Exploiting the remote server access support of CoAP protocol

Annie Gilda Roselin[1,2], Priyadarsi Nanda[1], Surya Nepal[2],
Xiangjian He[1], Jarod Wright[3,2].
1. University of Technology Sydney (UTS), Australia.
2. CSIRO/Data61, Marsfield, NSW, Australia.
3. University of Wollongong, NSW, Australia.

***Abstract*—The Constrained Application Protocol (CoAP) is a specially designed web transfer protocol for use with constrained nodes and low-power networks. The widely available CoAP implementations have failed to validate the remote CoAP clients. Each CoAP client generates a random source port number when communicating with the CoAP server. However, we observe that in such implementations it is difficult to distinguish the regular packet and the malicious packet, opening a door for a potential off-path attack. The off-path attack is considered a weak attack on a constrained network and has received less attention from the research community. However, the consequences resulting from such an attack cannot be ignored in practice. In this paper, we exploit the combination of IP spoofing vulnerability and the remote server access support of CoAP to launch an off-path attack. The attacker injects a fake request message to change the credentials of the 6LoWPAN smart door keypad lock system. This creates a request spoofing vulnerability in CoAP, and the attacker exploits this vulnerability to gain full access to the system. Through our implementation, we demonstrated the feasibility of the attack scenario on the 6LoWPAN-CoAP network using smart door keypad lock. We proposed a machine learning based approach to mitigate such attacks. To the best of our knowledge, we believe that this is the first article to analyze the remote CoAP server access support and request spoofing vulnerability of CoAP to launch an off-path attack and demonstrate how a machine learning based approach can be deployed to prevent such attacks.**

***Index Terms*—IoT security, CoAP, 6LoWPAN, Machine Learning model, off-path attack.**

I. Introduction

A playbook consisting of rules and actions is created to protect the network communication against various attacks such as Man In The Middle (MITM) attack, Denial Of Service (DoS) attack and off-path attacks. Preventing and mitigating attacks in a constrained network is more challenging than in the well-established Internet world. Emerging applications such as smart home, smart city, healthcare monitoring systems, transportation, industrial automation, and agriculture [1] [2] use the communication of constrained devices with the Internet. Such communication becomes possible because of the vital roles of the protocols in each layer of the communication stack.

Like HTTP (Hyper Text Transport Protocol), CoAP (Constrained Application Protocol) is an application layer protocol specifically designed for constrained network devices [3] [4] [5] [6]. It facilitates communication between the Internet and constrained devices. CoAP follows the REST (Representational State Transfer) architecture and supports GET, PUT, POST methods on the resources.

CoAP reinforces a request-response model of communication between the endpoints. It involves four types of messages: CON (Confirmable), NON (non-confirmable), ACK (Acknowledgment) and RST (Reset). Whenever a CoAP client sends a request to the CoAP server, a connection is opened with the server. When the client receives a response, the connection with the server is closed. CoAP is built on top of the UDP (User Datagram Protocol) transport protocol. UDP is not as reliable as TCP (Transmission Control Protocol) since it does not offer a proper handshake between the client and server. To increase the reliable communication, CoAP supports a simple stop and wait mechanism for re-transmission with an exponential back-off mechanism for CON messages and duplicate detection for both CON and NON messages.

The off-path attack does not need to interfere with the IoT traffic irrespective of whether it is cryptographically secured. The off-path attack does not insert or modify the payload of a message like a MITM attack. Instead, it sends a fake packet between the communicating entities by spoofing the IP address. Fig. 1 shows an off-path attack model on the CoAP protocol. The off-path attacker gets into the victim machine by installing malicious software. The attacker extracts the IP address of the CoAP server through the browser extension [7]. It then performs an off-path attack by directly communicating with the server via another path bypassing the credential checks.

We further explain the off-path attack using a case study of the smart door keypad lock. Using the current CoAP implementation whenever the CoAP server receives a PUT request from a CoAP client to update the doorlock resources, it accesses the database which contains the authentication/authorization credentials. The CoAP server does not validate the requests coming from the remote CoAP clients. Hence these implementations open the door for the off-path attack. Even if the smart door keypad lock application is protected with the standard IoT authentication protocol such as DTLS, the injection of a fake pin code by the off-path attack is possible in such implementations.

Furthermore, we demonstrate the attack in the presence of a firewall and a two-factor authentication method for the remote CoAP client. The consequences of fake information injection

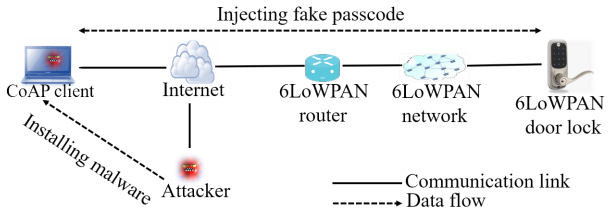


Fig. 1: Off-path attack model on CoAP protocol

on smart door keypad lock are high compared to smart light or smart metering. Therefore, we choose smart door keypad lock as our usecase to explore the vulnerability of the CoAP protocol. **Our contributions** are summarized as follows:

- We identify the "***Request Spoofing***" vulnerability of CoAP by exploiting the remote server access support of CoAP implementation along with the IP spoofing vulnerability of CoAP using an off-path attack. We observe that most available CoAP implementations are not performing the validation of remote CoAP clients. Even the widely used Python implementation of CoAP (CoAPthon [8]) has this vulnerability. It is thus a critical vulnerability in CoAP implementations which has not been reported thus far.
- We analyze and demonstrate the limitations of DTLS and firewall respectively to defend against the identified off-path attack. Also, we experiment the attack in a two-way authenticated environment and present a detailed analysis of our results to show that such authentication method alone cannot defend against such attack.
- We provide a detailed description of why an off-path attack is possible so that it can be repeated by researchers in their lab environment for different applications. More specifically, the difficulty of distinguishing the packets from an actual CoAP client and the attacker is discussed in detail. Also, the possible solutions to prevent the attack are discussed in detail. Without adding much overhead to IoT devices, we propose a simple machine learning approach to detect the abnormal behavior of the compromised CoAP client for preventing the identified off-path attack.

The rest of the paper is organized as follows. Section II provides a background information on how the CoAP protocol and WebSocket works. Section III describes the works related to the off-path attack. Our testbed architecture is described in Section IV. Section V outlines how the identified off-path attack works. The limitations of existing IoT security protocol such as DTLS, firewall, and two-factor authentication is explained in Section VI. Section VII outlines the potential defense mechanisms including a machine learning approach to mitigate the off-path attack. Finally, we conclude our analysis with a discussion in Section VIII.

II. Background

A. *6LoWPAN protocol stack with CoAP*

6LoWPAN communication protocol used in Low Power Area Networks (LoWPAN) has the ability of adopting IPv6

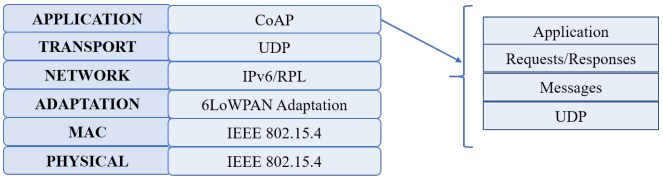
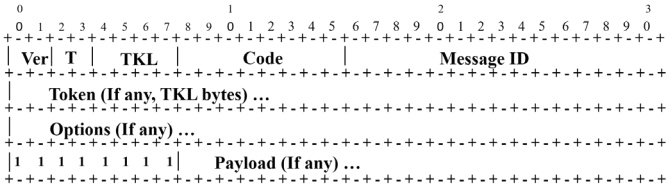


Fig. 2: 6LoWPAN protocol stack with abstract layering of CoAP



Ver (Version) :CoAP version number. Messages with unknown version number will be ignored
T (Type) :Indicates the type of message such as CON, NON, ACK, RST
TKL (Token Length) :Indicates the length of variable token length field. Usually 0-8 bytes. 9-15 bytes are reserved and processed for message format error.
Code :Request (1-10) or response method (40-255)
Message ID :16 bit indicator to detect duplicate messages. Used to match ACK/RST messages to CON/NON messages.

Fig. 3: CoAP packet

Internet protocol [3] [9] [10] [11]. Since IPv6 has large address space, it can subsume many constrained devices into the Internet. Fig. 2 shows the presence of CoAP and associated protocols in different layers of 6LoWPAN and explains the application layer in detail. 6LoWPAN supports CoAP protocol in its application layer and UDP in transport layer [12] [13]. 6LoWPAN adaptation layer does the job of header compression that grants the communication of IPv6 packets over the IEEE802.15.4 network.

6LoWPAN adopts the bottom-most two layers (Physical and MAC layers) from IEEE 802.15.4 standard and supports 127 bytes of data. Although link-layer security inside a LoWPAN (employing the 128-bit AES encryption in IEEE 802.15.4) provides some protection, communication beyond LoWPAN Routers is still vulnerable which increases the need for end-to-end security at the application layer [3].

B. *CoAP message format and its functionality*

A CoAP client, which needs a reliable transmission, sends a request CON message to the CoAP server and gets an ACK message back.

NON messages from the client do not get an acknowledgment back from the server, but still, have MessageID to avoid duplication of the same message. If the server is not able to process the CON message, then it replies with RST message instead of ACK.

Fig. 3 shows the CoAP packet format. CoAP packet consists of four bytes header information followed by optional token values and payload. The version field (two bits) indicates the CoAP version number. The CoAP server automatically ignores the CoAP messages that are having an unknown version number. The type field indicates the type of CoAP messages such as CON, NON, ACK, and RST. TKL field specifies the

Potential attacks on CoAP	Description of Attack	Possible Countermeasures
Attack on Complex protocol parsers	Crash a node remotely and execute arbitrary code remotely on parsers	Reducing parser complexity; moving much of the URI processing to CoAP clients; Care must be given to CoAP access control implementations
Man In The Middle attack on proxies	Breaks the confidentiality and integrity of the CoAP message by breaking any IPSec or DTLS protection on a direct CoAP message exchange through caching of proxies	Access control of resources must be considered. Do not perform caching on requests that have lesser transport-security properties
Amplification	The attacker attempts to overload a victim packet by turning a small packet into a large packet leading to a denial of service attack	Make the constrained network to generate a small amount of traffics. CoAP server can use Slicing/Blocking modes of CoAP. Limiting the support of multicast requests to specific resources.
IP address spoofing	Attacks the endpoint and even a whole network by spoofing the response and multicast request messages.	Response spoofing: by choosing the randomized token in the request. Use the security mode of communication. Request Spoofing: the focus of this paper.
Cross Protocol Attacks	Attackers send and receive a message to the CoAP endpoint	Strictly check the syntax of the received packets. Authorization of endpoints needed
Timing attacks	As constrained nodes are low in processing power, the attack can happen on cryptographic key generation and recovery of keying materials.	Care must be taken on the implementation of cryptographic primitives.

TABLE I: Potential attacks on CoAP

length of the variable token field which is followed by the CoAP header information. The usual length of TKL field is 0-8 bytes. Code field represents the unique code for request and response messages. The Message ID is a 16-bit indicator used to detect duplicate messages and to match ACK/RST messages to CON/NON messages. The token value will be 0-8 bytes used to correlate request and response messages. In the presence of payload, payload field is prefixed with one-byte payload marker (0xFF) which denotes the end of options and the start of payload. Since CoAP relies on UDP in its transport layer, it uses stop and wait scheme for re-transmission of CON messages and duplicate detection for both CON and NON messages to increase reliability between CoAP endpoints.

The endpoint (node) is determined by its IP address and UDP port number in the case of "NoSec" mode. CoAP-to-CoAP proxy maps a CoAP request to a CoAP request which means both the client and server uses the CoAP protocol to communicate. A CoAP-to-CoAP forward proxy is acting on behalf of the CoAP client to make requests to the CoAP server. CoAP-to-CoAP reverse proxy acting on behalf of the CoAP server to give the resources to the CoAP clients. Reverse proxy builds a namespace so that the client will get more control over where the request goes by embedding information such as host IP address and port number of the URI (Unified Resource Identifier) to direct the request to its intended resources. CoAP URI consists of URI-Host (Host IP address), URI-Port (transport layer port number), URI-Path (Absolute path of the resource) and URI-Query (Argument parameterizing the request). UDP port is the port where the CoAP server locates.

C. *Potential attacks on CoAP*

Potential attacks on CoAP [4] are stated in Table I. Our work focuses on IP spoofing, more specifically on "***Request Spoofing***" vulnerability, which is not even described/captured in RFC7252, of CoAP by exploiting the remote server access support of CoAP implementation. According to RFC7252, spoofing attacks on CoAP "***response messages***" can be performed as follows.

The malicious programmer prevents the CoAP client from re-transmitting the CON message by spoofing the ACK message and stops the actual response of the CoAP server. Another method is making the CoAP server disabled so that it is unable to receive any CON messages. This can be done by spoofing the RST messages.

The attacker spoofs the NON messages by making the CoAP server unable to receive any CON messages by spoofing RST messages. Spoofing the entire response is done by changing the entire payload of CoAP message with fake information. Spoofing a multicast request can lead to congestion in the network, DoS (Denial of Service) attack, and intentionally wake up the constrained device from sleeping (energy depletion attack).

However, we spoof a CoAP request CON messages to cause significant damage to the smart door keypad lock system, even before the actual user realizes the presence of an attack. This attack is different from the potential attacks identified above.

III. Related Work

Security issues caused by off-path attacks on TCP and DNS are very well researched and how they compromise challenge-response defense are analysed in [14] [15] [16] [17] [18]. Gilad et al. [19] showed that TCP injection is possible by the following method. Off-path attackers learn the connection sequence numbers of both the client and server in a TCP connection by exploiting a globally increasing IP-ID counter of Windows machine. Moreover, they suggested the use of security protocols such as SSL/TLS or IPsec to defend against such off-path attacks.

In [20], Gilad et al. experimented a practical off-path TCP-injection attack which allows web-cache poisoning. They suggested to modify the client port selection algorithm at NAT (Network Address Translation) level and deploy cryptographic methods such as SSL/TLS at the server side as a defensive mechanism against such off-path TCP injection attack. However, we analyze and present in Section VI that SSL/TLS (i.e., DTLS) based defensive mechanisms do not

work in our identified attack. Hence, we use a machine learning based approach to monitor the malicious activities of the compromised client and defense against our attack.

Source Port Randomization (SPR) is the mitigation technique for an off-path attack on TCP [21]. Fernandes et al. [22] used the vulnerability of the mobile app to launch the pin code injection attack on the smart door lock. However, our off-path attack uses the remote server access support of CoAP implementation to launch the off-path attack thereby injecting the fake password into the Smart Door Keypad lock system. Hence, the following defensive approaches do not work for our attack. As a mitigation technique of an off-path attack on TCP, DNS resolvers send a challenge - 16-bit TXID field with the request and expecting the same TXID in the response. Unpredictable port randomization of the client and dropping the connection with too many empty ACKs at the server side are the defensive mechanisms supported by a majority of the resolvers against the off-path attack. References [15] [23] used side channels for port prediction in order to execute the off-path attack.

