Model Checking M2M and Centralised IOT authentication Protocols.

Rekha H\textsuperscript{1} and Dr. M. Siddappa\textsuperscript{2}

\textsuperscript{1,2}Department of Computer Science and Engineering, Sri Siddhartha Academy of Higher Education, Tumkur-Karnataka, India 572103

E-mail: rekhasitcse@gmail.com

Abstract. It is very difficult to develop a perfect security protocol for communication over the IoT network and developing a reliable authentication protocol requires a detailed understanding of cryptography. To ensure the reliability of security protocols of IoT, the validation method is not a good choice because of its several disadvantages and limitations. To prove the high reliability of Cryptographic Security Protocols(CSP) for IoT networks, the functional correctness of security protocols must be proved secure mathematically. Using the Formal Verification technique we can prove the functional correctness of IoT security protocols by providing the proofs mathematically. In this work, The CoAP Machine to Machine authentication protocol and centralized IoT network Authentication Protocol RADIUS is formally verified using the well-known verification technique known as model checking technique and we have used the Scyther model checker for the verification of security properties of the respective protocols. The abstract protocol models of the IoT authentication protocols were specified in the security protocol description language and the security requirements of the authentication protocols were specified as claim events.

1. Introduction

Testing, the conventional way of validating a system, indicates the existence of bugs but it never guarantees its absence because the outcome of each test case is dependent on the specific feedback provided to the system during testing phases, but there may be certain conditions that testing cannot test. It’s difficult to test corner cases\textsuperscript{[1]}, and it’s not always practical. When a piece of software is delivered with flaws and a specific input is triggered, the entire system crashes, resulting in system failure. Exhaustive testing\textsuperscript{[5]} is difficult to do while testing a software system, and the errors discovered during integration testing are difficult to monitor and correct. Testing cannot detect functional or design flaws because it is easy to test desired behaviors but difficult to test undesirable ones. Many of these flaws in testing can be solved using formal verification techniques.

Formal Verification\textsuperscript{[2]} is the method of determining whether a device design meets all of the specifications/requirements when subjected to certain formal conditions/properties. Most requirements are defined in natural languages, such as English, which is an informal way of representing specifications\textsuperscript{[6]} that may have several ambiguities, since the client may not have a good understanding of the device requirements they want, or the client may not be able to define full functionality requirements. To address these ambiguities, formal notations can be used to describe a system’s specifications in mathematical notations.
Writing correct code is one of the most difficult challenges in any system’s development process, from design to implementation. The chances of introducing errors at each point of the system’s development are higher. Since the specifications are provided in informal language [6], the individual who is going to collect the system requirements from the client will not be able to understand exactly what the client needs. During the process of converting these vague criteria into a specification, the system designer may introduce design errors [7] in the system design itself, which may go unnoticed and are difficult to detect through testing. When it comes to system development, the main module (task) is broken down into smaller modules. The development teams work on various modules, and when implementing a particular module, the developers can make mistakes. A tester tests the code after all of the modules have been introduced. After integrating all of the modules, the tester can skip a few test cases and be unable to conduct regression testing. And if a bug was discovered, it’s difficult to detect which module has the flaws. Even if the system passes all possible test cases, it can fail to provide the necessary functionality because the system design contains flaws that are difficult to detect through testing.

Debugging is more difficult than scripting, and validation in the implementation of every device requires more time and money. We cannot rely solely on software testing for software reliability, especially when it comes to safety-critical systems, because even a single system vulnerability can result in complete system failure.

Figure 1 shows an IoT network scenario where the IoT devices are connected via wifi. In the proposed work the M2M and central IOT authentication protocol model is formally verified using the model checking technique.

