A Novel DevSecOps Model for Robust Security in an MQTT Internet of Things

Manasa Ekoramaradhya and Christina Thorpe
Technological University Dublin, Blanchardstown, Dublin, Ireland
manasa151685@gmail.com
christina.thorpe@tudublin.ie

Abstract: Message Queuing Telemetry Transport (MQTT) is a standard messaging protocol for the Internet of Things (IoT). Among the various communication protocols used in IoT, MQTT stands unique because of its multiple advantages such as being efficient and light weight, reliability in message delivery and scalability to millions of things. However, the fact that the data privacy of the MQTT messages can be compromised while the data is in transit poses risks to the security mechanism. Attack scenarios related to MQTT have exposed multiple risks and vulnerabilities such as thousands of MQTT brokers being accessible over the default port, data privacy, authentication, data integrity, port obscurity, and botnet over MQTT. These risks and vulnerabilities undermine security mechanism which results in compromised IoT systems. Development Security and Operations (DevSecOps) aims at integrating security at every phase of the IoT lifecycle with enhanced automation, tools, and a process for determining security vulnerabilities at every stage. This results in a rapid and cost-effective IoT system which is enabled by proactive security mechanisms, threat prediction, threat detection, and alerting mechanisms. The aim of this work is to build a DevSecOps pipeline utilizing open source MQTT servers and brokers. A comparative study was performed to identify the risk posture provided by the DevSecOps pipeline across MQTT ports offering different combinations of security mechanisms. Firstly, threat modelling was conducted wherein the IoT system was analyzed at an architectural level from an attacker's perspective and appropriate risk mitigation and defense mechanisms were accommodated into the design. The IoT system was then subjected to rigorous static and dynamic analysis followed by vulnerability scanning and third component checks. Penetration test cases and controls are automated to check threats and vulnerabilities like escalation of privileges, denial of service, spoofing, information disclosure, and repudiation. An alerting mechanism is also integrated into the system to monitor risks and vulnerabilities. Our proposed DevSecOps models achieves standard maturity in security systems with earlier threat prediction and detection.

Keywords: Message Queuing Telemetry Transport (MQTT), Internet of Things (IoT), Development Security and Operations (DevSecOps), Cybersecurity

1. Introduction

An increasing number of devices connected to the Internet of Things (IoT) is contributing to increased cyber threats and cybercrimes across the world. For example, we now commonly see connected appliances, smart home security systems, autonomous farming equipment, wearable health monitors, smart manufacturing equipment, etc. These connected devices pose a threat to cyber security for several reasons (Hassija 2019) (Meneghello 2019): firstly, there is an abundance of data transmitted and stored by these IoT systems; secondly, the IoT systems are hyper connected to both virtual and physical environments; thirdly, there is massive complexity in IoT environments due to the volume and diversity of devices and connections; and finally, many of these devices have been manufactured with security as an afterthought. It is reported that cybercrime alone cost $1 Trillion USD in 2020 (McAfee 2020) and is estimated to reach over 10 Trillion annually by 2025, driving the need for cybersecurity solutions or systems for IoT. This is especially important given the critical role IoT will play in applications such Industry 4.0 and Intelligent Transport (Okano 2017).

Examining IoT systems at an architectural level reveals that it uses many protocols such as CoAP, MQTT, XMPP, RESTFUL Services, 6LoWPAN and many more (Sharma 2018), however, the MQTT protocol has gained widespread use for machine-to-machine communication in bandwidth-constrained environments where compact data transport is a strict requirement. The multiple advantages of MQTT protocol makes it a very attractive option for IoT devices. These include minimal resource requirements, lightweight, efficiency, message persistence and the ability to support bi-directional communications (Mishra 2020).