In summary, existing works on the off-path attack for TCP focus on how such response spoofing is executed along with their countermeasures. However, our off-path attack on CoAP analyses how a "request spoofing" causes the damage to the constrained devices. In the case of CoAP; SPR technique is built in with the protocol, making it more vulnerable to "request spoofing". Since the off-path attacker injects the fake password value with the request CON message; the server cannot distinguish the spoofed packet from the original packet. Our article is the first and novel approach to explore the vulnerability of CoAP through the off-path attack. And, we identified the request spoofing vulnerability of CoAP by exploiting the remote server access support of CoAP implementations. Also, we have tested a machine learning model on our private network to predict the abnormal behavior of the infected CoAP client.

IV. Our Testbed Architecture

Our testbed architecture has two major parts. The first part focuses on the construction of testbed and the second part provides Two Factor Authentication (2FA) to the CoAP user. The Communication flow of our testbed starts with the user entering his/her credentials to authenticate themselves using 2FA mechanism as discussed in Section IV-B. Then the authenticated user can open the door, close the door, or even change the passcode of the door remotely. We enabled port-forwarding mechanism in Internet Protocol Router (IPR) to achieve CoAP communication with the 6LoWPAN network. 6LoWPAN border router acts as a bridge between Internet packets and 6LoWPAN packets by compressing Ethernet packets into IEEE 802.15.4 packets. Section II-A describes the 6LoWPAN protocol stack and the border router. We strongly assume that the off-path attacker enters the private network of the smart home to inject the fake passcode into the 6LoWPAN door lock, as discussed in Section V-A. Network traffic capture tool captures the network traffic of the CoAP remote clients and attacker to train the Machine Learning (ML) models in order to analyze and predict the behavior of the user agents.

A. ***Construction of Testbed***

The testbed consists of a constrained 6LoWPAN IoT network and the Internet. We explore the vulnerability of CoAP protocol and investigate how a smart door lock application may be exploited by the adversary injecting malicious information. Our overall implementation is based on Raspberry Pi. A similar implementation is possible using ARDUINO platform. Fig. 4 shows the architecture components and the communication between them.

We use the following hardware components to establish the 6LoWPAN IoT network. Raspberry Pi 3 Model B+ is used for a door lock and a border router. Raspberry Pi 802.15.4 radio from openlabs [34] is used to enable 6LoWPAN communication over 802.15.4 to the door lock. GrovePi is placed on top of the Raspberry Pi and connects the Grove LCD. GroveLCD is used to display the smart door lock status to the user. The keypad allows the user to enter the password and the LEDs reflect the user action on the door lock. 6LoWPAN IoT network establishment has the following two modules.

Module 1: *Building a CoAP enabled 6LoWPAN smart door keypad lock to perform the off-path attack.*

There are two ways to have the CoAP enabled smart door keypad lock system which follows the 6LoWPAN/CoAP communication stack protocol. One is to use a commercially available product such as IKILOCK [35]. Moreover, the other way is to build the prototype of the smart door keypad lock. At the time of establishing a test-bed, commercial 6LoWPAN smart door keypad lock was not available in our region as per our specifications. So we developed a prototype of 6LoWPAN smart door keypad lock using hardware components as mentioned earlier. Fig. 5 shows our smart door keypad lock prototype and explains the process flow of the prototype.

We opted Python programming language to develop the software of our smart door keypad lock prototype. The prototype software has three separate components. Firstly, a Python program for handling the hardware components. Secondly, a CoAP server which deals with how to access the resources of the prototype and communicating with a database. Thirdly a password management database which stores a password of the smart door keypad lock as shown in Fig. 6.

The Smart door keypad lock prototype displays two options such as "Enter Password," and "Change Password" on its LCD. The user will choose option A and enter the four-digit password if he wants to enter the home. The prototype accepts the four-digit password and validates it with the Password Management Database. The door will open once the validation of password returns success and the LCD displays the "Access Granted" message. Otherwise, the main options start to display on the LCD followed by the error message "Try again Later".

On the other hand, option B is for changing the password of the smart door keypad lock. To change the password, the user provides the old password for a credibility check with the database and then sets the new password which will be updated in the password management database by the CoAP server. The connection between the CoAP server and a password management database is established by giving the access control list credentials such as username, password

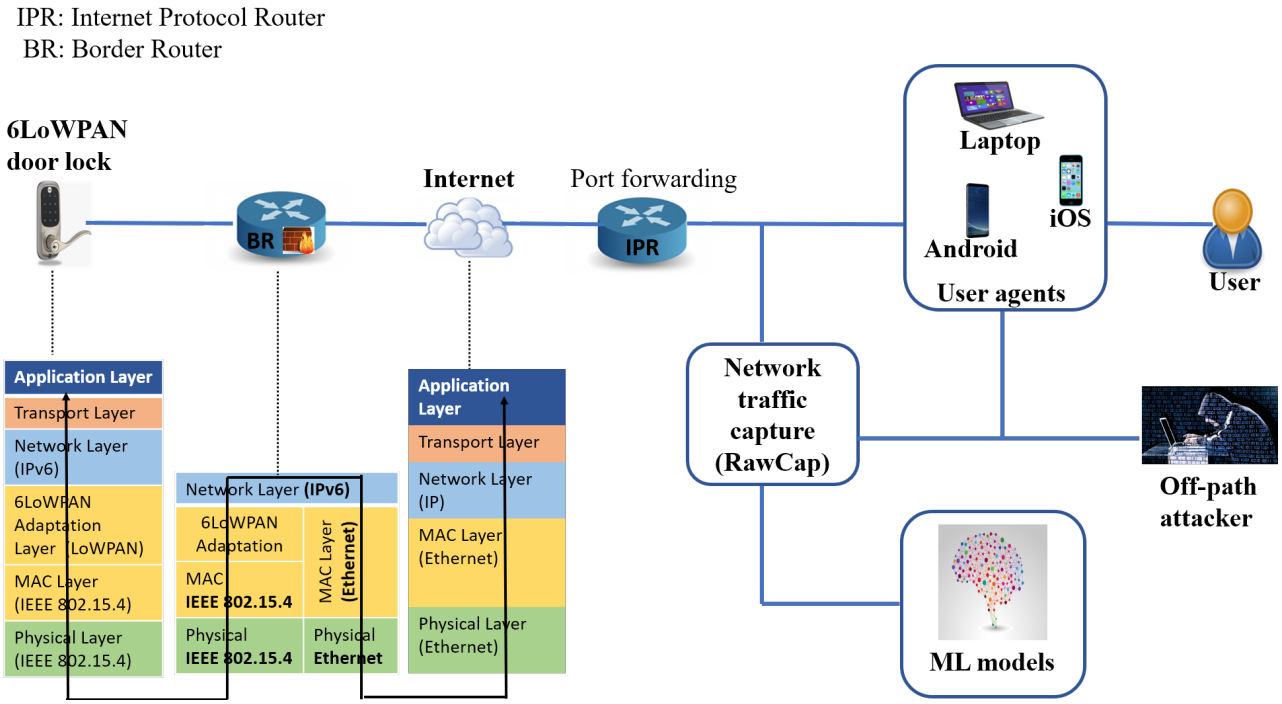
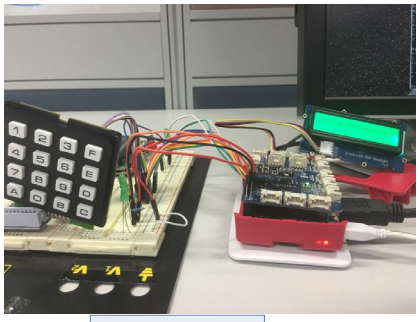


Fig. 4: Our testbed architecture



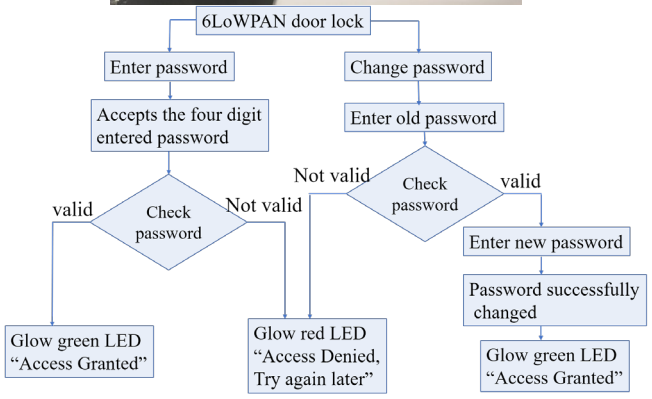


Fig. 5: Smart door keypad lock prototype and its process flow

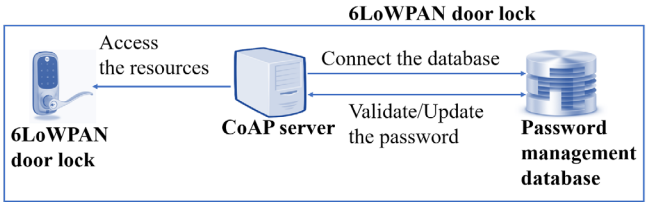


Fig. 6: Software components of smart door keypad lock

and the name of the database. Since MySQLdb implements the Python database API v2.0, we have used MySQLdb as an interface for the connection between a MySQL password management database and the CoAP server.

There are many open source CoAP protocol implementations available. Among them, we preferred CoAPthon implementation [8] of CoAP server and FireFox ESR with copper add-on as a CoAP client. The Firefox Copper has been disabled by Firefox after Firefox Quantum has been announced. We can use "Copper for Chrome (Cu4Cr) CoAP user-agent [36]" as a CoAP web interface to interact with the 6LoWPAN smart door lock. However, if still want to use Firefox browser, downgrade the Firefox Quantum to Firefox ESR 52.7.2. Since CoAPthon has the implementations of more CoAP features (CoAP server, client, forward proxy, reverse proxy, observe, multicast server discovery, CoRE Link Format parsing and block-wise transfer) as described in RFC7252 compared to other available CoAP implementations. We modified their code found in [37] according to our resources available in the 6LoWPAN smart door keypad lock prototype.

We execute our CoAP server on 6LoWPAN smart door keypad lock prototype and the CoAP client on the laptop. Also, we use Aneska and IoT-CoAP Android mobile apps which are available in the play store and app store as CoAP clients to access the CoAP resources of the 6LoWPAN prototype. The CoAP user gets the existence of the CoAP resources through CoAP resource discovery. Our prototype has three CoAP resources such as "OptionA", "OptionB"

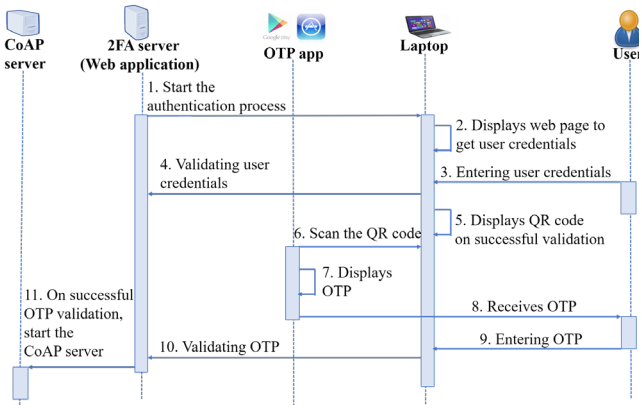


Fig. 7: Two factor authentication process for CoAP user

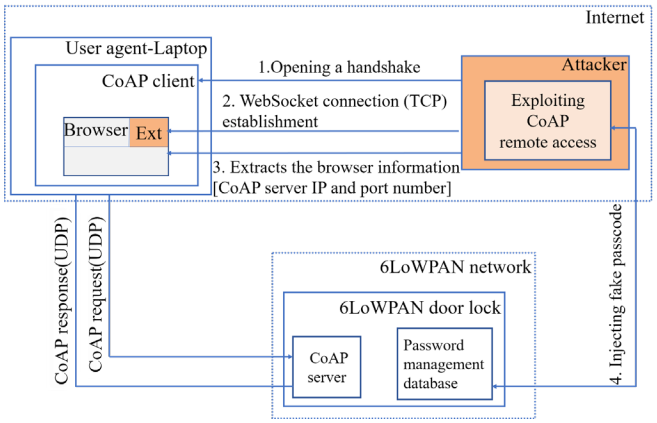


Fig. 8: Off-path attack scenario

and "Password". The user gets the status of the door, by accessing the OptionA and OptionB CoAP resources through GET command. If the door is not used by the user at that moment, the corresponding resources return OptionA/OptionB is in standby mode. Otherwise, GET command returns "The Door is Opened" or "Access Denied" messages respective to the CoAP resources accessed. The "Password" CoAP resource is used to change the password of the 6LoWPAN smart door lock prototype remotely through the PUT command of CoAP protocol. The users of the prototype can successfully change the password remotely using their Android or iOS mobile apps.

Module 2: *Building a 6LoWPAN border router on Raspberry pi.* For the 6LoWPAN border router, we use the native border router GitHub code found in [38]. We changed the RADVD configuration file by setting the prefix of the IPv6 address of the constrained 6LoWPAN network. When the RADVD server starts at the border router, the 6LoWPAN prototype within the constrained network is assigned with the IPv6 address using Neighbor Discovery Router Advertisement (RA) messages. This facilitates the CoAP client to access the CoAP resources of the 6LoWPAN prototype from the internet.

B. Two Factor Authentication (2FA) for the CoAP user

Fig. 7 explains the communication flows of remote CoAP user authentication process-2FA in detail. We choose the 2FA method code which is available online [39] in conjunction with CoAP. CoAP server starts its service only after a successful 2FA user authentication process. This method of IoT user authentication includes authentications of regular user password and a one-time token thereby increasing the level of security in a web application. The one-time password is based on the Time-based One-Time Password (TOTP) algorithm [40] and changes every 30 seconds. The user passwords are not stored as plaintext in the database. Instead, the hash value of the password is saved for verification. Upon successful password verification, the user receives a QR code to proceed further. This QR code is scanned by a Free OTP app which is available in android and app store. The user is authenticated when providing the TOTP within its validity time, thereby getting the access of CoAP resources. This TOTP is not a CoAP responses payload. After successful user authentication, CoAP server of our 6LoWAPN prototype will be enabled, and by hitting the "Discover" button, we get to know the available CoAP resources on the CoAP server. Communication flow-5 of Fig. 4 represents the CoAP resources accessed by CoAP clients.

We agree that the 2FA implementation in the context of CoAP for low power devices is computationally complex. Also, the CoAP endpoint user has to wait until the authentication process is finished and getting the response from the CoAP server. However, we have implemented 2FA with CoAP to provide the endpoint security to the CoAP client. We firmly believe that the existing standard security protocol DTLS in the context of CoAP for constrained devices provides data integrity between the communicating endpoints as per RFC 7252 and has limitations related to IP Spoofing and cross-protocol attacks as we discussed clearly in section IV-A.

V. Our Off-Path Attack

We assume that the attacker gets into the user's device (laptop) to spoof the IP address and communicate with the desired 6LoWPAN device. To launch the off-path attack, the attacker needs IP address and port number of the CoAP server. Even if the port number is not mentioned in the browser, the attacker uses the default port number 5683, of CoAP server to execute the attack.