1.1. RADIUS (Remote Authentication Dial-In User Service)

RADIUS network security protocol provides centralised Authentication, Authorization, and Accounting (AAA) for remote users who wants to connect to and use an IoT [3] network resource and services. It runs on ports 1812 and 1813. There are many message forms used for sending and receiving messages between IoT client and server, such as Access-Request, Accept, Reject and Challenge. For sending the credentials of IoT users to and from the authentication server RADIUS supports several different protocols and in this paper, we examine RADIUS’ vulnerability using its RFC document [8] radius and the commonly used Free RADIUS software package. The RADIUS protocol is primarily used by a Network Access Service (NAS) system to
Figure 2. IOT devices using M2M communication.

provide AAA services to users who wish to connect to any private network, while PPP CHAP is used to authenticate its peer users before allowing Network Layer protocols to transmit through the connection. RADIUS is a protocol that communicates authentication, authorization, and configuration information between a Network Access Server and a mutual Authentication Server.

1.2. Constrained Application Protocol - CoAP Protocol.
CoAP[13] is a customised web transfer protocol for IoT devices with limited resources. CoAP was created to allow basic, resource-limited devices to connect to the IoT network, even across restricted networks with poor bandwidth and availability. Machine-to-machine (M2M) applications. Figure 2 shows IoT device to device connectivity and message exchanges.

The CoAP protocol was designed by the IETF and specified in the RFC7252[13]. The IETF Group created the CoAP protocol. Its made to work in cramped conditions. The protocol treats the many items on the network as resources, using a REST-style design[4]. Each resource is given a unique URI. To operate the various resources, the protocol employs the corresponding URI. The CoAP Protocol was created to appear and work like the Hypertext Transfer Protocol (HTTP). This gives the protocol a more typical website-based business appearance, allowing it to be compatible with an existing web service-based system. CoAP is a specific web transfer protocol designed for usage with restricted nodes and constrained networks. The purpose of CoAP’s design was to keep message overhead low, reducing the need for fragmentation. CoAP communication is comparable to HTTP client/server communication. HTTP, on the other hand, has a high level of computational difficulty, a low data throughput, and large use of energy. In terms of the REST paradigm, CoAP is comparable to HTTP, with GET, POST, PUT, and DELETE methods, URIs allocated to each resource and service, MIME types, and so on. HTTP, on the other hand, is based on the TCP protocol and uses point-to-point (p2p) communication, but CoAP is not restricted to UDP and may be implemented via various channels such as TCP, DTLS, or SMS. UDP offers IP multicast for CoAP, which meets group communication for IoT.

In an unrestricted network, HTTP is utilised, whereas, in a limited network, CoAP is used. CoAP, on the other hand, can simply connect to HTTP via proxy components, allowing HTTP clients to communicate with CoAP servers and vice versa, allowing for greater Web integration and the capacity to satisfy IoT requirements. The advantages of a REST architecture developed particularly for low- power loss wireless networks are available with CoAP. The IETF CoRE Working Group is working on a protocol that will allow tiny devices in Machine to Machine (M2M) communication to access resources in a similar way to HTTP. The Internet of Things includes such devices. Instead of coding the fields as strings, they are saved in binary format, i.e. as sparsely as feasible, in the CoAP header containing choices. The two-layered interaction model proposed by CoAP was presented: The message layer ensures transmission reliability, while the request-response layer implements the REST architecture’s functionality.
Using the DTLS encryption protocol, CoAP communications may be rendered tap-proof. The thin form of a CoAP message’s header fields provides a convincing impact. M2M is plainly evident as the major guideline in CoAP’s functioning. The true benefits and drawbacks of HTTP in comparison to alternative application protocols will only become evident with more widespread use.

2. Literature Review
The creation of secure communication between communicative agents is aided by security protocols. The majority of digital communication networks rely on them, including encrypted internet conversations, bank deposits, and any electronic form of financial transfer. Such programmes should be formally analysed for the security weaknesses in the core algorithms.

CSPs are tough to develop even with flawless cryptography. The most well-known authentication protocol Needam-Schroeder authentication protocol [11] was initially presented in 1978 and It’s been around for a long time to authenticate the communicating agents using a combined approach that includes a public key and symmetric key cryptosystems. In 1995 Galvin-Lowe[11] broke the Needam-Schroeder protocol using FDR model checker, and a new modified version of NS protocol called Needam-Schroeder-Lowe protocol was developed. This demonstrates the need for formal security protocol verification.