Despite their popularity, MQTT and IoT devices came under wide scrutiny when Palo Alto Networks reported that 98% of all IoT device traffic exposed private and confidential information on to the network as it was unencrypted; and 57% of these devices were susceptible to medium or high severity attacks which attracted the hackers (Palo Alto 2020). Extensive research on MQTT security has proved that multiple brokers were discovered by Shodan (Harsha 2018), which we also confirmed. Since MQTT by default does not involve any encryption standards, it poses a high risk in terms of data privacy. MQTT authentication can be compromised by extracting

X.509 certificates or by sniffing techniques. Posts to an MQTT node authenticates at a broker, there is no specific process to determine which publishers or subscribers can publish or receive messages, posing serious authorization issues (Syaiful 2017) (Palo Alto 2020). When penetration testing was done on open MQTT systems, multiple risks and vulnerabilities were revealed related to elevation of privileges, denial of service and many more.

In the past decade, traditional development cycles have become less common with many moving towards a more lean and agile Development and Operations (DevOps) cycle. DevOps means end-to-end automation in software development and delivery (Ebert 2016) (Zhu 2016). While this new paradigm presents many benefits in the form of agility and efficiency, it does pose a risk for security as many of the robust security processes are no longer integrated as the deployment is placed in the control of the developers (Fox 2020) (Perera 2016). Absence of an efficient threat detection and alerting mechanism while applying a DevOps approach in MQTT application development leads to the risks with high severities skipping the focus of the development team.

Development, Security and Operations (DevSecOps) introduces enhanced automation and aims at security integration at every stage of software development lifecycle (Kenyon 2021). Implementation of this method in IoT systems helps in identifying security risks from the initial design stage until the live stage. It also integrates threat detection and alerting mechanisms for advanced monitoring. This includes security integration starting from static analysis and source code reviews, vulnerability scanning, scanning for components with known vulnerabilities and third-party components. Dynamic analysis and extensive penetration testing are done to reveal any vulnerabilities that might be potentially exploited. However, while there has been work on developing a DevSecOps (Myrbakken 2017) models for Cloud applications (Rajasekharaiah 2021) (Kumar 2021) and IoT systems (Bahaa 2021) (Wolf 2021) there is no work in the literature on DevSecOps for MQTT specifically.

Using a DevSecOps model, the entire architecture of the IoT system which uses the MQTT protocol will undergo extensive threat modelling to determine the possible risks in the system. The overall risk posture will be determined as a result of all these DevSecOps activities and the integrated alerting mechanisms will send notifications of these results. In this paper, we will compare the results of DevSecOps methodology which was implemented across three different port services offered by the opensource MQTT broker, Mosquitto. Our results show that our model achieves standard maturity in security systems with earlier threat prediction and detection.

The remainder of the paper is organised as follows: Section 2 discusses the technical background information and the related work pertinent to the research presented in this paper. Section 3 defines the methodology defined and implemented in this research. Section 4 presents the results of the experimentation conducted. Finally, Section 5 concludes this paper and presents items for future work.

2. Background and Related Work

Though MQTT is widely used across most of the IoT systems, the security vulnerabilities and the risks associated with this protocol poses severe challenges. DevSecOps deals with proactively identifying and monitoring these vulnerabilities. The below sections detail out the working and security challenges associated with MQTT and the DevSecOps methodology.

2.1 MQTT

MQTT is an OASIS standard protocol and hence the specifications is provided by OASIS MQTT Technical Committee[1]. ***MQTT v3.1.1*** contained a number of threats that had to be considered, such as : devices were vulnerable to compromise; data at rest in clients and servers were potentially accessible; protocol behaviors could have side effects (e.g. “timing attacks”); Denial of Service (DoS) attacks; communications could be intercepted, altered, re-routed or disclosed; and spoofed Control Packets could be injected. The implementations had to consider solutions such as: authentication of users and devices; authorization of access to server resources; integrity of MQTT Control Packets and application data contained therein; privacy of MQTT Control Packets and application data contained therein. ***MQTT v5.0*** has enhanced security features - both MQTT client and server must provide: authentication, authorization, and secure communications options.