A. Off-path attack scenario

Fig. 8 explains the attack scenario of off-path attack which involves the attacker, user agent-laptop and a 6LoWPAN door lock in detail.

Make a connection with CoAP Client: An attacker reaches the victim's machine (the client) by making it install a malware thereby the attacker loads a JSON file on the browser. This creates an extension in the client's browser. The attacker's code running inside the client's machine is making a TCP 3-way handshake (SYN, SYN-ACK, ACK) connection with a CoAP client whenever the malicious extension bar gets clicked by the user. Thereby a TCP socket connection is opened for malicious communication. We use WebSockets and asyncio libraries of Python to establish such a malicious connection. The execution

of off-path attack follows the establishment of a WebSocket connection between the malicious code and the CoAP client [41] [42]. CoAP proxy at the WebSocket endpoint forwards the request to the CoAP server with a CoAP UDP endpoint. Such a malicious connection stays alive until the malicious program terminates.

Launching an off-path attack: The malicious code running behind the browser extracts the browser information upon successful establishment of a malicious WebSocket connection. The attacker receives the CoAP server's IP (destination IP) address and a destination port number with the help of the victim browser's extension bar. Then the attacker attempts to inject the malicious passcode value to the CoAP server by exploiting remote server access support of CoAP protocol. The off-path attack happens following a sequence of message transmission which uses TCP and WebSocket protocol.

Moreover, the attack is possible despite the usage of the addresses IPv4 or IPv6 user communication with the CoAP server. The attacker updates the password database of the 6loWPAN prototype by sending the CoAP request with his/her own hard-coded password value without the knowledge of the actual user. By this way, the attacker creates the "***Request Spoofing***" vulnerability on CoAP protocol to inject the fake password in the desired 6LoWPAN device's database. Once the attacker succeeds in his/her off-path attack, he gets the full access of the 6LoWPAN smart door keypad lock.

Such an off-path attack can be performed in any commercial device which follows the 6LoWPAN/CoAP protocol stack if the attacker reaches the authenticated/non-authenticated client's machine. Our work will be enhanced in the future, with an IoT testbed that includes a large number of constrained IEEE 802.15.4 sensor nodes to check the scalability, packet losses, and packet delay of constrained devices.

B. 6LoWPAN-CoAP packet analysis with off-path attack

We analyze 6LoWPAN-CoAP packet based on the CoAP message format as discussed in Section II-B. We presented the actual user CoAP request packet in Fig. 9 and the off-path attacker request packet in Fig. 10 which are captured by Wireshark tool.

CoAP over WebSocket: If the CoAP communication happens over WebSocket, then the message packet will have zero in its length (Len=0) field thereby the version field of CoAP packet is suppressed with Len field of WebSocket [42]. Also, CoAP over WebSocket transmission does not distinguish CON and NON messages and does not provide ACK/RST messages. However, the off-path attack's CoAP request packet does have all the fields as in CoAP over UDP transmission packet, and it is being sent as a CON message. Moreover, it receives the ACK message from the CoAP server. Since the attacker's request packet is very similar to CoAP client request packet, it is hard to distinguish the poisoned packet from the actual client packet.

Off-path in TCP and SPR: The conventional method to prevent the off-path attack in TCP and UDP is to include a nonce value with the client's request so that validation of the request message is possible when receiving the response

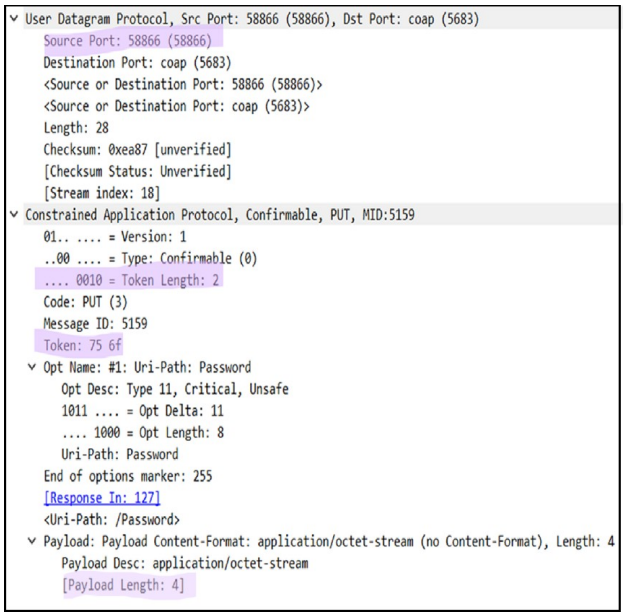


Fig. 9: Actual user's CoAP request packet captured by Wireshark

message. Unlike the TCP injection attack, the CoAP off-path attacker is spoofing the request message (injecting the fake passcode), not the response message from the server. So the countermeasure of including the nonce value with the request message is not enough in our case. Moreover, CoAP supports SPR technique in its clients; making the protocol more prone to off-path attack. From Fig. 9 and Fig. 10; we can not distinguish the CoAP packets based on its origin (either the user or the hacker) since both of the packets are having random source port numbers.

Server Port Number: According to RFC6335 [43] IANA (Internet Assigned Numbers Authority) has assigned 5683 as a default port number for CoAP server. If the UDP port is not given in the URI or the field is empty, then the default port 5683 will be assigned as a CoAP server port. Hence CoAP server port is explicit to the endpoints; leads the off-path attacker easy to guess the CoAP server port number.

Randomized token number: A randomized token number is generated when the CoAP messages are not protected by the transport layer security to mitigate the response spoofing [42]. CoAP client must generate a randomized token ID for every request it makes to the CoAP server to match the request and response. The token ID also referred as "request ID". This token length would be up to eight bytes, and at least 32 bits of token length must be used if the CoAP endpoint is connected to the Internet. CoAP server will echo the same token value with its response. In some situation, the token ID generation is not mandatory. For example, when the CoAP client sends a serial request to the CoAP server and CoAP client gets piggybacked responses from the CoAP server. Hence it is not possible to differentiate the malicious and original CoAP request packets based on randomized token ID value.

Payload length field: According to the RFC7252, in the spoofed CoAP packet format; after the one-byte payload

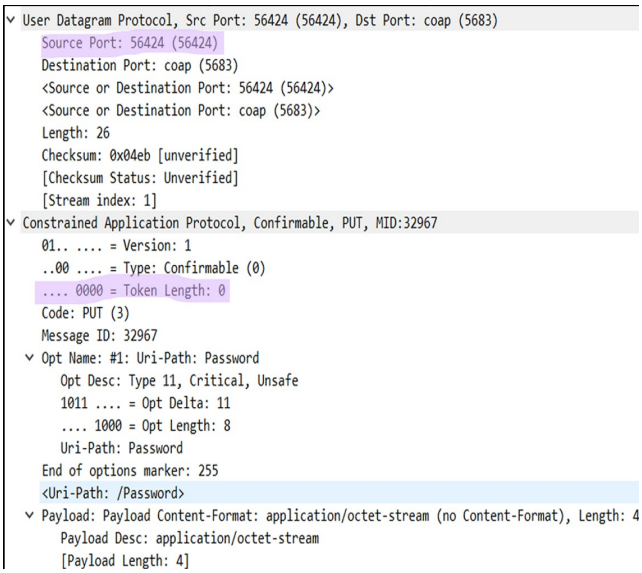


Fig. 10: The attacker's CoAP request packet captured by Wireshark

marker (0xFF), the payload length must be zero. Then automatically the packet will be thrown out for message format error. Also, the code field must be set with the reserved class (1, 6, or 7). However, our captured off-path attacker's packet does have the payload length the same as the original CoAP packet.

The CoAP server rejects the packets which are having random token length beyond 8 bytes (i.e., 9-15 bytes). However, CoAP allows same token length value for different client port and supports SPR. This makes an off-path attack very easy to implement.

VI. Limitations of Existing IoT Security Protocol and IoT User Authentication

We analyzed and observed that the existing IoT security protocols are not giving protection against our implemented off-path attack. They are thereby creating the loophole/ vulnerability in CoAP protocol. This section describes the limitations of DTLS, limitations of user authentication, and the limitations of the firewall briefly.

A. *Limitations of DTLS*

DTLS (Datagram Transport Layer Security) security protocol is the most common method used in conjunction with CoAP to reduce the communication vulnerability of IoT [24] [25]. Existing works [26] [27] [28] [29] use standard security protocol DTLS to secure the payload of CoAP communication. However, DTLS does not provide security against the cross-protocol attack if the same DTLS security connection is used to carry the data of multiple protocols [4]. All the modes of DTLS are not applicable for all constrained devices. Running of DTLS is impossible on class 0 devices and nearly impossible to be hosted on class 1 devices (10k RAM/100k ROM).

Implementation complexities of DTLS in constrained devices are high. Initial handshake overhead is high. DTLS adds 13 bytes of per-datagram overhead, excluding initialization vectors/nonce, integrity check values and padding values of cipher suite, which increase the overhead of LoWPAN devices. DTLS does not applicable to group keying communication (multicast communication). Since UDP protocol will not verify the source address of the request packet, CoAP is vulnerable to cross-protocol attacks with the fake source address. UDP based protocols are relatively easy targets for the cross-protocol attacks. Also, the probability of network delay is higher for asymmetric based handshake security protocols [30] [10].

Cross-protocol attacks are possible in UDP based communication if the security properties of the CoAP server and client rely only on the process of checking the source IP address. Also if we use proxies such as HTTP to CoAP or CoAP to HTTP, the transport layer security called DTLS has to be terminated at the proxy [31]. So we need a security protocol with features that offer defensive mechanisms against the combination of vulnerabilities such as IP spoofing and cross-protocol attacks of CoAP.

B. *Limitations of user authentication*

The possible best practice to protect the remote server access capability in the Internet world is protecting the network by VPN (Virtual Private Network), tunneling the network traffic [32] and know the user (the process of authenticating the user). We strongly assume that the attacker somehow reached the authenticated client's machine and started the malicious server program to extract the browser information. The widely used mitigation technique on the Internet against remote server access exploitation is by knowing the user using two-factor authentication mechanisms [33]. None of the research papers addressed CoAP user authentication through 2FA (Two Factor Authentication) method. We implemented it in our testbed as in Section IV-B.

Despite a secured 2FA CoAP user authentication, the off-path attack is possible, and we demonstrated it since we access the CoAP server resources by the IP address and port number of the 6LoWPAN device. Moreover, the memory occupancy by the 2FA authentication process is high on 6LoWPAN constrained devices, and obviously, it is adding additional initial computational time (user validation time) on the normal communication of CoAP.

C. *Limitations of firewall*

As discussed in Section V, the off-path attacker spoofs the IP address of an authenticated client. So filtering the packets based on their source IP address using the firewall is not going to prevent the off-path attack. Specifically, from the Section V-B we observed that the user CoAP packets and the poisoned packets are following the same protocol stack for the communication. Thereby we cannot merely deny the incoming UDP port transmission on the firewall. Though we installed and configured the UFW (Uncomplicated Firewall) firewall to reject the standard UDP port communication on our testbed, the off-path attack by-passes the firewall and gains access

to the 6LoWPAN doorlock device. We believe that the SPR feature of the CoAP protocol makes the 6LoWPAN network unable to use the firewall to defend against the off-path attack.

VII. Mitigation technique

Since the IoT traffic captured from our testbed has no significant differences between the authenticated client's CoAP packet and the attacker's packet as discussed in Section V-B, leads us not to stop the anomaly traffic with the rule-based approach. Also, the existing IoT security algorithms are failed to determine this kind of attack, as discussed in Section VI. For these reasons, we have implemented the Machine Learning (ML) model in CoAP client's machine (Laptop with Intel Core i7) to identify and predict the abnormal behavior of the infected CoAP client's network traffic. We can also allocate a dedicated machine inside the private network (for example, smart home) to monitor and predict the fake network traffic.

The technique of analyzing network traffic data using machine learning algorithms getting popular among the researchers [44], [45]. Unsupervised ML algorithms do not require pre-labeling of the dataset and create clusters on the network traffic data set based on their similar characteristics. Whereas our dataset needs a binary classification in the form of, whether the network traffic is malicious or normal traffic, we choose supervised learning ML methods to create an ML model for outlier prediction.

Random Forest supervised machine learning algorithm is used in [46] [47] to identify the IoT device traffic from non-IoT device traffic based on the characteristics of network traffic, it generates. Detecting the IoT nodes in wireless sensor network also presented in [48]. However, we used a machine learning algorithm to classify the traffic of the infected CoAP client which is trying to access the CoAP IoT server maliciously. Multiple experiments carried out on our IoT testbed network dataset with different ML algorithms to find out the best ML model. We have used the WEKA machine learning tool [49] to create the ML models.

A. Dataset

The training dataset (internal traffic of the infected client) which is captured by Rawcap tool, has "2676" instances. The dataset is converted into Attribute-Relation File Format (.arff) for processing using the WEKA tool. There is no feature selection algorithm applied to our IoT network traffic dataset since we are interested in all the features of the dataset except the number of attributes. We add one attribute called "Threat" to specify the label of each traffic instance. After labeling the training dataset, "Threat" attribute weight is distributed among the classes "No" and "Yes" with the weights "2449.0" and "227.0" respectively. It is obvious that the real-time IoT traffic dataset has the unbalanced weight distribution among the classes. The attribute "Protocol" weight is shared among "TCP", "HTTP" and "Websocket" with the corresponding weight of "2611.0", "26.0" and "39.0" respectively.

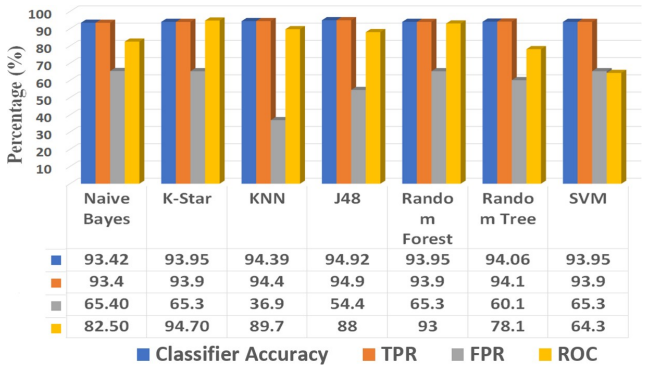


Fig. 11: Training phase: performance comparison of different ML classifiers

B. Training phase: Building ML models

We applied the ML algorithms such as Naive Bayes (NB), K*, K-NN (k-nearest neighbors), J48, Random Forest, Random Tree and Support Vector Machine (SVM) algorithms to build ML models and presented a graph in Fig. 11 with their performance comparison statistics. We use 10-fold cross-validation method to evaluate the performance of our ML models. This approach divides the data set of samples into ten groups/fold. For each group, keep the group as a test dataset and treat the remaining as the training dataset to evaluate the ML model. Hence keep the evaluation score for the corresponding ML model [50].