If design errors were introduced during the system development process of authentication security protocols, it is extremely difficult to identify and fix using testing methods. Formal Verification is the best solution for finding design errors, it also helps to prove the reliability by providing statistical proof.

The creation of secure communication between communicative agents is aided by security protocols. The majority of digital communication networks rely on them, including encrypted internet conversations, bank deposits, and any electronic form of financial transfer. Such programs should be formally analysed for the security weaknesses in the core algorithms.

Figure 3 shows the Formal Verification techniques. For all these three approaches the state-space explosion problem is a common obstacle and the Temporal Logics helps to prevent the issue of state explosion. Current research outcomes describe the current state of the art for using the spin model checker to formally verify the NSPK[11] and Denning Sacco DS protocol [14]. The recent advances in the research describe how a model checker can be used to formally validate the protocol’s security properties.
To formally verify a security protocol there are several tools available that work based on the model checking technique such as Casper, Proverif, Cryptoverif, Span-Avispa, and Scyther model checker\[10\] etc. Out of these tools, the Scyther tool has the strongest adversary model and it has got better GUI and attack graphs which is very helpful in analysing the security protocol.

The DY attacker model \[12\] is an active attacker who has got full control over the network through which the communication agents send and receive data, on which the agents interact and the adversary has complete control over the channel. This allows the attacker to listen in on any communication messages, execute reply attacks, and get information about the agents’ cryptographic primitives, he can delete the sent messages and can analyse their data contents, he can insert his own messages or data, or he can reroute the data traffic or simply he can retransmit messages that he has received and he can observe the traffic.

Figure.4 shows that parallel to the security protocol model, the Dolev-Yao adversary model will run. Solving the derivability issue is an important part of determining which words the intruder has access to. The DY-channel aids in channel modeling and provides a good adversary for CSP verification.

3. Proposed Methodology

Formal verification is a difficult process that necessitates a great deal of effort and experience required while developing a system, with help of automated tools verification of even a complex security system is made very easy. Figure.5 shows the architecture of verifying the IoT Authentication protocols. The IoT authentication security protocols RADIUS and CoAP PPP protocol specifications were obtained from their respective RFPs, using those specifications an abstract protocol model is developed, and the security properties specified as claims were verified using the scyther model checker.

The model checker creates a finite automaton for the protocol specified and a state-space which is represented in a tree structure that contains all the possible behaviors of the security protocol. The security property specified as a claim event is verified by the model checker by searching through the state space starting from the initial state to the final state. If there is a single path that violates the claimed security property then a corresponding attack is generated via an attack graph.

4. Security Properties specified as claim events

The security properties were specified as a part of the protocol specification itself using the temporal logic and the claim events are local in nature.

Let $\alpha$ be the claim event of a role $A$. The security properties defined requires that some imply $B$ on the trace of $(A)$. Claim Run Ev is true for every instance of $\alpha$ for all traces of role $A$ . the property is true if
The secrecy property is one of the fundamental security properties that any authentication security protocol must satisfy, and we have validated the secrecy property in the suggested. The secrecy property is defined as - A protocol X with a role M with claim protocol event $\alpha = \text{claim}(M, \text{secret}, Y)$ satisfies secrecy of Y, notation $\text{SECRET}(X, \alpha)$, if and only if

$$\forall t \in \text{traces}(A) \forall (\text{inst}, \alpha) \in t : B(t, (\text{inst}, \alpha))$$

4.1. **RADIUS IOT Authentication Protocol verification model**

The Formal specifications of the RADIUS protocol are obtained from the rfc3579 [8] which provides the complete specification of the RADIUS protocol that how the protocol must be implemented. The below Algorithm1 shows the abstract RADIUS Authentication Protocol model. Figure.6 shows that the RADIUS model contains 3 communication roles and these roles are modeled as, namely 1. User, 2. Radius-IoT-Client and 3. Radius-Server.