[1] http://docs.oasis-open.org/mqtt/mqtt/

MQTT is a TCP/IP based protocol which requires an established TCP connection before communication. A three-phase handshake is employed; in first phase, the TCP connection is established; in the second phase, MQTT connections and data transfer takes place; the third phase terminates the established TCP connection. The reliability of the MQTT messages depends on the QoS (Quality of Service) levels used. The selection depends on the type and importance of the MQTT messages/topics. In level 0, the transmission is unreliable, the message is delivered without duplication and no acknowledgement is required. In level 1 (default), the message is delivered at least once without duplication and an acknowledgement is required. In level 2, the message is delivered without duplication.

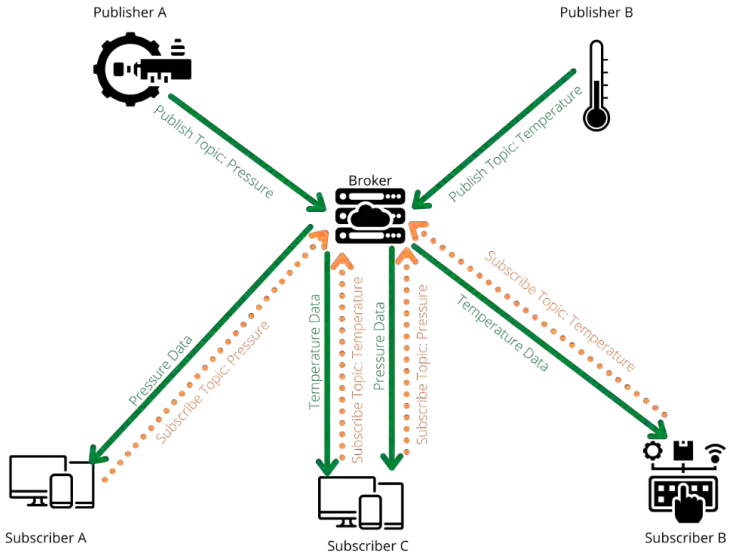

Figure 1: Typical MQTT publish-subscribe scenario

2.2 Typical MQTT publish-subscribe model

MQTT is an application layer protocol which relies on TCP and works on publisher-subscriber model (Mishra 2020). Figure 1 illustrates a typical MQTT communication wherein the centralized broker intermediates the communications between the publisher and the subscriber nodes. *Publisher A* sends the data related to the topic *'Pressure'* and the *Subscriber A* receives this data by subscribing to this topic via the *Broker*. A subscriber can subscribe to multiple topics (as illustrated by *Subscriber C*) and any node in the network can act as publishers and subscribers. The connection between the broker and the client nodes will always stay open and the transmitted packet size is relatively small. These factors speed up the MQTT communications when compared to other protocols. MQTT protocol consumes low power and is efficient for communicating shorter messages, which supports high scalability.

2.3 Security risks and vulnerabilities in MQTT

Though MQTT is a highly resourceful protocol it poses several risks in terms of security. Previous research work has shown that thousands of MQTT brokers were openly available on port 1883 and were indexed by Shodan (Syaiful 2017). This research also highlighted that the MQTT brokers with client connection code 0 did not have any client authentication mechanism which enable anonymous clients to either publish a message or subscribe to any topic on that network. Since MQTT does not provide any encryption protocol by default the privacy of the data in transit can be easily compromised by capturing the packets with the help of tools such as Wireshark or Burpsuite. After the KeepAlive time expires the publisher must resend the connect packet which contains the username and password. Capturing this packet will eventually compromise the MQTT authentication mechanism. After the capture of the packet traffic the attacker can easily modify the data which leads to the compromise in data integrity (Syaiful 2017). Furthermore, MQTT is susceptible to fuzzing and a lot of vulnerabilities were detected by fuzzing techniques (Casteur 2020). MQTT has shown severe risks when it comes to replay attacks and Man-In-The-Middle attacks (Chen 2020). MQTT has also posed severe challenges with respect to confidentiality, access control, and Denial of Service (Hintaw 2019). Moreover, since these vulnerabilities are mostly discovered from a hacker's perspective or while they are in production, fixing the vulnerabilities becomes a challenging task. This points to the increased need of the system which will integrate

security at every phase of the IoT system development while using MQTT and drives a more proactive approach in threat hunting and alerting which can be fulfilled by a DevSecOps methodology.