Classifier accuracy, True Positive Rate (TPR: number of examples predicted positive that are positive), False Positive Rate (FPR: number of examples predicted positive that are actually negative) and ROC (Receiver Operating Characteristic) is the parameters used to compare the performance of different ML classifiers. In our dataset, the payload and the structure of the malicious packet are very similar to the original CoAP packet as described in Section V-B. Generally, the FPR value is high, when the dataset is going through an intrusion detection with anomaly-based technique. We follow the hybrid of anomaly detection and signature-based intrusion detection so that we could detect unknown attacks in IoT.

Ranking an instance based on the area under the ROC curve is a widely accepted parameter to evaluate the ML algorithms [51]. We present the values of Classifier accuracy, FPR, TPR and ROC parameters for our dataset by applying ML algorithms such as NB, K*, K-NN, J48, Random Forest, Random Tree and SVM in Fig. 11. From the observation of Fig. 11, the ML algorithms such as K-Star and random forest have higher accuracy and ROC value as well.

Naive Bayes (NB): Naive Bayes ML classifier uses a probabilistic approach based on Bayes theorem with independence assumptions among the features of a dataset. Malicious traffic is classified based on the prior knowledge of the condition that might be related to the occurrence of suspicious network traffic. NB classifies the dataset according to every attribute into two intended classes. For our dataset, the obtained NB classifier model accuracy is "93.42%" and "82.50%" ROC area under the curve.

K-Star: K* is an instance-based classifier, that is the class of a test instance is based on the class of those training instances similar to it, as determined by some similarity function [52]. For our dataset, the obtained K* classifier model accuracy is "93.95%" and "94.70%" ROC area under the curve.

K-NN (k-nearest neighbors): Unlike Naive Bayes classifier, K-NN is not a probability-based classifier. However, K-NN classification is based on non-parametric statistics such as descriptive statistics and statistical inference of the instances. Our K value is 1 with Linear NearestNeighbour search which uses a Euclidean distance function to find out the nearest traffic. For our dataset, the obtained K-NN classifier model accuracy is "94.39%" and "89.7%" ROC area under the curve.

J48: J48 is a tree-based statistical classifier which is the implementation of the C4.5 algorithm in the Weka data mining tool. It produces a decision tree developed by Ross Quinlan [53]. In our training data set, the attribute "Protocol" acts as a root node. All the HTTP and WebSocket packets are classified as malicious traffic whereas the TCP packets are further classified based on time and length feature. For our dataset, the obtained J48 classifier model accuracy is "94.92%" and "88%" ROC area under the curve.

Random Forest: Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest [54]. Bagging with 100 iterations and base learner functions are performed on the training dataset. For our dataset, the obtained random forest classifier model accuracy is "93.95%" and "93%" ROC area under the curve.

Random Tree: To construct a tree it considers K randomly chosen attributes at each node. It performs no pruning. Also has an option to allow estimation of class probabilities (or target mean in the regression case) based on a hold-out set (back-fitting). KValue sets the number of randomly picked attributes. We choose the value of K is 0. If 0, int (log 2(#predictors) + 1) is used. For our dataset, the obtained random tree classifier model accuracy is "94.06%" and "78.1%" ROC area under the curve.

Support Vector Machine (SVM): Stable machine learning algorithm called SVM constructs hyper-plane to classify the instances of the training dataset. The larger functional margin between the hyper-plane and the training data points yields higher the accuracy of the classifier. Generalization of the classifier varies on the kernel function. We use Linear Kernel function $K(x, y) =< x, y >$ for the outlier detection. For our dataset, the obtained SVM classifier model accuracy is "93.95%" and "64.3%" of ROC area under the curve.

C. Testing phase: Threat Prediction

The test data set (internal traffic of the infected client captured by Rawcap tool) has "1078" instances. Among the 1078 instances, "210" traffic instances are malicious traffic. When applying this test data set to the previously constructed ML model, we got the malicious traffic prediction accuracy as shown in Fig. 12. Though K* and random tree classifier models

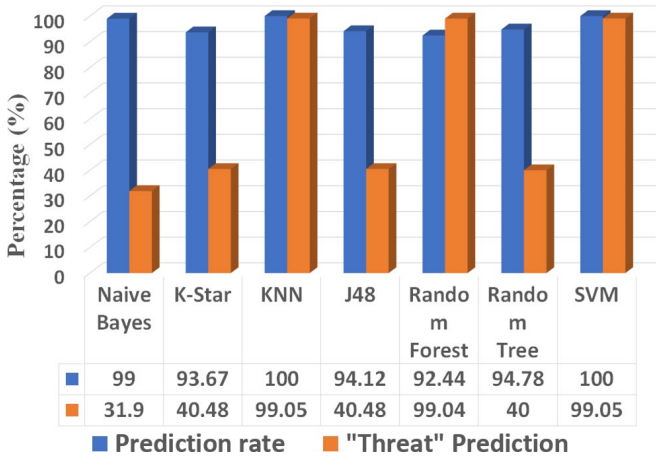


Fig. 12: Testing phase: Prediction accuracy comparison of different ML models

give higher classification accuracy and ROC area under the curve results, such algorithms failed to produce high "Threat" prediction accuracy. Comparing the prediction results, SVM and KNN models are giving "99.05%" of threat prediction accuracy. Hence we use SVM and KNN models for further real-time outlier detection in our IoT testbed. In the future, we will analyze the ML models in detail and train them with more datasets to predict the abnormal behavior of the IoT CoAP client.

VIII. Discussion and Conclusion

6LoWPAN and CoAP protocols are IETF standardized protocols. Sine 6LoWPAN supports IPv6; there is enough room for connecting more IoT devices in the WSN network, thereby supports scalability. Moreover, CoAP follows request-response architecture to maintain secure communication. This architecture will reduce the risk of having potential threats such as a drone controlling the entire light of the building found in [22] [55] which follows the publish-subscribe method of communication.

Our work in this paper is a concrete contribution to the IoT Cyber Security community to strengthen the security of the application layer protocol. Also, this work endorses the IoT device manufacturers to have machine learning model as an IoT network monitor to protect the IoT network against the off-path attack and the IoT devices' communication as well. With our observations and findings, it is recommended that the IoT industry can have more IoT products with 6LoWPAN-CoAP secure communication stack and ML model as a network monitoring tool in the IoT network.

Exploring the vulnerability of CoAP protocol and a mitigation technique is done through a case study of a smart door keypad lock system. While analyzing the CoAP packets, we observed that the spoofed CoAP request message packets have the randomized token number. This is the only difference between the spoofed CoAP request message and the normal CoAP request message of the CoAP server. However, in some situation, the token ID generation is not mandatory as discussed in Section V-B. Even with the presence of

authentication security protocols, we realize the attack after it has done the damage already.

The reason behind fake pincode injection in the 6LoWPAN/CoAP stack is, the implementations of CoAP protocol are not validating the remote CoAP clients. Hence creates "Request Spoofing" vulnerability, and if it is not treated properly, the damage for the 6LoWPAN device on the constrained network would be high. We experimented the off-path attack by constructing the prototype and launched the attack in an authenticated environment. Also, we presented the results of the countermeasure technique to mitigate the off-path attack using supervised machine learning model. In the future, we will automate the machine learning model to find out the abnormal behavior of live IoT traffic which help to detect and prevent the off-path attack. We believe that paying attention to "***Request Spoofing***" vulnerability and implementation method will help to improve the security against off-path attacks on CoAP.

ACKNOWLEDGMENT

This research is supported and funded by Data61, CSIRO, Marsfield, Australia.

REFERENCES

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, 2011.

[3] Z. Shelby and C. Bormann, *6LoWPAN: The wireless embedded Internet*. John Wiley & Sons, 2011, vol. 43.

[4] Z. Shelby, K. Hartke, and C. Bormann, "IETF RFC 7252," *The Constrained Application Protocol (CoAP)*, 2014.

[5] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, no. 2, pp. 62–67, 2012.

[6] K. Hartke, "Observing resources in the constrained application protocol (CoAP)," Tech. Rep., 2015.

[7] A. Barth, "The web origin concept," 2011.

[8] G. Tanganelli, C. Vallati, and E. Mingozzi, "CoAPthon: Easy development of CoAP-based IoT applications with Python," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 63–68.

[9] P. Pongle and G. Chavan, "A survey: Attacks on RPL and 6LoWPAN in IoT," in *Pervasive Computing (ICPC), 2015 International Conference on*. IEEE, 2015, pp. 1–6.

[10] A. G. Roselin, P. Nanda, and S. Nepal, "Lightweight Authentication Protocol (LAUP) for 6LoWPAN Wireless Sensor Networks," in *Trustcom/BigDataSE/ICESS, 2017 IEEE*. IEEE, 2017, pp. 371–378.

[11] A. G. R. Arockia Baskaran, P. Nanda, S. Nepal, and S. He, "Testbed evaluation of Lightweight Authentication Protocol (LAUP) for 6LoWPAN wireless sensor networks," *Concurrency and Computation: Practice and Experience*, p. e4868.

[12] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[13] M. Bouaziz and A. Rachedi, "A survey on mobility management protocols in Wireless Sensor Networks based on 6LoWPAN technology," *Computer Communications*, vol. 74, pp. 3–15, 2016.

[14] Y. Gilad, A. Herzberg, and H. Shulman, "Off-path hacking: The illusion of challenge-response authentication," *IEEE Security & Privacy*, vol. 12, no. 5, pp. 68–77, 2014.

[15] A. Herzberg and H. Shulman, "Security of patched DNS," in *European Symposium on Research in Computer Security*. Springer, 2012, pp. 271–288.

[16] H. Herzberg, Amir and Shulman, "Socket overloading for fun and cache-poisoning," in *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, 2013, pp. 189–198.

[17] A. Klein, "OpenBSD DNS cache poisoning and multiple O/S predictable IP ID vulnerability," 2007.

[18] Z. Qian and Z. M. Mao, "Off-path TCP sequence number inference attack-how firewall middleboxes reduce security," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 347–361.

[19] Y. Gilad and A. Herzberg, "Off-Path Attacking the Web." in *WOOT*, 2012, pp. 41–52.

[20] A. Gilad, Yossi and Herzberg, "Off-path TCP injection attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 13, 2014.

[21] A. Hubert and R. van Mook, "RFC 5452: Measures for making DNS more resilient against forged answers," 2009.

[22] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 636–654.

[23] A. Hubert and R. van Mook, "RFC 5452: Measures for Making DNS More Resilient against Forged Answers, 2009," *URL https://www. ietf. org/rfc/rfc5452. txt*.

[24] T. Fossati and H. Tschofenig, "Transport layer security (TLS)/datagram transport layer security (DTLS) profiles for the internet of things," *Transport*, 2016.

[25] E. Rescorla and N. Modadugu, "RFC 6347: Datagram transport layer security version 1.2," *Internet Engineering Task Force*, vol. 13, p. 101, 2012.

[26] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN compressed DTLS for CoAP," in *2012 8th IEEE International Conference on Distributed Computing in Sensor Systems*. IEEE, 2012, pp. 287–289.

[27] A. Capossele, V. Cervo, G. De Cicco, and C. Petrioli, "Security as a CoAP resource: an optimized DTLS implementation for the IoT," in *Communications (ICC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 549–554.

[28] P. Urien, "Innovative DTLS/TLS security modules embedded in SIM cards for IoT trusted and secure services," in *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*. IEEE, 2016, pp. 276–277.

[29] S. Raza, T. Helgason, P. Papadimitratos, and T. Voigt, "SecureSense: End-to-end secure communication architecture for the cloud-connected Internet of Things," *Future Generation Computer Systems*, vol. 77, pp. 40–51, 2017.

[30] H. Kwon, J. Park, and N. Kang, "Challenges in deploying CoAP over DTLS in resource constrained environments," in *International Workshop on Information Security Applications*. Springer, 2015, pp. 269–280.

[31] "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth) draft-ietf-ace-oauth-authz-12," https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-12, accessed: 2018-06-22.

[32] Y. Gilad and A. Herzberg, "LOT: a defense against IP spoofing and flooding attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 2, p. 6, 2012.

[33] Q. Xie, D. S. Wong, G. Wang, X. Tan, K. Chen, and L. Fang, "Provably Secure Dynamic ID-Based Anonymous Two-Factor Authenticated Key Exchange Protocol With Extended Security Model." *IEEE Trans. Information Forensics and Security*, vol. 12, no. 6, pp. 1382–1392, 2017.

[34] "OpenLabs Raspberry-Pi-802.15.4-radio," http://openlabs.co/store/Raspberry-Pi-802.15.4-radio, accessed: 2017-12-07.

[35] "IKILOCK smart lock design for Smart Home," https://www.ikilock.com/, [Online;].

[36] "Copper4Cr Copper for Chrome (Cu4Cr) CoAP user-agent," https://github.com/mkovatsc/Copper4Cr, [Online;].

[37] "CoAPthon CoAP implementation," https://github.com/Tanganelli/CoAPthon.

[38] "Native 6LoWPAN Router Using Raspbian and RADVD," https://github.com/RIOT-Makers/wpan-raspbian/wiki/Setup-native-6LoWPAN-router-using-Raspbian-and-RADVD, accessed: 2017-11-07.

[39] M. Grinberg, "Two Factor Authentication with Flask," https://blog.miguelgrinberg.com/post/two-factor-authentication-with-flask, [Online; accessed 14-Sep-2018].

[40] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "Totp: Time-based one-time password algorithm," Tech. Rep., 2011.

[41] T. Savolainen, K. Hartke, and B. Silverajan, "CoAP over WebSockets," 2015.

[42] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets," Tech. Rep., 2018.

[43] Cotton, M and Eggert, L and Touch, J and Westerlund, M, "S. Cheshire," Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry," BCP 165, RFC 6335, August, Tech. Rep., 2011.

[44] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 4, pp. 1257–1270, 2015.

[45] O. M. Alhawi, J. Baldwin, and A. Dehghantanha, "Leveraging machine learning techniques for windows ransomware network traffic detection," *Cyber Threat Intelligence*, pp. 93–106, 2018.

[46] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis," in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 506–509.

[47] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, "Detection of Unauthorized IoT Devices Using Machine Learning Techniques," *arXiv preprint arXiv:1709.04647*, 2017.

[48] S. Althunibat, A. Antonopoulos, E. Kartsakli, F. Granelli, and C. Verikoukis, "Countering intelligent-dependent malicious nodes in target detection wireless sensor networks," *IEEE Sensors Journal*, vol. 16, no. 23, pp. 8627–8639, 2016.

[49] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[50] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[51] H. A. Güvenir and M. Kurtcephe, "Ranking instances by maximizing the area under ROC curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2356–2366, 2013.

[52] J. G. Cleary and L. E. Trigg, "K*: An instance-based learner using an entropic distance measure," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 108–114.

[53] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[54] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[55] E. Ronen, A. Shamir, A.-O. Weingarten, and C. OFlynn, "IoT goes nuclear: Creating a ZigBee chain reaction," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 195–212.