The main goal of this protocol is to successfully authenticate the IoT user with the central RADIUS server via Radius-IoT-Client. The user has to authenticate with the RADIUS server which is a gateway of the network. In RADIUS NAS is a connecting point between the remote user and RADIUS server (AAA server). NAS can be a remote access server for remote users. RADIUS not only supports remote users but also supports local users through wired and wireless connections. A RADIUS client can be a Wireless Acess Point (WAP) such as a wifi-hot spot which is a middle man between a RADIUS user and a AAA RADIUS server.

The RADIUS Authentication Protocol has mainly 8 steps. Step 1 the User role sends the User Name, Password, User identity, User-Port-id to the Radius-IOT-Client by encrypting with the Radius-IoT-Client’s public key. Step 2, the Radius-IoT-Client adds NASID and encrypts the message with a pre-shared key between the Radius-IoT-Client and Radius-server k(Radius-IoT-Client, Radius-Server). Step 3: The Radius-Server sends the Session-id and a nonce value from the server to the Radius-IoT-Client, encrypted with a pre-shared session key k. (Radius-IoT-Client, Radius-Server). Step 4 The Radius-IoT-Client forwards the server nonce value to the User role by encrypting it with the public key of user pk(User). Step 5 the User sends the Radius-Server with the User nonce Nusr and server nonce number
encrypted with the public key of the Radius-Server. Step 6 and 7 represents the challenge and response mechanism. In Step8. Radius-Server sends the User with the Authentication-Success message encrypted with the public key of the user $pk(\text{User})$

4.2. CoAP protocol model description

Algorithm 2 shows the CoAp protocol security model, the CoAp message authentication protocol has 2 communicating agents namely- Client and Server. CoAp protocol has mainly 4 steps. In step1 the Client role sends its ID, the POST method which is a REST function used to send the data to the server, the URI Uniform Resource Identifier path, the Authentication details Auth, payload, and finally a nonce value $N_{\text{client}}$ encrypted using the public key of server $pk(\text{server})$. In step2 the server replies back a token to the client by sending the server identity and a Token by encrypting Server, POST, URI-path, AUTH, Payload, $N_{\text{client}}$, Token using the public key of the client. In step3 the client sends the payload by including the Token which was obtained in the previous step by encrypting with the public key of the server. The Server finally sends back an acknowledgment packet to the client.
Algorithm 1 RADIUS IOT Protocol Description

Communicating Roles: USER, Radius-IOT-Client, Radius-Server

# User role data
UPSWD - User Password of type String.
UNAME - User Name of type String.
User-Port-id - User Port Id of type String.
Nusr - User nonce of type Nonce.

# Radius-IOT-Client data
NAS-ID - Network Authentication Server ID of type Nonce

# Radius-Server data
Server-nonce-pseudo-random - Pseudo random number generated by radius server of type Nonce.
Session-id - Session-Id generated by Radius server of type Nonce

User,Radius-IOT-Client,Radius-IOT-Client,Radius-Server : Communicating Roles

k(x,y) - Shared session key between roles x and y.

Step1. User → Radius-IOT-Client:
{UNAME,UPSWD,User,User-Port-id}pk(Radius-IOT-Client)

Step2. Radius-IOT-Client → Radius-Server:
{UNAME,UPSWD,User,User-Port-id,NAS-ID}k(Radius-IOT-Client,Radius-Server)

Step3. Radius-Server → Radius-IOT-Client:
{Session-id,Server-nonce-pseudo-random}k(Radius-IOT-Client,Radius-Server)

Step4. Radius-IOT-Client → User:
{Server-nonce-pseudo-random}pk(User)

Step5. User → Radius-Server:
{Nusr,Server-nonce-pseudo-random}pk(Radius-Server)

Step6. Radius-Server → User:
{Radius-Server,Nusr,NRadius-Server}pk(Radius-Server)

Step7. User → Radius-Server:
{NRadius-Server,Radius-Server}pk(Radius-Server)

Step8. Radius-Server → User:
{Authentication-Success}pk(User)