2.4 Development, Security and Operations (DevSecOps)

DevSecOps brings in the idea of enhanced automation and security integration in the Software Development Lifecycle (SDLC). It aims at implementing security at each phase starting from the initial design phase till the production and maintenance phase (Kenyon 2021). This is more efficient and a cost-effective method since finding the vulnerabilities at the early stages of the software development Lifecyle are easy to be fixed than the ones found in live or production. This introduces a proactive way of finding the vulnerabilities and it also accelerates the patching of security vulnerabilities. Nearly half of all software development organizations are now relying on this model to implement a proactive security methodology. The Gartner Hype Cycle for Agile and DevOps, 2020, indicates that DevSecOps is in the early stages of mainstream adoption. Gartner quotes a 20-50% market penetration among DevSecOps' target audience. Gartner projects that DevSecOps will reach mainstream adoption within 2-5 years. Figure 2 illustrates the concept of DevSecOps and how it relates to the more well-known DevOps phases.

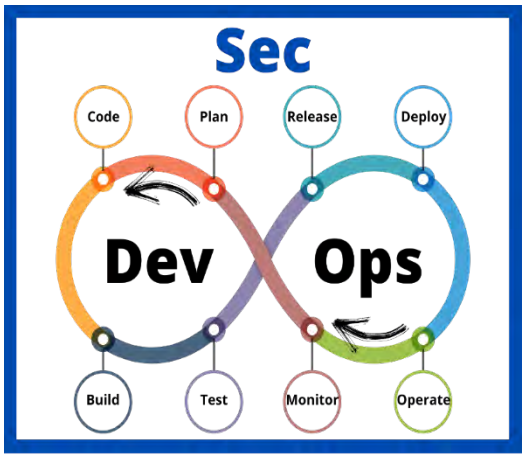

Figure 2: General DevSecOps Model

3. Methodology

Implementing DevSecOps for IoT systems and devices, especially where a protocol like MQTT is being used for communications, helps in identifying the security risks, vulnerabilities, and severities at every stage. Along with detection and identification, treating/controlling these risks can happen in a more efficient and cost-effective manner.

3.1 Implementation

Figure 3 illustrates the DevSecOps pipeline developed for an MQTT IoT system. In the following sections, we describe the DevSecOps activities that form the pipeline. Each of the three test scenarios were assessed using the sample pipeline to validate our model. The tools selected for each phase are popular tools used by security professionals.