Exploiting the remote server access support of CoAP protocol

Annie Gilda Roselin[1,2], Priyadarsi Nanda[1], Surya Nepal[2],
Xiangjian He[1], Jarod Wright[3,2].
1. University of Technology Sydney (UTS), Australia.
2. CSIRO/Data61, Marsfield, NSW, Australia.
3. University of Wollongong, NSW, Australia.

***Abstract*—The Constrained Application Protocol (CoAP) is a specially designed web transfer protocol for use with constrained nodes and low-power networks. The widely available CoAP implementations have failed to validate the remote CoAP clients. Each CoAP client generates a random source port number when communicating with the CoAP server. However, we observe that in such implementations it is difficult to distinguish the regular packet and the malicious packet, opening a door for a potential off-path attack. The off-path attack is considered a weak attack on a constrained network and has received less attention from the research community. However, the consequences resulting from such an attack cannot be ignored in practice. In this paper, we exploit the combination of IP spoofing vulnerability and the remote server access support of CoAP to launch an off-path attack. The attacker injects a fake request message to change the credentials of the 6LoWPAN smart door keypad lock system. This creates a request spoofing vulnerability in CoAP, and the attacker exploits this vulnerability to gain full access to the system. Through our implementation, we demonstrated the feasibility of the attack scenario on the 6LoWPAN-CoAP network using smart door keypad lock. We proposed a machine learning based approach to mitigate such attacks. To the best of our knowledge, we believe that this is the first article to analyze the remote CoAP server access support and request spoofing vulnerability of CoAP to launch an off-path attack and demonstrate how a machine learning based approach can be deployed to prevent such attacks.**

***Index Terms*—IoT security, CoAP, 6LoWPAN, Machine Learning model, off-path attack.**

I. Introduction

A playbook consisting of rules and actions is created to protect the network communication against various attacks such as Man In The Middle (MITM) attack, Denial Of Service (DoS) attack and off-path attacks. Preventing and mitigating attacks in a constrained network is more challenging than in the well-established Internet world. Emerging applications such as smart home, smart city, healthcare monitoring systems, transportation, industrial automation, and agriculture [1] [2] use the communication of constrained devices with the Internet. Such communication becomes possible because of the vital roles of the protocols in each layer of the communication stack.

Like HTTP (Hyper Text Transport Protocol), CoAP (Constrained Application Protocol) is an application layer protocol specifically designed for constrained network devices [3] [4] [5] [6]. It facilitates communication between the Internet and constrained devices. CoAP follows the REST (Representational State Transfer) architecture and supports GET, PUT, POST methods on the resources.

CoAP reinforces a request-response model of communication between the endpoints. It involves four types of messages: CON (Confirmable), NON (non-confirmable), ACK (Acknowledgment) and RST (Reset). Whenever a CoAP client sends a request to the CoAP server, a connection is opened with the server. When the client receives a response, the connection with the server is closed. CoAP is built on top of the UDP (User Datagram Protocol) transport protocol. UDP is not as reliable as TCP (Transmission Control Protocol) since it does not offer a proper handshake between the client and server. To increase the reliable communication, CoAP supports a simple stop and wait mechanism for re-transmission with an exponential back-off mechanism for CON messages and duplicate detection for both CON and NON messages.

The off-path attack does not need to interfere with the IoT traffic irrespective of whether it is cryptographically secured. The off-path attack does not insert or modify the payload of a message like a MITM attack. Instead, it sends a fake packet between the communicating entities by spoofing the IP address. Fig. 1 shows an off-path attack model on the CoAP protocol. The off-path attacker gets into the victim machine by installing malicious software. The attacker extracts the IP address of the CoAP server through the browser extension [7]. It then performs an off-path attack by directly communicating with the server via another path bypassing the credential checks.

We further explain the off-path attack using a case study of the smart door keypad lock. Using the current CoAP implementation whenever the CoAP server receives a PUT request from a CoAP client to update the doorlock resources, it accesses the database which contains the authentication/authorization credentials. The CoAP server does not validate the requests coming from the remote CoAP clients. Hence these implementations open the door for the off-path attack. Even if the smart door keypad lock application is protected with the standard IoT authentication protocol such as DTLS, the injection of a fake pin code by the off-path attack is possible in such implementations.

Furthermore, we demonstrate the attack in the presence of a firewall and a two-factor authentication method for the remote CoAP client. The consequences of fake information injection

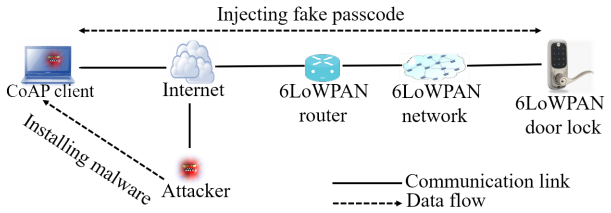


Fig. 1: Off-path attack model on CoAP protocol

on smart door keypad lock are high compared to smart light or smart metering. Therefore, we choose smart door keypad lock as our usecase to explore the vulnerability of the CoAP protocol. **Our contributions** are summarized as follows:

- We identify the "***Request Spoofing***" vulnerability of CoAP by exploiting the remote server access support of CoAP implementation along with the IP spoofing vulnerability of CoAP using an off-path attack. We observe that most available CoAP implementations are not performing the validation of remote CoAP clients. Even the widely used Python implementation of CoAP (CoAPthon [8]) has this vulnerability. It is thus a critical vulnerability in CoAP implementations which has not been reported thus far.
- We analyze and demonstrate the limitations of DTLS and firewall respectively to defend against the identified off-path attack. Also, we experiment the attack in a two-way authenticated environment and present a detailed analysis of our results to show that such authentication method alone cannot defend against such attack.
- We provide a detailed description of why an off-path attack is possible so that it can be repeated by researchers in their lab environment for different applications. More specifically, the difficulty of distinguishing the packets from an actual CoAP client and the attacker is discussed in detail. Also, the possible solutions to prevent the attack are discussed in detail. Without adding much overhead to IoT devices, we propose a simple machine learning approach to detect the abnormal behavior of the compromised CoAP client for preventing the identified off-path attack.

The rest of the paper is organized as follows. Section II provides a background information on how the CoAP protocol and WebSocket works. Section III describes the works related to the off-path attack. Our testbed architecture is described in Section IV. Section V outlines how the identified off-path attack works. The limitations of existing IoT security protocol such as DTLS, firewall, and two-factor authentication is explained in Section VI. Section VII outlines the potential defense mechanisms including a machine learning approach to mitigate the off-path attack. Finally, we conclude our analysis with a discussion in Section VIII.

II. Background

A. *6LoWPAN protocol stack with CoAP*

6LoWPAN communication protocol used in Low Power Area Networks (LoWPAN) has the ability of adopting IPv6

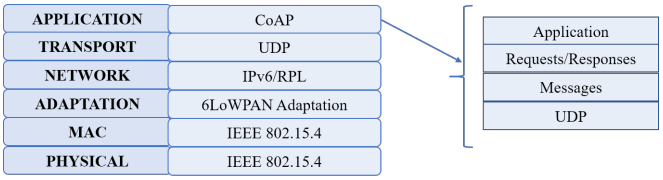
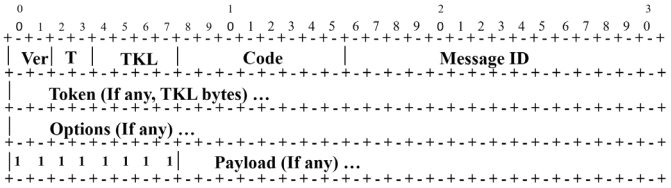


Fig. 2: 6LoWPAN protocol stack with abstract layering of CoAP



Ver (Version) :CoAP version number. Messages with unknown version number will be ignored
T (Type) :Indicates the type of message such as CON, NON, ACK, RST
TKL (Token Length) :Indicates the length of variable token length field. Usually 0-8 bytes. 9-15 bytes are reserved and processed for message format error.
Code :Request (1-10) or response method (40-255)
Message ID :16 bit indicator to detect duplicate messages. Used to match ACK/RST messages to CON/NON messages.

Fig. 3: CoAP packet

Internet protocol [3] [9] [10] [11]. Since IPv6 has large address space, it can subsume many constrained devices into the Internet. Fig. 2 shows the presence of CoAP and associated protocols in different layers of 6LoWPAN and explains the application layer in detail. 6LoWPAN supports CoAP protocol in its application layer and UDP in transport layer [12] [13]. 6LoWPAN adaptation layer does the job of header compression that grants the communication of IPv6 packets over the IEEE802.15.4 network.

6LoWPAN adopts the bottom-most two layers (Physical and MAC layers) from IEEE 802.15.4 standard and supports 127 bytes of data. Although link-layer security inside a LoWPAN (employing the 128-bit AES encryption in IEEE 802.15.4) provides some protection, communication beyond LoWPAN Routers is still vulnerable which increases the need for end-to-end security at the application layer [3].

B. *CoAP message format and its functionality*

A CoAP client, which needs a reliable transmission, sends a request CON message to the CoAP server and gets an ACK message back.

NON messages from the client do not get an acknowledgment back from the server, but still, have MessageID to avoid duplication of the same message. If the server is not able to process the CON message, then it replies with RST message instead of ACK.

Fig. 3 shows the CoAP packet format. CoAP packet consists of four bytes header information followed by optional token values and payload. The version field (two bits) indicates the CoAP version number. The CoAP server automatically ignores the CoAP messages that are having an unknown version number. The type field indicates the type of CoAP messages such as CON, NON, ACK, and RST. TKL field specifies the

Potential attacks on CoAP	Description of Attack	Possible Countermeasures
Attack on Complex protocol parsers	Crash a node remotely and execute arbitrary code remotely on parsers	Reducing parser complexity; moving much of the URI processing to CoAP clients; Care must be given to CoAP access control implementations
Man In The Middle attack on proxies	Breaks the confidentiality and integrity of the CoAP message by breaking any IPSec or DTLS protection on a direct CoAP message exchange through caching of proxies	Access control of resources must be considered. Do not perform caching on requests that have lesser transport-security properties
Amplification	The attacker attempts to overload a victim packet by turning a small packet into a large packet leading to a denial of service attack	Make the constrained network to generate a small amount of traffics. CoAP server can use Slicing/Blocking modes of CoAP. Limiting the support of multicast requests to specific resources.
IP address spoofing	Attacks the endpoint and even a whole network by spoofing the response and multicast request messages.	Response spoofing: by choosing the randomized token in the request. Use the security mode of communication. Request Spoofing: the focus of this paper.
Cross Protocol Attacks	Attackers send and receive a message to the CoAP endpoint	Strictly check the syntax of the received packets. Authorization of endpoints needed
Timing attacks	As constrained nodes are low in processing power, the attack can happen on cryptographic key generation and recovery of keying materials.	Care must be taken on the implementation of cryptographic primitives.

TABLE I: Potential attacks on CoAP

length of the variable token field which is followed by the CoAP header information. The usual length of TKL field is 0-8 bytes. Code field represents the unique code for request and response messages. The Message ID is a 16-bit indicator used to detect duplicate messages and to match ACK/RST messages to CON/NON messages. The token value will be 0-8 bytes used to correlate request and response messages. In the presence of payload, payload field is prefixed with one-byte payload marker (0xFF) which denotes the end of options and the start of payload. Since CoAP relies on UDP in its transport layer, it uses stop and wait scheme for re-transmission of CON messages and duplicate detection for both CON and NON messages to increase reliability between CoAP endpoints.

The endpoint (node) is determined by its IP address and UDP port number in the case of "NoSec" mode. CoAP-to-CoAP proxy maps a CoAP request to a CoAP request which means both the client and server uses the CoAP protocol to communicate. A CoAP-to-CoAP forward proxy is acting on behalf of the CoAP client to make requests to the CoAP server. CoAP-to-CoAP reverse proxy acting on behalf of the CoAP server to give the resources to the CoAP clients. Reverse proxy builds a namespace so that the client will get more control over where the request goes by embedding information such as host IP address and port number of the URI (Unified Resource Identifier) to direct the request to its intended resources. CoAP URI consists of URI-Host (Host IP address), URI-Port (transport layer port number), URI-Path (Absolute path of the resource) and URI-Query (Argument parameterizing the request). UDP port is the port where the CoAP server locates.

C. *Potential attacks on CoAP*

Potential attacks on CoAP [4] are stated in Table I. Our work focuses on IP spoofing, more specifically on "***Request Spoofing***" vulnerability, which is not even described/captured in RFC7252, of CoAP by exploiting the remote server access support of CoAP implementation. According to RFC7252, spoofing attacks on CoAP "***response messages***" can be performed as follows.

The malicious programmer prevents the CoAP client from re-transmitting the CON message by spoofing the ACK message and stops the actual response of the CoAP server. Another method is making the CoAP server disabled so that it is unable to receive any CON messages. This can be done by spoofing the RST messages.

The attacker spoofs the NON messages by making the CoAP server unable to receive any CON messages by spoofing RST messages. Spoofing the entire response is done by changing the entire payload of CoAP message with fake information. Spoofing a multicast request can lead to congestion in the network, DoS (Denial of Service) attack, and intentionally wake up the constrained device from sleeping (energy depletion attack).

However, we spoof a CoAP request CON messages to cause significant damage to the smart door keypad lock system, even before the actual user realizes the presence of an attack. This attack is different from the potential attacks identified above.

III. Related Work

Security issues caused by off-path attacks on TCP and DNS are very well researched and how they compromise challenge-response defense are analysed in [14] [15] [16] [17] [18]. Gilad et al. [19] showed that TCP injection is possible by the following method. Off-path attackers learn the connection sequence numbers of both the client and server in a TCP connection by exploiting a globally increasing IP-ID counter of Windows machine. Moreover, they suggested the use of security protocols such as SSL/TLS or IPsec to defend against such off-path attacks.

In [20], Gilad et al. experimented a practical off-path TCP-injection attack which allows web-cache poisoning. They suggested to modify the client port selection algorithm at NAT (Network Address Translation) level and deploy cryptographic methods such as SSL/TLS at the server side as a defensive mechanism against such off-path TCP injection attack. However, we analyze and present in Section VI that SSL/TLS (i.e., DTLS) based defensive mechanisms do not

work in our identified attack. Hence, we use a machine learning based approach to monitor the malicious activities of the compromised client and defense against our attack.

Source Port Randomization (SPR) is the mitigation technique for an off-path attack on TCP [21]. Fernandes et al. [22] used the vulnerability of the mobile app to launch the pin code injection attack on the smart door lock. However, our off-path attack uses the remote server access support of CoAP implementation to launch the off-path attack thereby injecting the fake password into the Smart Door Keypad lock system. Hence, the following defensive approaches do not work for our attack. As a mitigation technique of an off-path attack on TCP, DNS resolvers send a challenge - 16-bit TXID field with the request and expecting the same TXID in the response. Unpredictable port randomization of the client and dropping the connection with too many empty ACKs at the server side are the defensive mechanisms supported by a majority of the resolvers against the off-path attack. References [15] [23] used side channels for port prediction in order to execute the off-path attack.