# Security Claims by User role
claim_C1(Radius-IOT-Client,Secret,Nusr);

# Security Claims by Radius-IOT-Client role
claim_C1(USER,Radius-IOT-Client,Running,Radius-IOT-Client,UNAME,UPSWD);
claim_C2(USER,Secret,Session-id);

# Security Claims by Radius-Server role
claim_C1(Radius-IOT-Client,Secret,NRadius-Server);
claim_C2(Radius-IOT-Client,Secret,Session-id);
claim_C3(Radius-IOT-Client,Secret,Nusr);
claim_C4(Radius-IOT-Client,Secret,Authentication-Success);

# Intruder knowledge
Intruder = EVE
IntruderKnowledge = \{User,Radius-IOT-Client,Radius-Server,EVE,pk(User),pk(Radius-IOT-Client),pk(Radius-Server),pk(EVE),sk(EVE)\}
Algorithm 2 CoAp IOT message authentication protocol model.

Communicating Roles: Client, Server

#Data from Client
fresh Nclient: Nonce;
fresh POST: REST-method;
fresh URI-path: String;
fresh AUTH: String;
fresh Paload: String;
fresh Payload: String;
fresh URI-path2: String;

#Data from Server
fresh Nserver: Nonce;
fresh Token: TOKEN;
fresh Acknowledgement: REST-method;
Version-no - Version number of type integer
NAS-port-no - Network Authentication Server port number of type int
pk(Server) - Public key of server.
pk(Client) - Public key of client.

Step 1. Client → Server: {Client, POST, URI-path, AUTH, Payload, Nclient} \( pk(\text{Server}) \)

Step 2. Server → Client: {Server, POST, URI-path, AUTH, Payload, Nclient, Token} \( pk(\text{Client}) \)

Step 3. Client → Server: {Client, POST, URI-path2, AUTH, Payload, Token} \( pk(\text{Server}) \)

Step 4. Server → Client: {Server, POST, URI-path2, AUTH, Payload, Nclient, Token, Acknowledgement} \( pk(\text{Client}) \)

#Security Claims events by Client role
claim \( K1(\text{Client}, \text{Secret}, \text{POST}) \);
claim \( K2(\text{Client}, \text{Secret}, \text{URI-path}) \);
claim \( K3(\text{Client}, \text{Secret}, \text{AUTH}) \);
claim \( K4(\text{Client}, \text{Secret}, \text{Payload}) \);
claim \( K5(\text{Client}, \text{Secret}, \text{Nclient}) \);
claim \( K6(\text{Client}, \text{Nisynch}) \);
claim \( K7(\text{Client}, \text{Secret}, \text{URI-path2}) \);
claim \( K8(\text{Client}, \text{Niagree}) \);
claim \( K9(\text{Client}, \text{Alive}) \);

#Security Claim events by Server role
claim \( S1(\text{Server}, \text{Secret}, \text{Token}) \);
claim \( S2(\text{Server}, \text{Secret}, \text{Nserver}) \);
claim \( S3(\text{Server}, \text{Secret}, \text{Acknowledgement}) \);
claim \( S4(\text{Server}, \text{Nisynch}) \);
claim \( S5(\text{Server}, \text{Niagree}) \);
claim \( S6(\text{Server}, \text{Alive}) \);

#Intruder Information
Intruder = EVE
IntruderKnowledge = Client, Server, pk(Client), pk(Server), pk(EVE), sk(EVE)
5. Verification Results

5.1. RADIUS protocol Verification Results
In the proposed work, the model of the RADIUS protocol for IOT was built from its RFC. Figure 7 shows the proposed crypto for the RADIUS protocol is safe and free from attacks. The low-powered IoT device gets authenticated with the central RADIUS server (cloud) with the help of newly generated nonce values. Also, the proposed model satisfies the confidentiality property which is a primary security property requirement of RADIUS protocol to provide AAA services. Upon successful authentication, with the central RADIUS server, the IoT device was granted access to use the network resources. Further, the Authorisation and Accountability are handled based on the privileges assigned during the authentication process.