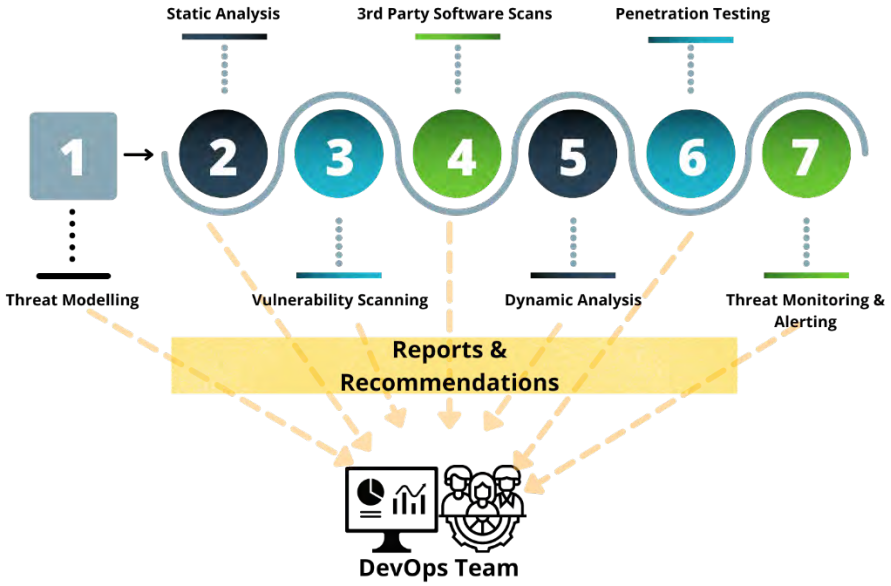

Figure 3: DevSecOps MQTT Pipeline

3.1.1 Threat Modelling

In this process, the architecture of the IoT model and the MQTT protocol were analyzed to identify and locate possible vulnerabilities, identifying probabilities and objectives for possible attacks and putting proper mitigation methods in place. This activity can be done manually, or to enhance the efficiency and accuracy, tools such as Microsoft Threat Modelling can be used. Many risks and vulnerabilities related to elevation of privileges, denial of service, information disclosure, spoofing, repudiation, device isolation was found in this stage. The Threat Modeling Tool is a core element of the Microsoft Security Development Lifecycle (SDL). It allows software architects to identify and mitigate potential security issues early when they are relatively easy and cost-effective to resolve. As a result, it greatly reduces the total cost of development. Also, the tool was designed with non-security experts in mind, making threat modeling easier for all developers by providing clear guidance on creating and analyzing threat models.

3.1.2 Static Analysis

Static Application Security Testing (SAST) is a white box testing method where the tester has access to the source code, framework, and the binary execution files. The vulnerabilities found at this phase are less expensive to fix. Tools used in this phase are CheckMarx and SonarQub. Checkmarx CxSAST is a highly accurate and flexible Static Code Analysis Tool that allows organizations to automatically scan un-compiled/un-built code and identify hundreds of security vulnerabilities in the most prevalent coding languages.

SonarQube is a Code Quality Assurance tool that collects and analyzes source code and provides reports for the code quality of your project. It combines static and dynamic analysis tools and enables quality to be measured continually over time.

3.1.3 Vulnerability Scanning

Web application vulnerabilities scanners help in scanning of the web applications from an outsider perspective to find common web application vulnerabilities like cross-site scripting, SQL injection, command injection and many more. In this phase the Nessus port scanner tool is used. Nessus is a remote security scanning tool, which scans a computer and raises an alert if it discovers any vulnerabilities that malicious hackers could use to gain access to any computer you have connected to a network. It does this by running over 1200 checks on a given computer, testing to see if any of these attacks could be used to break into the computer or otherwise harm it.

3.1.4 3rd Party Software Scans

The source code used to develop the devices or systems, or sometimes the protocols themselves, may have libraries which do not contain the latest code version, resulting in a lot of vulnerabilities. Tools such as BlackDuck and OWASP Dependency checker are used in this phase. Black Duck is a complete open-source management solution, which fully discovers all open source in your code. It can map components to known vulnerabilities

and identify license and component quality risks. You can use Black Duck to set and enforce open-source policies and integrate open-source management into your DevOps environment. Additionally, Black Duck monitors and alerts you when new threats are reported. Dependency-Check is a Software Composition Analysis (SCA) tool that attempts to detect publicly disclosed vulnerabilities contained within a project's dependencies. It does this by determining if there is a Common Platform Enumeration (CPE) identifier for a given dependency. If found, it will generate a report linking to the associated CVE entries

3.1.5 Dynamic Analysis

This is more of a black box testing, wherein the approach is like that of a hacker. Vulnerabilities here are discovered after the development phase and hence it is comparatively expensive to fix these. Tools such as OWASP Zap and HCL Appscan are run in this phase. Appscan runs automatic scans that explore and test web applications and includes one of the most powerful scanning engines in the world.

3.1.6 Penetration Testing

Penetration tests are cyber-attacks which are simulated against the system to identify the vulnerabilities which could potentially be exploited by malicious actors. It describes the intentional launching of simulated cyberattacks by "white hat" penetration testers using strategies and tools designed to access or exploit computer systems, networks, websites, and applications. Although the main objective of pen testing is to identify exploitable issues so that effective security controls can be implemented, security professionals can also use penetration testing techniques, along with specialized testing tools, to test the robustness of an organization's security policies, its regulatory compliance, its employees' security awareness, and the organization's ability to identify and respond to security issues and incidents such as unauthorized access, as they occur.