In summary, existing works on the off-path attack for TCP focus on how such response spoofing is executed along with their countermeasures. However, our off-path attack on CoAP analyses how a "request spoofing" causes the damage to the constrained devices. In the case of CoAP; SPR technique is built in with the protocol, making it more vulnerable to "request spoofing". Since the off-path attacker injects the fake password value with the request CON message; the server cannot distinguish the spoofed packet from the original packet. Our article is the first and novel approach to explore the vulnerability of CoAP through the off-path attack. And, we identified the request spoofing vulnerability of CoAP by exploiting the remote server access support of CoAP implementations. Also, we have tested a machine learning model on our private network to predict the abnormal behavior of the infected CoAP client.

IV. Our Testbed Architecture

Our testbed architecture has two major parts. The first part focuses on the construction of testbed and the second part provides Two Factor Authentication (2FA) to the CoAP user. The Communication flow of our testbed starts with the user entering his/her credentials to authenticate themselves using 2FA mechanism as discussed in Section IV-B. Then the authenticated user can open the door, close the door, or even change the passcode of the door remotely. We enabled port-forwarding mechanism in Internet Protocol Router (IPR) to achieve CoAP communication with the 6LoWPAN network. 6LoWPAN border router acts as a bridge between Internet packets and 6LoWPAN packets by compressing Ethernet packets into IEEE 802.15.4 packets. Section II-A describes the 6LoWPAN protocol stack and the border router. We strongly assume that the off-path attacker enters the private network of the smart home to inject the fake passcode into the 6LoWPAN door lock, as discussed in Section V-A. Network traffic capture tool captures the network traffic of the CoAP remote clients and attacker to train the Machine Learning (ML) models in order to analyze and predict the behavior of the user agents.

A. ***Construction of Testbed***

The testbed consists of a constrained 6LoWPAN IoT network and the Internet. We explore the vulnerability of CoAP protocol and investigate how a smart door lock application may be exploited by the adversary injecting malicious information. Our overall implementation is based on Raspberry Pi. A similar implementation is possible using ARDUINO platform. Fig. 4 shows the architecture components and the communication between them.

We use the following hardware components to establish the 6LoWPAN IoT network. Raspberry Pi 3 Model B+ is used for a door lock and a border router. Raspberry Pi 802.15.4 radio from openlabs [34] is used to enable 6LoWPAN communication over 802.15.4 to the door lock. GrovePi is placed on top of the Raspberry Pi and connects the Grove LCD. GroveLCD is used to display the smart door lock status to the user. The keypad allows the user to enter the password and the LEDs reflect the user action on the door lock. 6LoWPAN IoT network establishment has the following two modules.

Module 1: *Building a CoAP enabled 6LoWPAN smart door keypad lock to perform the off-path attack.*

There are two ways to have the CoAP enabled smart door keypad lock system which follows the 6LoWPAN/CoAP communication stack protocol. One is to use a commercially available product such as IKILOCK [35]. Moreover, the other way is to build the prototype of the smart door keypad lock. At the time of establishing a test-bed, commercial 6LoWPAN smart door keypad lock was not available in our region as per our specifications. So we developed a prototype of 6LoWPAN smart door keypad lock using hardware components as mentioned earlier. Fig. 5 shows our smart door keypad lock prototype and explains the process flow of the prototype.

We opted Python programming language to develop the software of our smart door keypad lock prototype. The prototype software has three separate components. Firstly, a Python program for handling the hardware components. Secondly, a CoAP server which deals with how to access the resources of the prototype and communicating with a database. Thirdly a password management database which stores a password of the smart door keypad lock as shown in Fig. 6.

The Smart door keypad lock prototype displays two options such as "Enter Password," and "Change Password" on its LCD. The user will choose option A and enter the four-digit password if he wants to enter the home. The prototype accepts the four-digit password and validates it with the Password Management Database. The door will open once the validation of password returns success and the LCD displays the "Access Granted" message. Otherwise, the main options start to display on the LCD followed by the error message "Try again Later".

On the other hand, option B is for changing the password of the smart door keypad lock. To change the password, the user provides the old password for a credibility check with the database and then sets the new password which will be updated in the password management database by the CoAP server. The connection between the CoAP server and a password management database is established by giving the access control list credentials such as username, password

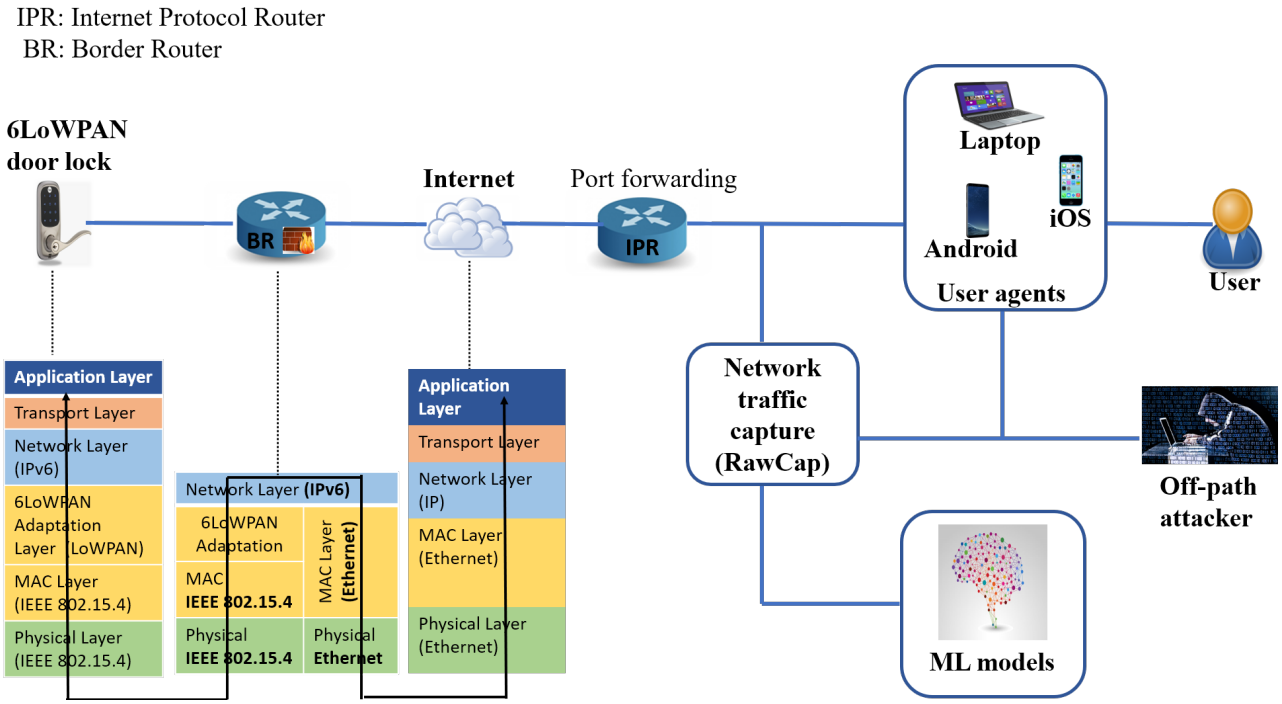


Fig. 4: Our testbed architecture

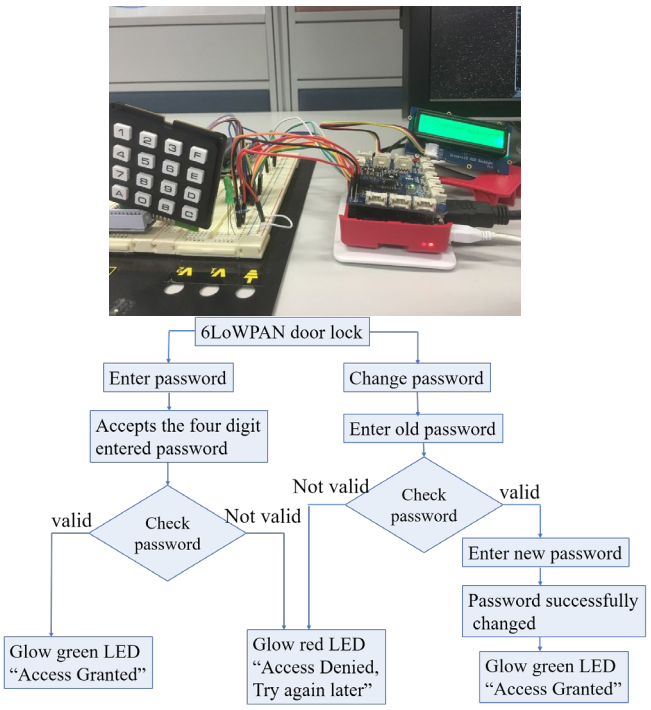


Fig. 5: Smart door keypad lock prototype and its process flow

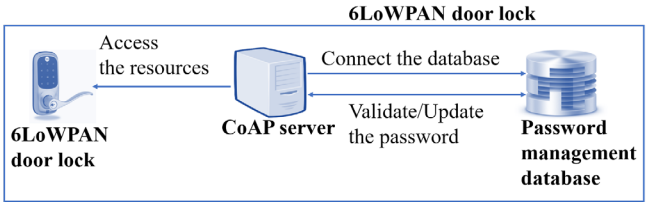


Fig. 6: Software components of smart door keypad lock

and the name of the database. Since MySQLdb implements the Python database API v2.0, we have used MySQLdb as an interface for the connection between a MySQL password management database and the CoAP server.

There are many open source CoAP protocol implementations available. Among them, we preferred CoAPthon implementation [8] of CoAP server and FireFox ESR with copper add-on as a CoAP client. The Firefox Copper has been disabled by Firefox after Firefox Quantum has been announced. We can use "Copper for Chrome (Cu4Cr) CoAP user-agent [36]" as a CoAP web interface to interact with the 6LoWPAN smart door lock. However, if still want to use Firefox browser, downgrade the Firefox Quantum to Firefox ESR 52.7.2. Since CoAPthon has the implementations of more CoAP features (CoAP server, client, forward proxy, reverse proxy, observe, multicast server discovery, CoRE Link Format parsing and block-wise transfer) as described in RFC7252 compared to other available CoAP implementations. We modified their code found in [37] according to our resources available in the 6LoWPAN smart door keypad lock prototype.

We execute our CoAP server on 6LoWPAN smart door keypad lock prototype and the CoAP client on the laptop. Also, we use Aneska and IoT-CoAP Android mobile apps which are available in the play store and app store as CoAP clients to access the CoAP resources of the 6LoWPAN prototype. The CoAP user gets the existence of the CoAP resources through CoAP resource discovery. Our prototype has three CoAP resources such as "OptionA", "OptionB"

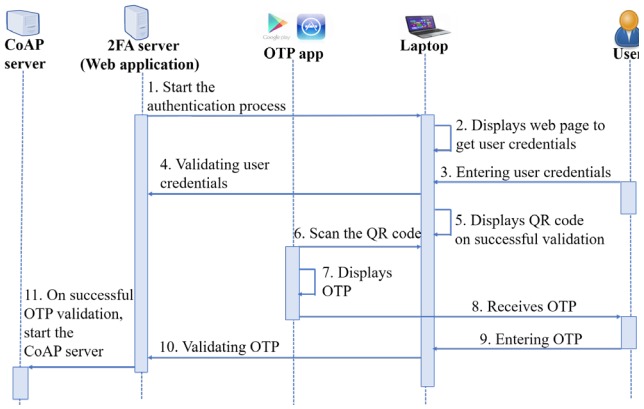


Fig. 7: Two factor authentication process for CoAP user

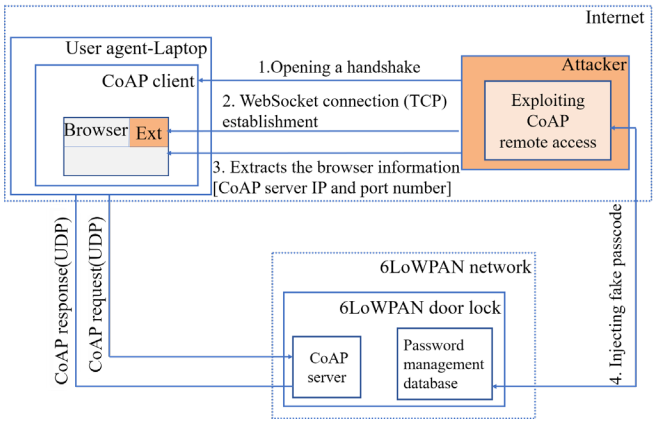


Fig. 8: Off-path attack scenario

and "Password". The user gets the status of the door, by accessing the OptionA and OptionB CoAP resources through GET command. If the door is not used by the user at that moment, the corresponding resources return OptionA/OptionB is in standby mode. Otherwise, GET command returns "The Door is Opened" or "Access Denied" messages respective to the CoAP resources accessed. The "Password" CoAP resource is used to change the password of the 6LoWPAN smart door lock prototype remotely through the PUT command of CoAP protocol. The users of the prototype can successfully change the password remotely using their Android or iOS mobile apps.

Module 2: *Building a 6LoWPAN border router on Raspberry pi.* For the 6LoWPAN border router, we use the native border router GitHub code found in [38]. We changed the RADVD configuration file by setting the prefix of the IPv6 address of the constrained 6LoWPAN network. When the RADVD server starts at the border router, the 6LoWPAN prototype within the constrained network is assigned with the IPv6 address using Neighbor Discovery Router Advertisement (RA) messages. This facilitates the CoAP client to access the CoAP resources of the 6LoWPAN prototype from the internet.

B. Two Factor Authentication (2FA) for the CoAP user

Fig. 7 explains the communication flows of remote CoAP user authentication process-2FA in detail. We choose the 2FA method code which is available online [39] in conjunction with CoAP. CoAP server starts its service only after a successful 2FA user authentication process. This method of IoT user authentication includes authentications of regular user password and a one-time token thereby increasing the level of security in a web application. The one-time password is based on the Time-based One-Time Password (TOTP) algorithm [40] and changes every 30 seconds. The user passwords are not stored as plaintext in the database. Instead, the hash value of the password is saved for verification. Upon successful password verification, the user receives a QR code to proceed further. This QR code is scanned by a Free OTP app which is available in android and app store. The user is authenticated when providing the TOTP within its validity time, thereby getting the access of CoAP resources. This TOTP is not a CoAP responses payload. After successful user authentication, CoAP server of our 6LoWAPN prototype will be enabled, and by hitting the "Discover" button, we get to know the available CoAP resources on the CoAP server. Communication flow-5 of Fig. 4 represents the CoAP resources accessed by CoAP clients.

We agree that the 2FA implementation in the context of CoAP for low power devices is computationally complex. Also, the CoAP endpoint user has to wait until the authentication process is finished and getting the response from the CoAP server. However, we have implemented 2FA with CoAP to provide the endpoint security to the CoAP client. We firmly believe that the existing standard security protocol DTLS in the context of CoAP for constrained devices provides data integrity between the communicating endpoints as per RFC 7252 and has limitations related to IP Spoofing and cross-protocol attacks as we discussed clearly in section IV-A.

