change in the terms of the contracts, give another maintenanceauthorization;
d. Reformat the device to allow for future upgrades to encryption protocols.

Effect of Corrupted Firmware

In this case, the simulated IoT devices randomly choose a firmware version, and submit 500 queries in succession to the name server in search of OX resource entries. We simulate the faulty firmware as indicated in the streamlined update logic in order to analyze how the update process behaves when the client is using corrupted firmware.

Each IoT device generates a homogeneous random number between [0, 1] at the start of every query. For instance, to simulate a 10% faulty firmware scenario, we determine whether the generated random number is less than or equal to 0.1. The IoT client will query with an erroneous hash value if the generated random number is less than 0.1, suggesting that the client's firmware is faulty, rather than choosing a valid hash value. Here, we presume that the client can still request OX RRs from the name server while having faulty client firmware. On the other hand, if the name server receives a query for which it does not have any OX RRs that match, it responds to the client with an NXDomain error message. The client requests with the Model ID to obtain an OX answer when it receives an NXDomain error response. The default OX RRs, which contain the URI of the firmware that can be installed directly on the given Model ID without any dependency issues, are returned to the device when it queries with the Model ID. Corruption scenarios could still occur before the query was sent with the Model ID.

The experiment is carried out using the streamlined updating logic. According to different corrupted firmware rates ranging from 10% to 100%, we estimated the percentage of successful firmware updates. Similar to other cases, this one's try value is 6, meaning IoT clients will only attempt up to six times. An upgrade is deemed unsuccessful if users continue to see NXDomain errors. Figure 6 displays the plot by changing the corrupted firmware rates on the x-axis and measuring the successful percentage on the y-axis. The plot makes it evident that even if 90% of our firmware is faulty, only 45% of updates are successful. Figure 7 illustrates how we also plotted the typical number of tries for various corrupted firmware rates.

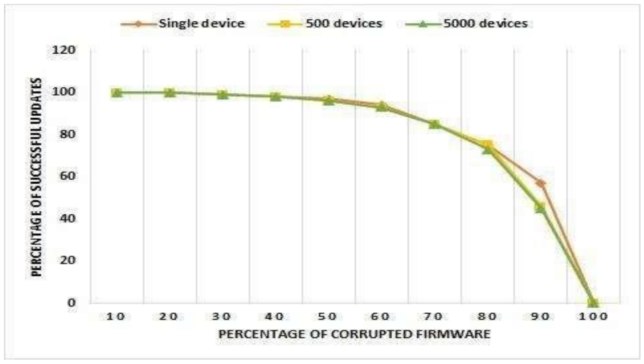

Figure 6: Percentage of Corrupted Firmware vs Percentage of Successful Updates

The graph in Figure 6 compares the percentage of successful updates to the percentage of faulty firmware for a single device, 500 devices, and 5000 devices. The successful upgrades tend to zero out when the defective firmware does the same, thus this is to be expected. As a result, they are exactly proportional. Even with 90% corruption, updates have a higher probability of being successful, which is a crucial aspect of software updates. As a result, only the absolute minimum amount of manual interaction is required, and the majority of network upgrades are automated.

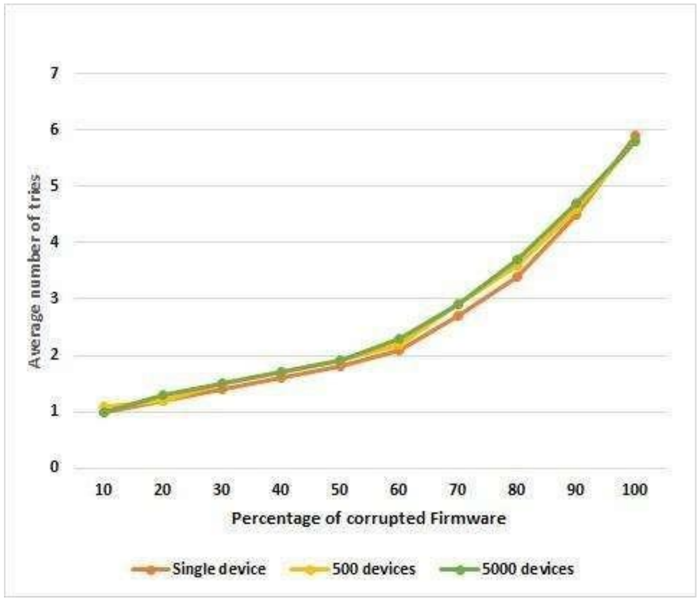

Figure 7: Percentage of Corrupted Firmware Average Number of Tries

The graph in Figure 7 compares the proportion of devices with corrupted firmware to the typical number of retries on one device, 500 devices, and 5000 devices. Retires are maximal when the defective firmware tends to 100, as expected, and are hence directly proportional. If the amount of corruption is lower, the number of retries drops significantly.

Effect of nsupdate

In this experiment, the name server receives nsupdate requests from the nsupdate client using dnsperf at a constant rate of 2500 updates per second. In this case, nsupdate queries are sent to the name server using dnsperf [6]. The dnsperf client searches for the same record that is being updated while the update is taking place. Figure 7 compares response rates with and without nsupdate for various updates per second. When nsupdate is active, the response rate per second is lower than when nsupdate is notactive.