3.1.7 Threat Monitoring and Alerting

This is one of the important phases in DevSecOps. Several mature tools like Splunk can be integrated into the pipeline which will record the vulnerabilities, risks and sometimes even analyses the packets to find threats and block the source. When the behavior deviates, or a certain type of risk is found the alerting feature will automatically notify the user about the risks and vulnerabilities. It also keeps a count on the number of occurrences, source of the threat and uses machine learning algorithms to even predict the threats early.

3.2 Test environment

For this work, we used an IoT system with MQTTv5.0, configured to use QoS level 1 and other default parameters, as its base communication protocol for our DevSecOps implementation. Here, open-source servers and brokers from Mosquitto.org are used to set publisher and subscriber nodes and MQTT service provided across three different ports including:

- *Test Scenario 1 (Low Security):* port 1883 (MQTT, unencrypted, unauthenticated)
- *Test Scenario 2 (Medium Security):* port 8080 (MQTT, WebSockets, unencrypted, unauthenticated)
- *Test Scenario 3 (High Security):* port 8091 (MQTT, WebSockets, encrypted, authenticated).

Ports were selected to validate our DevSecOps model over extremes, ranging from totally unprotected with unencrypted services and no authentication to authenticated and encrypted services.

4. Results

In the proposed DevSecOps model (Figure 3) the risks and vulnerabilities are found in seven stages. The reports of the scans will be sent to the user at every stage for analysis and the tools used here will also propose certain mitigation methods which can be incorporated to fix those security defects at the same stage. Thus, it aims at achieving security shift left. The vulnerabilities found in these activities are also tested for false positives and the defects are logged. The standardized alerting mechanism will send alerts and reports to the user after the security testing at each stage is completed and the dashboard will display the risk summary. The tables in this section represent the vulnerability count and the risk posture across all three ports. Results show that as the ports provide an increased level of security services there is a sharp decrease in the risk posture.

Likelihood	Level of Impact				
	Critical	High	Medium	Low	Informational
Critical	Critical	High	Medium	Low	Informational
High	Critical	High	Medium	Low	Informational
Medium	High	Medium	Medium	Low	Informational
Low	Medium	Low	Low	Low	Informational
Informational	Low	Low	Informational	Informational	Informational

Figure 4: NIST 800-30 Risk Matrix

The risk posture is calculated using NIST 800-30 revision1 standard which defines risk as: ***the likelihood of an attacker exploiting that vulnerability and the impact caused by it in case it is exploited by a hacker*** (see matrix in Figure 4). Each tool outputs the total number of security issues detected, in the form of the number of issues per risk category (e.g., x low, y medium, z high). The output from the tools contributes to the security risk rating by taking the number of issues detected in the highest risk level reported. When nothing is reporting from a tool, the low category is incremented by 1. The aggregate risk is determined by summing each category in the contribution security risk rating to identify which category has the highest number of reported issues.

Table1: Security risk posture of the product when MQTT on 1883 is used.

Security Posture			
DevSecOps Activity	**Security Issues**	**Contribution**	**Aggregate Security Risk**
Threat Modelling	5 High, 3 Medium,4 Low	5 High	**HIGH**
Static Analysis	76 High, 1,477 Medium, 1,199 Low	76 High	
Vulnerability Scan	0 High, 3 Medium, 0 Low	3 Medium	
3rd Party	1 High, 0 Medium, 0 Low	1 High	
Dynamic Analysis	0 High, 2 Medium, 10 Low	2 Medium	
Penetration Testing	2 Critical, 0 High, 0 Medium, 0 Low	2 Critical	

Table 1 depicts the security posture after the DevSecOps process which was run on test.mosquitto.org -p 1883 (MQTT, unencrypted, unauthenticated). The overall risk posture is High due to issues identified during SDL activities. There are currently 83 High, 1486 Medium and 1214 Low open security issues which are present in the production. This is an expected result given that the configuration of the MQTT service had very low security in Test Scenario 1.