V. Our Off-Path Attack

We assume that the attacker gets into the user's device (laptop) to spoof the IP address and communicate with the desired 6LoWPAN device. To launch the off-path attack, the attacker needs IP address and port number of the CoAP server. Even if the port number is not mentioned in the browser, the attacker uses the default port number 5683, of CoAP server to execute the attack.

A. Off-path attack scenario

Fig. 8 explains the attack scenario of off-path attack which involves the attacker, user agent-laptop and a 6LoWPAN door lock in detail.

Make a connection with CoAP Client: An attacker reaches the victim's machine (the client) by making it install a malware thereby the attacker loads a JSON file on the browser. This creates an extension in the client's browser. The attacker's code running inside the client's machine is making a TCP 3-way handshake (SYN, SYN-ACK, ACK) connection with a CoAP client whenever the malicious extension bar gets clicked by the user. Thereby a TCP socket connection is opened for malicious communication. We use WebSockets and asyncio libraries of Python to establish such a malicious connection. The execution

of off-path attack follows the establishment of a WebSocket connection between the malicious code and the CoAP client [41] [42]. CoAP proxy at the WebSocket endpoint forwards the request to the CoAP server with a CoAP UDP endpoint. Such a malicious connection stays alive until the malicious program terminates.

Launching an off-path attack: The malicious code running behind the browser extracts the browser information upon successful establishment of a malicious WebSocket connection. The attacker receives the CoAP server's IP (destination IP) address and a destination port number with the help of the victim browser's extension bar. Then the attacker attempts to inject the malicious passcode value to the CoAP server by exploiting remote server access support of CoAP protocol. The off-path attack happens following a sequence of message transmission which uses TCP and WebSocket protocol.

Moreover, the attack is possible despite the usage of the addresses IPv4 or IPv6 user communication with the CoAP server. The attacker updates the password database of the 6loWPAN prototype by sending the CoAP request with his/her own hard-coded password value without the knowledge of the actual user. By this way, the attacker creates the "***Request Spoofing***" vulnerability on CoAP protocol to inject the fake password in the desired 6LoWPAN device's database. Once the attacker succeeds in his/her off-path attack, he gets the full access of the 6LoWPAN smart door keypad lock.

Such an off-path attack can be performed in any commercial device which follows the 6LoWPAN/CoAP protocol stack if the attacker reaches the authenticated/non-authenticated client's machine. Our work will be enhanced in the future, with an IoT testbed that includes a large number of constrained IEEE 802.15.4 sensor nodes to check the scalability, packet losses, and packet delay of constrained devices.

B. 6LoWPAN-CoAP packet analysis with off-path attack

We analyze 6LoWPAN-CoAP packet based on the CoAP message format as discussed in Section II-B. We presented the actual user CoAP request packet in Fig. 9 and the off-path attacker request packet in Fig. 10 which are captured by Wireshark tool.

CoAP over WebSocket: If the CoAP communication happens over WebSocket, then the message packet will have zero in its length (Len=0) field thereby the version field of CoAP packet is suppressed with Len field of WebSocket [42]. Also, CoAP over WebSocket transmission does not distinguish CON and NON messages and does not provide ACK/RST messages. However, the off-path attack's CoAP request packet does have all the fields as in CoAP over UDP transmission packet, and it is being sent as a CON message. Moreover, it receives the ACK message from the CoAP server. Since the attacker's request packet is very similar to CoAP client request packet, it is hard to distinguish the poisoned packet from the actual client packet.

Off-path in TCP and SPR: The conventional method to prevent the off-path attack in TCP and UDP is to include a nonce value with the client's request so that validation of the request message is possible when receiving the response

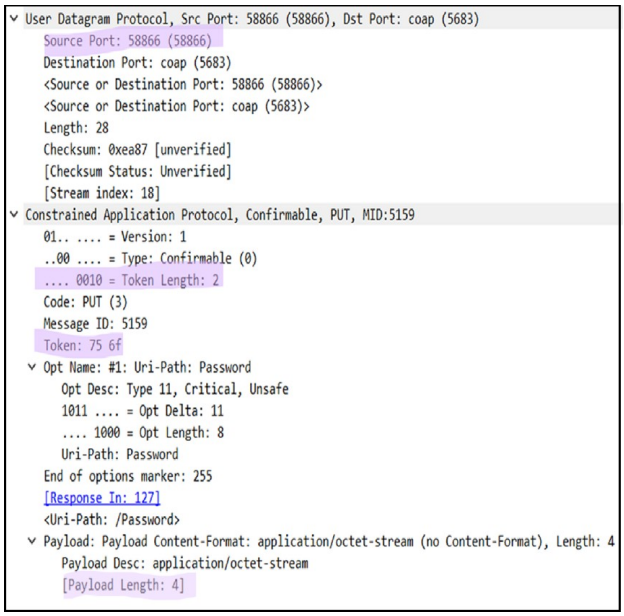


Fig. 9: Actual user's CoAP request packet captured by Wireshark

message. Unlike the TCP injection attack, the CoAP off-path attacker is spoofing the request message (injecting the fake passcode), not the response message from the server. So the countermeasure of including the nonce value with the request message is not enough in our case. Moreover, CoAP supports SPR technique in its clients; making the protocol more prone to off-path attack. From Fig. 9 and Fig. 10; we can not distinguish the CoAP packets based on its origin (either the user or the hacker) since both of the packets are having random source port numbers.

Server Port Number: According to RFC6335 [43] IANA (Internet Assigned Numbers Authority) has assigned 5683 as a default port number for CoAP server. If the UDP port is not given in the URI or the field is empty, then the default port 5683 will be assigned as a CoAP server port. Hence CoAP server port is explicit to the endpoints; leads the off-path attacker easy to guess the CoAP server port number.

Randomized token number: A randomized token number is generated when the CoAP messages are not protected by the transport layer security to mitigate the response spoofing [42]. CoAP client must generate a randomized token ID for every request it makes to the CoAP server to match the request and response. The token ID also referred as "request ID". This token length would be up to eight bytes, and at least 32 bits of token length must be used if the CoAP endpoint is connected to the Internet. CoAP server will echo the same token value with its response. In some situation, the token ID generation is not mandatory. For example, when the CoAP client sends a serial request to the CoAP server and CoAP client gets piggybacked responses from the CoAP server. Hence it is not possible to differentiate the malicious and original CoAP request packets based on randomized token ID value.

Payload length field: According to the RFC7252, in the spoofed CoAP packet format; after the one-byte payload

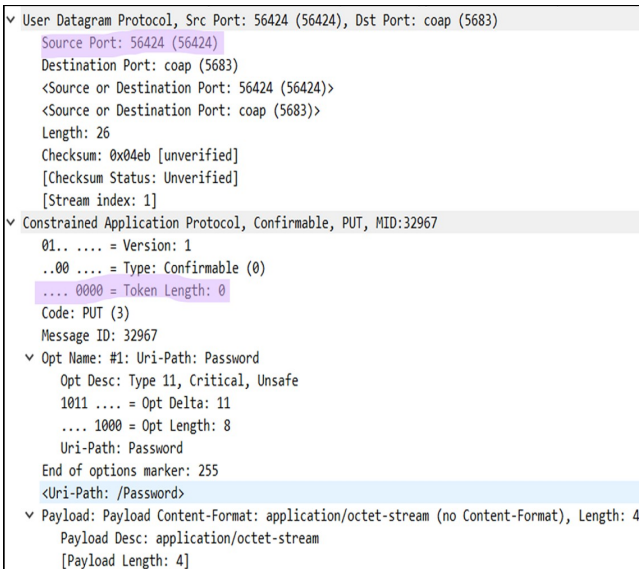


Fig. 10: The attacker's CoAP request packet captured by Wireshark

marker (0xFF), the payload length must be zero. Then automatically the packet will be thrown out for message format error. Also, the code field must be set with the reserved class (1, 6, or 7). However, our captured off-path attacker's packet does have the payload length the same as the original CoAP packet.

The CoAP server rejects the packets which are having random token length beyond 8 bytes (i.e., 9-15 bytes). However, CoAP allows same token length value for different client port and supports SPR. This makes an off-path attack very easy to implement.

VI. Limitations of Existing IoT Security Protocol and IoT User Authentication

We analyzed and observed that the existing IoT security protocols are not giving protection against our implemented off-path attack. They are thereby creating the loophole/ vulnerability in CoAP protocol. This section describes the limitations of DTLS, limitations of user authentication, and the limitations of the firewall briefly.

A. *Limitations of DTLS*

DTLS (Datagram Transport Layer Security) security protocol is the most common method used in conjunction with CoAP to reduce the communication vulnerability of IoT [24] [25]. Existing works [26] [27] [28] [29] use standard security protocol DTLS to secure the payload of CoAP communication. However, DTLS does not provide security against the cross-protocol attack if the same DTLS security connection is used to carry the data of multiple protocols [4]. All the modes of DTLS are not applicable for all constrained devices. Running of DTLS is impossible on class 0 devices and nearly impossible to be hosted on class 1 devices (10k RAM/100k ROM).

Implementation complexities of DTLS in constrained devices are high. Initial handshake overhead is high. DTLS adds 13 bytes of per-datagram overhead, excluding initialization vectors/nonce, integrity check values and padding values of cipher suite, which increase the overhead of LoWPAN devices. DTLS does not applicable to group keying communication (multicast communication). Since UDP protocol will not verify the source address of the request packet, CoAP is vulnerable to cross-protocol attacks with the fake source address. UDP based protocols are relatively easy targets for the cross-protocol attacks. Also, the probability of network delay is higher for asymmetric based handshake security protocols [30] [10].

Cross-protocol attacks are possible in UDP based communication if the security properties of the CoAP server and client rely only on the process of checking the source IP address. Also if we use proxies such as HTTP to CoAP or CoAP to HTTP, the transport layer security called DTLS has to be terminated at the proxy [31]. So we need a security protocol with features that offer defensive mechanisms against the combination of vulnerabilities such as IP spoofing and cross-protocol attacks of CoAP.

B. *Limitations of user authentication*

The possible best practice to protect the remote server access capability in the Internet world is protecting the network by VPN (Virtual Private Network), tunneling the network traffic [32] and know the user (the process of authenticating the user). We strongly assume that the attacker somehow reached the authenticated client's machine and started the malicious server program to extract the browser information. The widely used mitigation technique on the Internet against remote server access exploitation is by knowing the user using two-factor authentication mechanisms [33]. None of the research papers addressed CoAP user authentication through 2FA (Two Factor Authentication) method. We implemented it in our testbed as in Section IV-B.

Despite a secured 2FA CoAP user authentication, the off-path attack is possible, and we demonstrated it since we access the CoAP server resources by the IP address and port number of the 6LoWPAN device. Moreover, the memory occupancy by the 2FA authentication process is high on 6LoWPAN constrained devices, and obviously, it is adding additional initial computational time (user validation time) on the normal communication of CoAP.

C. *Limitations of firewall*

As discussed in Section V, the off-path attacker spoofs the IP address of an authenticated client. So filtering the packets based on their source IP address using the firewall is not going to prevent the off-path attack. Specifically, from the Section V-B we observed that the user CoAP packets and the poisoned packets are following the same protocol stack for the communication. Thereby we cannot merely deny the incoming UDP port transmission on the firewall. Though we installed and configured the UFW (Uncomplicated Firewall) firewall to reject the standard UDP port communication on our testbed, the off-path attack by-passes the firewall and gains access

to the 6LoWPAN doorlock device. We believe that the SPR feature of the CoAP protocol makes the 6LoWPAN network unable to use the firewall to defend against the off-path attack.

VII. Mitigation Technique

Since the IoT traffic captured from our testbed has no significant differences between the authenticated client's CoAP packet and the attacker's packet as discussed in Section V-B, leads us not to stop the anomaly traffic with the rule-based approach. Also, the existing IoT security algorithms are failed to determine this kind of attack, as discussed in Section VI. For these reasons, we have implemented the Machine Learning (ML) model in CoAP client's machine (Laptop with Intel Core i7) to identify and predict the abnormal behavior of the infected CoAP client's network traffic. We can also allocate a dedicated machine inside the private network (for example, smart home) to monitor and predict the fake network traffic.

The technique of analyzing network traffic data using machine learning algorithms getting popular among the researchers [44], [45]. Unsupervised ML algorithms do not require pre-labeling of the dataset and create clusters on the network traffic data set based on their similar characteristics. Whereas our dataset needs a binary classification in the form of, whether the network traffic is malicious or normal traffic, we choose supervised learning ML methods to create an ML model for outlier prediction.

Random Forest supervised machine learning algorithm is used in [46] [47] to identify the IoT device traffic from non-IoT device traffic based on the characteristics of network traffic, it generates. Detecting the IoT nodes in wireless sensor network also presented in [48]. However, we used a machine learning algorithm to classify the traffic of the infected CoAP client which is trying to access the CoAP IoT server maliciously. Multiple experiments carried out on our IoT testbed network dataset with different ML algorithms to find out the best ML model. We have used the WEKA machine learning tool [49] to create the ML models.

A. Dataset

The training dataset (internal traffic of the infected client) which is captured by Rawcap tool, has "2676" instances. The dataset is converted into Attribute-Relation File Format (.arff) for processing using the WEKA tool. There is no feature selection algorithm applied to our IoT network traffic dataset since we are interested in all the features of the dataset except the number of attributes. We add one attribute called "Threat" to specify the label of each traffic instance. After labeling the training dataset, "Threat" attribute weight is distributed among the classes "No" and "Yes" with the weights "2449.0" and "227.0" respectively. It is obvious that the real-time IoT traffic dataset has the unbalanced weight distribution among the classes. The attribute "Protocol" weight is shared among "TCP", "HTTP" and "Websocket" with the corresponding weight of "2611.0", "26.0" and "39.0" respectively.

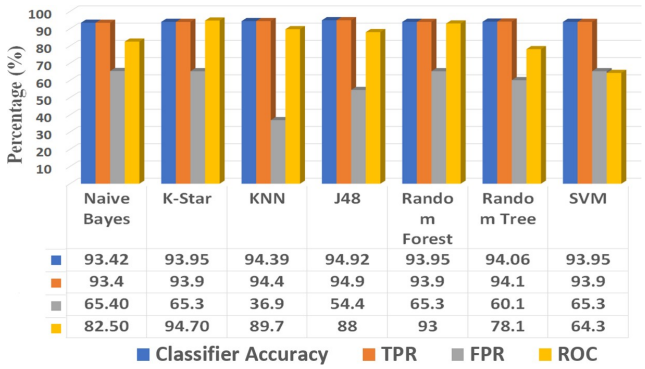


Fig. 11: Training phase: performance comparison of different ML classifiers

B. Training phase: Building ML models

We applied the ML algorithms such as Naive Bayes (NB), K*, K-NN (k-nearest neighbors), J48, Random Forest, Random Tree and Support Vector Machine (SVM) algorithms to build ML models and presented a graph in Fig. 11 with their performance comparison statistics. We use 10-fold cross-validation method to evaluate the performance of our ML models. This approach divides the data set of samples into ten groups/fold. For each group, keep the group as a test dataset and treat the remaining as the training dataset to evaluate the ML model. Hence keep the evaluation score for the corresponding ML model [50].

