Security Assurance in the SDLC for the Internet of Things

Do you have something to say about this article?
Visit the *Journal* pages of the ISACA® website *(www.isaca.org/journal)*, find the article and click on the Comments link to share your thoughts.

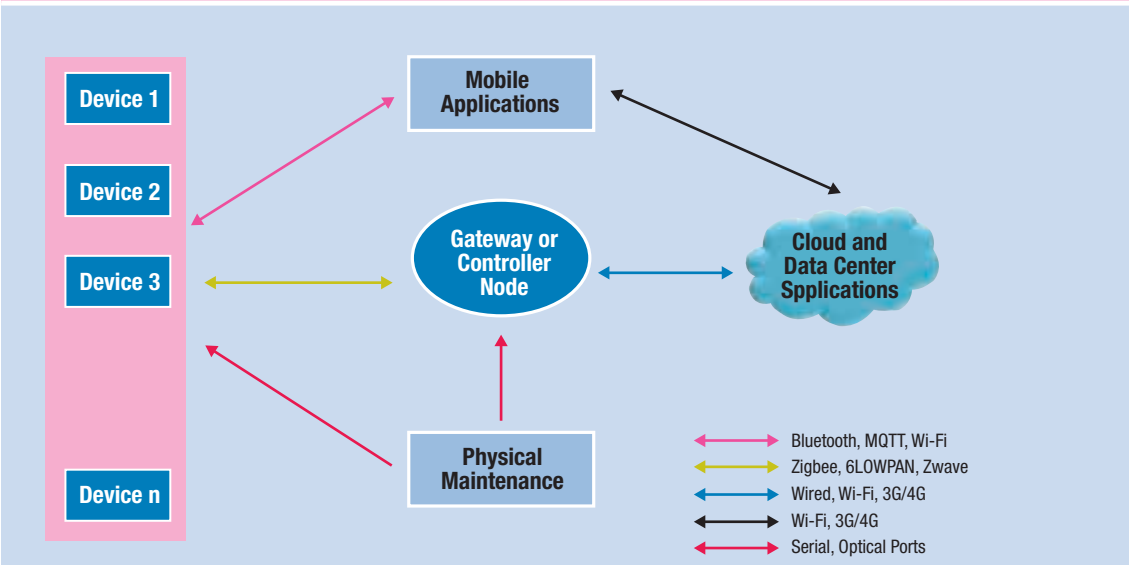
During the Internet of Things (IoT) Village held at the DEF CON security conference in August 2016, 47 new vulnerabilities affecting 23 IoT devices from 21 manufacturers were disclosed.[1] Among these 47 vulnerabilities were software-related vulnerabilities, e.g., design flaws, hard-coded passwords, configuration secrets, cryptographic issues, and common coding flaws, such as buffer overflows, invalidated inputs and command injection.

The software running on the devices, gateways, mobile and data center applications, and the interfacing application program interfaces (APIs) from which the services are consumed should be subjected to assurance to ensure they are free from security defects. This article describes the assurance techniques and processes involved in securing such applications and provides guidance on implementation across the IoT environment.

IoT Software Components

Each IoT component has its own software. How can there be assurance that, while running in a real environment, the component is not allowing malicious persons to hack the software and get access to the data and information collected by devices? Secure software development life cycle (S-SDLC) is the answer to software security assurance. **Figure 1** depicts typical IoT components.

Figure 1—Typical IoT Components



Source: S. Subramanian and B. Swaminathan. Reprinted with permission.

Sivarama Subramanian, CISA
Is principal security architect at Cognizant Technology Solutions, where he is leading the integrated vulnerability assessment delivery and research, enabling new service rollouts and aligning new security trends with customer needs. Subramanian can be reached at sivaramasubramanian.kailasam@cognizant.com.

Balaji Swaminathan M., CISA, CISSP
Is a security architect at Cognizant Technology Solutions, where he is managing the integrated vulnerability assessment delivery and research, enabling new service rollouts, and aligning new security trends with customer needs. Swaminathan can be reached at balajiswaminathan.m@cognizant.com.

Security should be embedded into the development cycle of the IoT components—be they the device firmware, gateway source code, application source code or API source code.