Table2: Security risk posture of the product when MQTT on 8080 is used

Security Posture			
DevSecOps Activity	**Security Issues**	**Contribution**	**Aggregate Security Risk**
Threat Modelling	Medium: 2	Medium: 2	**Medium**
Static Analysis	--	--	
Vulnerability Scan	High: 3, Medium: 1	High: 3	
3rd Party	High: 1, Medium: 2, Low: 1	High: 1	
Dynamic Analysis	Medium: 2	Medium: 2	
Penetration Testing	--	--	

Table 2 depicts the security posture after the DevSecOps process which was run on test.mosquitto.org -p 8080 (WebSockets, unencrypted, unauthenticated). The overall risk posture is Medium due to issues identified during SDL activities. There are currently 4 High, 7 Medium and 1 Low open security issues which are present in the production. This is an expected result given the configuration used in Test Scenario 2.

Table3: Security risk posture of the product when MQTT on 8091 is used

Security Posture			
DevSecOps Activity	**Security Issues**	**Contribution**	**Aggregate Security Risk**
Threat Modelling	--	--	**Low**
Static Analysis	--	--	
Vulnerability Scan	--	--	
3rd Party	Medium 1, Low: 3	Medium 1	
Dynamic Analysis	--	--	
Penetration Testing	0 High, 2 Medium, 0 Low	2 Medium	

Table 3 depicts the security posture after the DevSecOps process which was run on test.mosquitto.org -p 8091 (MQTT over WebSockets, encrypted, authenticated). The overall risk posture is Low due to issues identified

during SDL activities. There are currently 3 Medium and 3 Low open security issues which are present in the production, with 4 of the 6 phases not identifying any security issues. This is expected due to the higher security of the configuration used in Test Scenario 3.

The results show that all three test scenarios defined and assessed produced expected results given the specific security levels of each configuration. This demonstrated that our proposed MQTT DevSecOps model is valid and can be used to detect security issues at each stage of the development lifecycle, which can then be remediated before deployment.

5. Conclusion

With the increasing IoT devices and weak IoT communication protocols, cybercrime is costing more than ever. Among the several popular protocols for IoT communication MQTT stands in the top for several reasons such as its power to be scalable across millions of devices, lightweight, its efficiency, the reliability it offers and the speed of communication being extremely fast. However, the major setbacks include data privacy issues when the data is in transit, authentication, authorization and being controlled by botnet. This potentially exposes huge customer data in the open internet and attracts hackers due to the risks and vulnerabilities it includes. Fixing any such security defect which is in the open, especially in IoT products become highly expensive and cause huge loss if exploited. Therefore, a strong system is required wherein vulnerabilities can be found in the initial stages of the product cycle itself.

DevSecOps is one such methodology which involves security integration with enhanced automation and tooling at every stage of the product development lifecycle. It aims at finding vulnerabilities from the initial stages itself and helps in rigorous patching. There are multiple stages involved in this method such as threat modelling, secure code review SAST, DAST and penetration testing along with industry standard tools and frameworks which helps in achieving the job. These pipelines can be run along with the development pipelines on the need basis to spot the risks. It involves early threat prediction, detection, and alerting mechanism. Thus, it helps in achieving standard maturity for the IoT devices. e proposed a new model for DevSecOps in MQTT IoT systems and validated this on three different configurations using the open-source Mosquitto broker. Our results show that our model provides an effective pipeline for risk identification in these systems. Security issues were identified during SDL activities for test.mosquitto.org -p 1883, -p 8080 and -p 8091. SDL helps developers build more secure software and address security compliance requirements while reducing development cost. The product underwent various phases of the SDL and security issues were identified. The overall risk posture is High on ports 1883, Medium on 8080, and Low on 8091.

Future work could include a more performance aware pipeline development. That is, the introduction of any security tool or assessment incurs a delay overhead, which could breach the requirements of certain application releases. We propose to create a more sophisticated selection algorithm to create the pipeline by taking as input, the risk tolerance, release constraints, overhead of tools, etc. The ultimate goal is to create a tailored pipeline that respects the release constraints and minimizes the security/agility trade-off in DevSecOps.