5.2. CoAP protocol Verification Results
In the proposed work, the CoAp message authentication protocol abstract model was built using its RFC. Figure 8 shows the verification result of the CoAp message authentication protocol in which the proposed crypto for the CoAp protocol is safe, secure, and free from crypto attacks for the corresponding security property claims. The IoT client successfully authenticates its secure messages with the server using the token obtained from the server. From the verification results, it is clear that the CoAp protocol satisfies the secrecy, non-injective agreement, non-injective synchronization, and aliveness security properties for the secure message authentication.

6. Conclusion
By giving mathematical proofs for IoT authentication protocol’s functional correctness, the formal verification technique can improve the reliability of security protocols used in IoT networks. One of the most difficult aspects of formal verification of IoT Authentication protocols is creating a model for CSPs and defining security properties. The Scyther model checker was used to formally verify the RADIUS and CoAP authentication protocols, and the protocols were found to be attack-free. The RADIUS protocol is best suited for the centralised authentication of IoT devices and the CoAP protocol can be used for M2M authentication securely.
7. Future Work
The proposed RADIUS and CoAP protocol models must be further needs to be optimised by reducing the number of states in the state space. Most advanced Authentication protocols such as TACACS[16], TACACS+[17], Kerberos[15], and other advanced IoT network authentication protocols must be formally verified using model checking techniques.

References
[1] Stefinko, Y, Piskozub, A and Banakh, R 2016 Manual and automated penetration testing Benefits and drawbacks Modern tendency In 2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET) pp 488-491
[2] Avalle, M, Pironti, A and Sisto, R 2014 Formal verification of security protocol implementations: a survey Formal Aspects of Computing. 26 99-123
[3] Lee, I and Lee, K 2015 The Internet of Things (IoT): Applications, investments, and challenges for enterprises Business Horizons. 58 431-440
[4] Rodríguez, C, Baez, M, Daniel, F, Casati, F, Trabucco, J.C, Canali, L and Percannella, G 2016 June REST APIs: a large-scale analysis of compliance with principles and best practices In International conference on web engineering Springer Cham pp 21-39.
[5] Kuhn, D.R and Okun, V 2006 April Pseudo-exhaustive testing for software In 2006 30th Annual IEEE/NASA Software Engineering Workshop pp 153-158
[6] Fraser, M.D, Kumar, K and Vaishnavi, V.K 1991 Informal and formal requirements specification languages: bridging the gap IEEE transactions on Software Engineering 17(5) p 454
[7] Kemmerer, R.A 1985 Testing formal specifications to detect design errors IEEE transactions on software engineering. 1 32-43
[8] Aboba, B and Calhoun, P 2003 RFC3579: RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP).
[9] Bormann, C, Castellani, A.P and Shelby, Z 2012 Coap: An application protocol for billions of tiny internet nodes IEEE Internet Computing. 16 62-67
[10] Cremers, C.J 2008 July The Scyther Tool: Verification, falsification, and analysis of security protocols In International conference on computer aided verification Springer Berlin Heidelberg pp 414-418
[11] Lowe, G 1995 An Attack on the Needham Schroeder Public Key Authentication Protocol Information processing letters. 56
[12] Rocchetto, M and Tippenhauer, N.O 2016 November CPDY: extending the Dolev-Yao attacker with physical-layer interactions. In International Conference on Formal Engineering Methods Springer Cham pp 175-192
[13] Shelby, Z, Hartke, K and Bormann, C 2014 The constrained application protocol (CoAP)
[14] Steiner, M, Tsudik, G and Waidner, M 1995 Refinement and extension of encrypted key exchange ACM SIGOPS Operating Systems Review 29(3) pp 22-30
[15] Neuman, B.C and Ts’o, T 1994 Kerberos: An authentication service for computer networks IEEE Communications magazine. 32 33-38
[16] Finseth, C 1993 An access control protocol sometimes called TACACS RFC 1492 July.
[17] Carrel, D 1997 TACACS+ Protocol Specification