Classifier accuracy, True Positive Rate (TPR: number of examples predicted positive that are positive), False Positive Rate (FPR: number of examples predicted positive that are actually negative) and ROC (Receiver Operating Characteristic) is the parameters used to compare the performance of different ML classifiers. In our dataset, the payload and the structure of the malicious packet are very similar to the original CoAP packet as described in Section V-B. Generally, the FPR value is high, when the dataset is going through an intrusion detection with anomaly-based technique. We follow the hybrid of anomaly detection and signature-based intrusion detection so that we could detect unknown attacks in IoT.

Ranking an instance based on the area under the ROC curve is a widely accepted parameter to evaluate the ML algorithms [51]. We present the values of Classifier accuracy, FPR, TPR and ROC parameters for our dataset by applying ML algorithms such as NB, K*, K-NN, J48, Random Forest, Random Tree and SVM in Fig. 11. From the observation of Fig. 11, the ML algorithms such as K-Star and random forest have higher accuracy and ROC value as well.

Naive Bayes (NB): Naive Bayes ML classifier uses a probabilistic approach based on Bayes theorem with independence assumptions among the features of a dataset. Malicious traffic is classified based on the prior knowledge of the condition that might be related to the occurrence of suspicious network traffic. NB classifies the dataset according to every attribute into two intended classes. For our dataset, the obtained NB classifier model accuracy is "93.42%" and "82.50%" ROC area under the curve.

K-Star: K* is an instance-based classifier, that is the class of a test instance is based on the class of those training instances similar to it, as determined by some similarity function [52]. For our dataset, the obtained K* classifier model accuracy is "93.95%" and "94.70%" ROC area under the curve.

K-NN (k-nearest neighbors): Unlike Naive Bayes classifier, K-NN is not a probability-based classifier. However, K-NN classification is based on non-parametric statistics such as descriptive statistics and statistical inference of the instances. Our K value is 1 with Linear NearestNeighbour search which uses a Euclidean distance function to find out the nearest traffic. For our dataset, the obtained K-NN classifier model accuracy is "94.39%" and "89.7%" ROC area under the curve.

J48: J48 is a tree-based statistical classifier which is the implementation of the C4.5 algorithm in the Weka data mining tool. It produces a decision tree developed by Ross Quinlan [53]. In our training data set, the attribute "Protocol" acts as a root node. All the HTTP and WebSocket packets are classified as malicious traffic whereas the TCP packets are further classified based on time and length feature. For our dataset, the obtained J48 classifier model accuracy is "94.92%" and "88%" ROC area under the curve.

Random Forest: Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest [54]. Bagging with 100 iterations and base learner functions are performed on the training dataset. For our dataset, the obtained random forest classifier model accuracy is "93.95%" and "93%" ROC area under the curve.

Random Tree: To construct a tree it considers K randomly chosen attributes at each node. It performs no pruning. Also has an option to allow estimation of class probabilities (or target mean in the regression case) based on a hold-out set (back-fitting). KValue sets the number of randomly picked attributes. We choose the value of K is 0. If 0, int (log 2(#predictors) + 1) is used. For our dataset, the obtained random tree classifier model accuracy is "94.06%" and "78.1%" ROC area under the curve.

Support Vector Machine (SVM): Stable machine learning algorithm called SVM constructs hyper-plane to classify the instances of the training dataset. The larger functional margin between the hyper-plane and the training data points yields higher the accuracy of the classifier. Generalization of the classifier varies on the kernel function. We use Linear Kernel function $K(x, y) =< x, y >$ for the outlier detection. For our dataset, the obtained SVM classifier model accuracy is "93.95%" and "64.3%" of ROC area under the curve.

C. Testing phase: Threat Prediction

The test data set (internal traffic of the infected client captured by Rawcap tool) has "1078" instances. Among the 1078 instances, "210" traffic instances are malicious traffic. When applying this test data set to the previously constructed ML model, we got the malicious traffic prediction accuracy as shown in Fig. 12. Though K* and random tree classifier models

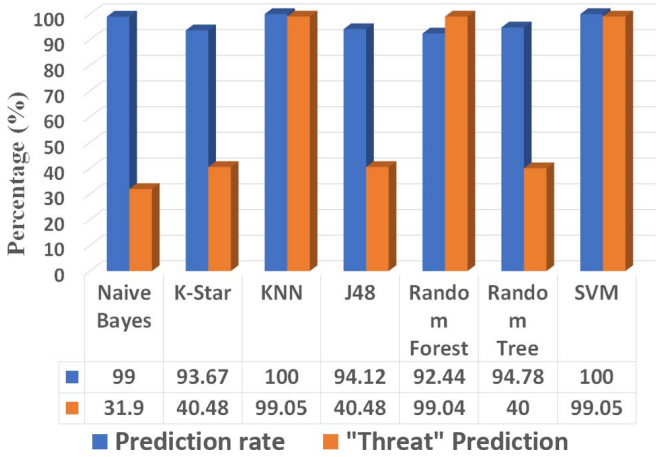


Fig. 12: Testing phase: Prediction accuracy comparison of different ML models

give higher classification accuracy and ROC area under the curve results, such algorithms failed to produce high "Threat" prediction accuracy. Comparing the prediction results, SVM and KNN models are giving "99.05%" of threat prediction accuracy. Hence we use SVM and KNN models for further real-time outlier detection in our IoT testbed. In the future, we will analyze the ML models in detail and train them with more datasets to predict the abnormal behavior of the IoT CoAP client.

VIII. Discussion and Conclusion

6LoWPAN and CoAP protocols are IETF standardized protocols. Sine 6LoWPAN supports IPv6; there is enough room for connecting more IoT devices in the WSN network, thereby supports scalability. Moreover, CoAP follows request-response architecture to maintain secure communication. This architecture will reduce the risk of having potential threats such as a drone controlling the entire light of the building found in [22] [55] which follows the publish-subscribe method of communication.

Our work in this paper is a concrete contribution to the IoT Cyber Security community to strengthen the security of the application layer protocol. Also, this work endorses the IoT device manufacturers to have machine learning model as an IoT network monitor to protect the IoT network against the off-path attack and the IoT devices' communication as well. With our observations and findings, it is recommended that the IoT industry can have more IoT products with 6LoWPAN-CoAP secure communication stack and ML model as a network monitoring tool in the IoT network.

Exploring the vulnerability of CoAP protocol and a mitigation technique is done through a case study of a smart door keypad lock system. While analyzing the CoAP packets, we observed that the spoofed CoAP request message packets have the randomized token number. This is the only difference between the spoofed CoAP request message and the normal CoAP request message of the CoAP server. However, in some situation, the token ID generation is not mandatory as discussed in Section V-B. Even with the presence of

authentication security protocols, we realize the attack after it has done the damage already.

The reason behind fake pincode injection in the 6LoWPAN/CoAP stack is, the implementations of CoAP protocol are not validating the remote CoAP clients. Hence creates "Request Spoofing" vulnerability, and if it is not treated properly, the damage for the 6LoWPAN device on the constrained network would be high. We experimented the off-path attack by constructing the prototype and launched the attack in an authenticated environment. Also, we presented the results of the countermeasure technique to mitigate the off-path attack using supervised machine learning model. In the future, we will automate the machine learning model to find out the abnormal behavior of live IoT traffic which help to detect and prevent the off-path attack. We believe that paying attention to "***Request Spoofing***" vulnerability and implementation method will help to improve the security against off-path attacks on CoAP.

ACKNOWLEDGMENT

This research is supported and funded by Data61, CSIRO, Marsfield, Australia.

REFERENCES

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[2] D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, 2011.

[3] Z. Shelby and C. Bormann, *6LoWPAN: The wireless embedded Internet*. John Wiley & Sons, 2011, vol. 43.

[4] Z. Shelby, K. Hartke, and C. Bormann, "IETF RFC 7252," *The Constrained Application Protocol (CoAP)*, 2014.

[5] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, no. 2, pp. 62–67, 2012.

[6] K. Hartke, "Observing resources in the constrained application protocol (CoAP)," Tech. Rep., 2015.

[7] A. Barth, "The web origin concept," 2011.

[8] G. Tanganelli, C. Vallati, and E. Mingozzi, "CoAPthon: Easy development of CoAP-based IoT applications with Python," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 63–68.

[9] P. Pongle and G. Chavan, "A survey: Attacks on RPL and 6LoWPAN in IoT," in *Pervasive Computing (ICPC), 2015 International Conference on*. IEEE, 2015, pp. 1–6.

[10] A. G. Roselin, P. Nanda, and S. Nepal, "Lightweight Authentication Protocol (LAUP) for 6LoWPAN Wireless Sensor Networks," in *Trustcom/BigDataSE/ICESS, 2017 IEEE*. IEEE, 2017, pp. 371–378.

[11] A. G. R. Arockia Baskaran, P. Nanda, S. Nepal, and S. He, "Testbed evaluation of Lightweight Authentication Protocol (LAUP) for 6LoWPAN wireless sensor networks," *Concurrency and Computation: Practice and Experience*, p. e4868.

[12] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[13] M. Bouaziz and A. Rachedi, "A survey on mobility management protocols in Wireless Sensor Networks based on 6LoWPAN technology," *Computer Communications*, vol. 74, pp. 3–15, 2016.

[14] Y. Gilad, A. Herzberg, and H. Shulman, "Off-path hacking: The illusion of challenge-response authentication," *IEEE Security & Privacy*, vol. 12, no. 5, pp. 68–77, 2014.

[15] A. Herzberg and H. Shulman, "Security of patched DNS," in *European Symposium on Research in Computer Security*. Springer, 2012, pp. 271–288.

[16] H. Herzberg, Amir and Shulman, "Socket overloading for fun and cache-poisoning," in *Proceedings of the 29th Annual Computer Security Applications Conference*. ACM, 2013, pp. 189–198.

[17] A. Klein, "OpenBSD DNS cache poisoning and multiple O/S predictable IP ID vulnerability," 2007.

[18] Z. Qian and Z. M. Mao, "Off-path TCP sequence number inference attack-how firewall middleboxes reduce security," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 347–361.

[19] Y. Gilad and A. Herzberg, "Off-Path Attacking the Web." in *WOOT*, 2012, pp. 41–52.

[20] A. Gilad, Yossi and Herzberg, "Off-path TCP injection attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 13, 2014.

[21] A. Hubert and R. van Mook, "RFC 5452: Measures for making DNS more resilient against forged answers," 2009.

[22] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 636–654.

[23] A. Hubert and R. van Mook, "RFC 5452: Measures for Making DNS More Resilient against Forged Answers, 2009," *URL https://www. ietf. org/rfc/rfc5452. txt*.

[24] T. Fossati and H. Tschofenig, "Transport layer security (TLS)/datagram transport layer security (DTLS) profiles for the internet of things," *Transport*, 2016.

[25] E. Rescorla and N. Modadugu, "RFC 6347: Datagram transport layer security version 1.2," *Internet Engineering Task Force*, vol. 13, p. 101, 2012.

[26] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN compressed DTLS for CoAP," in *2012 8th IEEE International Conference on Distributed Computing in Sensor Systems*. IEEE, 2012, pp. 287–289.

[27] A. Capossele, V. Cervo, G. De Cicco, and C. Petrioli, "Security as a CoAP resource: an optimized DTLS implementation for the IoT," in *Communications (ICC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 549–554.

[28] P. Urien, "Innovative DTLS/TLS security modules embedded in SIM cards for IoT trusted and secure services," in *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*. IEEE, 2016, pp. 276–277.

[29] S. Raza, T. Helgason, P. Papadimitratos, and T. Voigt, "SecureSense: End-to-end secure communication architecture for the cloud-connected Internet of Things," *Future Generation Computer Systems*, vol. 77, pp. 40–51, 2017.

[30] H. Kwon, J. Park, and N. Kang, "Challenges in deploying CoAP over DTLS in resource constrained environments," in *International Workshop on Information Security Applications*. Springer, 2015, pp. 269–280.

[31] "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth) draft-ietf-ace-oauth-authz-12," https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-12, accessed: 2018-06-22.

[32] Y. Gilad and A. Herzberg, "LOT: a defense against IP spoofing and flooding attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 2, p. 6, 2012.

[33] Q. Xie, D. S. Wong, G. Wang, X. Tan, K. Chen, and L. Fang, "Provably Secure Dynamic ID-Based Anonymous Two-Factor Authenticated Key Exchange Protocol With Extended Security Model." *IEEE Trans. Information Forensics and Security*, vol. 12, no. 6, pp. 1382–1392, 2017.

[34] "OpenLabs Raspberry-Pi-802.15.4-radio," http://openlabs.co/store/Raspberry-Pi-802.15.4-radio, accessed: 2017-12-07.

[35] "IKILOCK smart lock design for Smart Home," https://www.ikilock.com/, [Online;].

[36] "Copper4Cr Copper for Chrome (Cu4Cr) CoAP user-agent," https://github.com/mkovatsc/Copper4Cr, [Online;].

[37] "CoAPthon CoAP implementation," https://github.com/Tanganelli/CoAPthon.

[38] "Native 6LoWPAN Router Using Raspbian and RADVD," https://github.com/RIOT-Makers/wpan-raspbian/wiki/Setup-native-6LoWPAN-router-using-Raspbian-and-RADVD, accessed: 2017-11-07.

[39] M. Grinberg, "Two Factor Authentication with Flask," https://blog.miguelgrinberg.com/post/two-factor-authentication-with-flask, [Online; accessed 14-Sep-2018].

[40] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "Totp: Time-based one-time password algorithm," Tech. Rep., 2011.

[41] T. Savolainen, K. Hartke, and B. Silverajan, "CoAP over WebSockets," 2015.

[42] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets," Tech. Rep., 2018.

[43] Cotton, M and Eggert, L and Touch, J and Westerlund, M, "S. Cheshire," Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry," BCP 165, RFC 6335, August, Tech. Rep., 2011.

[44] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 4, pp. 1257–1270, 2015.

[45] O. M. Alhawi, J. Baldwin, and A. Dehghantanha, "Leveraging machine learning techniques for windows ransomware network traffic detection," *Cyber Threat Intelligence*, pp. 93–106, 2018.

[46] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis," in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 506–509.

[47] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, "Detection of Unauthorized IoT Devices Using Machine Learning Techniques," *arXiv preprint arXiv:1709.04647*, 2017.

[48] S. Althunibat, A. Antonopoulos, E. Kartsakli, F. Granelli, and C. Verikoukis, "Countering intelligent-dependent malicious nodes in target detection wireless sensor networks," *IEEE Sensors Journal*, vol. 16, no. 23, pp. 8627–8639, 2016.

[49] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[50] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[51] H. A. Güvenir and M. Kurtcephe, "Ranking instances by maximizing the area under ROC curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2356–2366, 2013.

[52] J. G. Cleary and L. E. Trigg, "K*: An instance-based learner using an entropic distance measure," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 108–114.

[53] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[54] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[55] E. Ronen, A. Shamir, A.-O. Weingarten, and C. OFlynn, "IoT goes nuclear: Creating a ZigBee chain reaction," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 195–212.