References

Bahaa, A., Abdelaziz, A., Sayed, A., Elfangary, L. and Fahmy, H., (2021). Monitoring Real Time Security Attacks for IoT Systems Using DevSecOps: A Systematic Literature Review. *Information*, *12*(4), p.154.

Casteur, G., Aubaret, A., Blondeau, B., Clouet, V., Quemat, A., Pical, V. and Zitouni, R., (2020), June. Fuzzing attacks for vulnerability discovery within MQTT protocol. In 2020 International Wireless Communications and Mobile Computing (IWCMC) (pp. 420-425). IEEE.

Chen, F., Huo, Y., Zhu, J. and Fan, D., (2020), November. A Review on the Study on MQTT Security Challenge. In 2020 IEEE International Conference on Smart Cloud (SmartCloud) (pp. 128-133). IEEE.

Ebert, C., Gallardo, G., Hernantes, J. and Serrano, N., (2016). DevOps. Ieee Software, 33(3), pp.94-100.

Fox, M.R., (2020). IT Governance in a DevOps World. IT Professional, 22(5), pp.54-61.

Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P. and Sikdar, B., (2019). A survey on IoT security: application areas, security threats, and solution architectures. IEEE Access, 7, pp.82721-82743.

Harsha, M.S., Bhavani, B.M. and Kundhavai, K.R., (2018), September. Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and ACLs. In 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI) (pp. 2244-2250). IEEE.

Hintaw, A.J., Manickam, S., Karuppayah, S. and Aboalmaaly, M.F., (2019). A Brief Review on MQTT's Security Issues within the Internet of Things (IoT). J. Commun., 14(6), pp.463-469.

Kenyon, H.S., (2021). DevSecOps Puts Security at the Heart of Program Development. Defense AR Journal, 28(2), pp.241-241.

Kumar, R. and Goyal, R. (2021). When Security Meets Velocity: Modeling Continuous Security for Cloud Applications using DevSecOps. In Innovative Data Communication Technologies and Application (pp. 415-432). Springer, Singapore.

Meneghello, F., Calore, M., Zucchetto, D., Polese, M. and Zanella, A., (2019). IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices. IEEE Internet of Things Journal, 6(5), pp.8182-8201.

McAfee (2020). https://www.mcafee.com/enterprise/en-us/assets/reports/rp-hidden-costs-of-cybercrime.pdf

Mishra, B. and Kertesz, A., (2020). The use of MQTT in M2M and IoT systems: A survey. IEEE Access, 8, pp.201071-201086.

Myrbakken, H. and Colomo-Palacios, R., (2017), October. DevSecOps: a multivocal literature review. In International Conference on Software Process Improvement and Capability Determination (pp. 17-29). Springer, Cham.

Okano, M.T., (2017), September. IoT and industry 4.0: the industrial new revolution. In International Conference on Management and Information Systems (Vol. 25, p. 26).

Palo Alto (2020). https://unit42.paloaltonetworks.com/iot-threat-report-2020/

Perera, P., Bandara, M. and Perera, I., (2016), September. Evaluating the impact of DevOps practice in Sri Lankan software development organizations. In 2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer) (pp. 281-287). IEEE.

Rajasekharaiah, C., (2021). Securing Microservices on Cloud. In Cloud-Based Microservices (pp. 179-202). Apress, Berkeley, CA.

Sharma, C. and Gondhi, N.K., (2018), February. Communication protocol stack for constrained IoT systems. In 2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU) (pp. 1-6). IEEE.

Syaiful, S., Rahardjo, B., and Hanindhito B. (2017) "Attack scenarios and security analysis of MQTT communication protocol in IoT system." In 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), pp. 1-6. IEEE, 2017.

Wolf, A., Simopoulos, D., D'Avino, L. and Schwaiger, P., (2021). The PASTA threat model implementation in the IoT development life cycle. INFORMATIK 2020.

Zhu, L., Bass, L. and Champlin-Scharff, G., (2016). DevOps and its practices. IEEE Software, 33(3), pp.32-34.