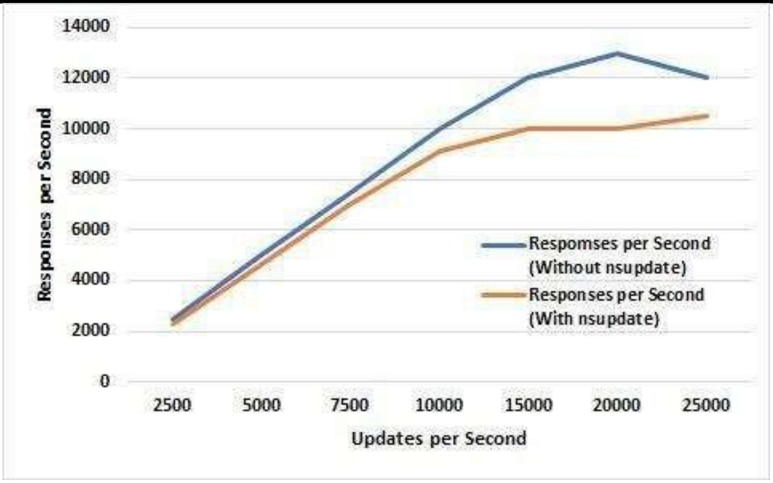

Figure 8: Updates per Second vs Responses per Second

Plotting updates per second versus responses per second results in the graph above. Without nsupdate, responses per second stay essentially constant up to 8000 updates per second. After that, there is a discernible difference in the responses per second received while nsupdate is active, and the responses per second without nsupdate are significantly higher than the responsesper second with nsupdate.

V. Conclusion

IoT devices are now widely employed in products like automobiles, industrial automation, and home appliances. In recent years, multiple big hacks and compromises have been caused by insecure IoT devices. The majority of these accidents are blamed on faulty and obsolete IoT device firmware. The proposed approach in this study tests the appropriateness and scalability of utilizing DNS infrastructure to reliably update the firmware of the IoT device by executing experiments under various unfavorable network conditions. In order to properly update IoT devices to the most recent version while guarding against Man-In-The-Middle (MITM) attacks, we have designed a straightforward update logic. This research proposes four alternative update logic versions that are suitable for varied applications. We tested the suggested system on a testbed that comprised a name server, simulated IoT devices, a ns-update client, and a firmware server on an IPv6 only network. By altering the number of simulated IoT devices from 1,500 to 5000, we were able to account for a range of situations, such as packet loss, corrupted firmware, managing simultaneous update and query scenarios, handling numerous fragmentation scenarios, etc. The results of the experiment showed that the name server can service requests from 5000 devices in the same way as it would serve requests from a single device while also handling the additional load created by 4000 devices searching for OX RRs. It has also been established that the manufacturer may update the OX RRs that are already present in the name server with a very rapid update rate as the clients request the OX RRs.

References

[1] Lee, Keuntae & Kim, Seokhwa & Jeong, Jaehoon & Lee, Sejun & Kim, Hyoungshick & Park, Jungsoo. (2018). A framework for DNS naming services for Internet-of-Things devices. Future Generation Computer Systems. 92. 10.1016/j.future.2018.01.023.

[2] C. Hesselman et al., "The DNS in IoT: Opportunities, Risks, and Challenges," in IEEE Internet Computing, vol. 24, no. 4, pp. 23-32, 1 July-Aug. 2020, doi: 10.1109/MIC.2020.3005388.

[2] Díaz-Sánchez, D.; Marín-Lopez, A.; Almenárez Mendoza, F.; Arias Cabarcos, P. DNS/DANE Collision-Based Distributed and Dynamic Authentication for Microservices in IoT †. Sensors 2019, 19, 3292. https://doi.org/10.3390/s19153292

[3]Bettayeb, Meriem & Nasir, Qassim & Abu Talib, Manar. (2019). Firmware Update Attacks and Security for IoT Devices: Survey. ArabWIC 2019: Proceedings of the ArabWIC 6th Annual International Conference Research Track. 1-6. 10.1145/3333165.3333169.

[4] K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig and E. Baccelli, "Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check," in IEEE Access, vol. 7, pp. 71907-71920, 2019, doi: 10.1109/ACCESS.2019.2919760.

[5] Atharva Rajendra Karpate "Development of firmware update mechanism for IoT devices using DNS and IPv6"Summer Research Fellowship Programme of India's Science Academies

[6] DNSPerf, https://github.com/DNS-OARC/dnsperf

[7] F. Ebbers, "A Large-Scale Analysis of IoT Firmware Version Distribution in the Wild," in *IEEE Transactions on Software Engineering*, doi: 10.1109/TSE.2022.3163969.

[8] Bradley, Conner & Barrera, David. (2022). Toward Identification and Characterization of IoT Software Update Practices.

[9] Tsaur, W.-J.; Chang, J.-C.; Chen, C.-L. A Highly Secure IoT Firmware Update Mechanism Using Blockchain. *Sensors* 2022, *22*, 530. https://doi.org/10.3390/s22020530

[10] Barrera, David & Molloy, Ian & Huang, Heqing. (2018). Standardizing IoT Network Security Policy Enforcement. 10.14722/diss.2018.23007.

[11] Kvarda, Lukas & Hnyk, Pavel & Vojtech, Lukas & Lokaj, Zdenek & Neruda, M. & Zitta, Tomas. (2016). Software Implementation of a Secure Firmware Update Solution in an IOT Context. Advances in Electrical and Electronic Engineering. 14. 10.15598/aeee.v14i4.1858.

[12] Karakostas, Bill. (2013). A DNS Architecture for the Internet of Things: A Case Study in Transport Logistics. Procedia Computer Science. 19. 594–601. 10.1016/j.procs.2013.06.079.