Applications in a typical IoT environment might fall into one of the following categories:

1. Device applications that reside on the nodes and gateways, e.g., a node.js- or python-based application running on a smart energy meter
2. Controlling applications that control and regulate the IoT environment, e.g., web-based or non-web-based applications built on Java, Dot Net, Perl or PHP, typically residing in the data center or the mobile application operating system, from where the devices can be controlled
3. Consuming applications that receive data from the devices for further processing, e.g., web- or non-web-based applications, typically residing in the data center, that perform analytics on the received data
4. Relay services that format and transfer data between different components, e.g., APIs, web services that transport the data across devices, applications and other IoT components

There are two categories of vulnerabilities related to software. Those are:

1. An insecure web interface, which could allow some of the following common attacks to take place:
   - **Injection attacks**—Malicious execution of scripts, database queries, system-level commands injected to apps
   - **Unprotected secrets**—Cleartext/unencrypted configuration secrets, keys and passwords
   - **Privilege escalation**—Elevating user privileges and user impersonation
2. Insecure software/firmware that allows malicious users to change the software or compromise the device, which can be used as a BOT (software robot) device. The following are some common attacks:
   - **Firmware corruption**—Sending malicious firmware that could improperly format the device logic or install a backdoor
   - **Unsigned firmware**—Forcing the devices to download firmware from unauthorized sources without validation

Following a secure SDLC approach would mitigate the common coding flaws and software vulnerabilities related to IoT components.

Secure SDLC Steps to Address Common Coding Flaws and Software Vulnerabilities

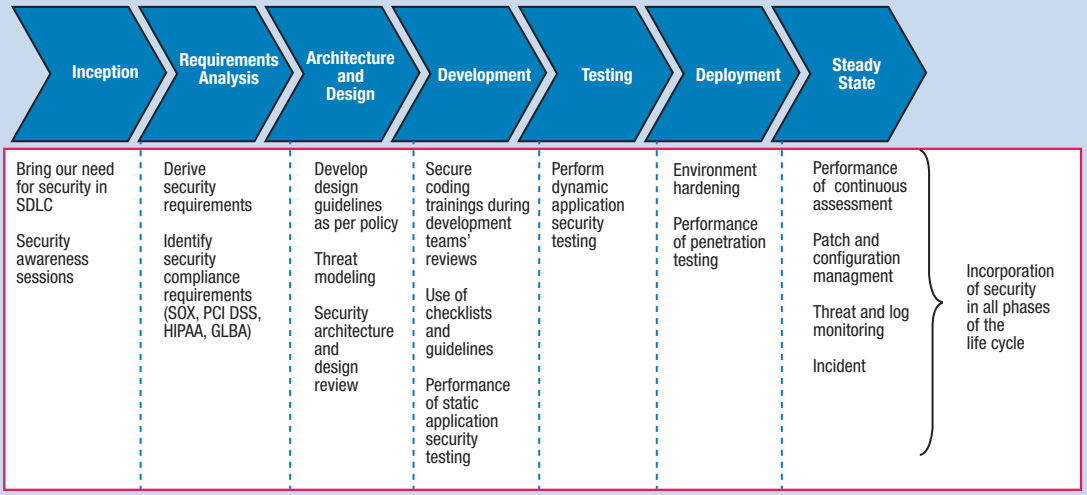
The cost of fixing a security bug varies depending on where it is discovered. If it is discovered in the production environment, the cost of fixing it would include the tangible costs of the developer effort, tester effort, user acceptance testing effort and deployment effort, and the intangible cost of reputation and customer trust. If it is discovered during the design phase, then it is very easy to correct the design flaw and introduce a security measure during the development phase. The cost of fixing a defect postproduction is approximately four times more than fixing it in the development stage.[2]

> **“The cost of fixing a defect postproduction is approximately four times more than fixing it in the development stage.”**

IoT software assurance is the level of confidence and trust that software is free from vulnerabilities (either intentionally designed into the software or accidentally inserted at any time during its life cycle) and the software functions in the intended manner.

Security assurance for IoT applications can be best achieved through the adoption of a defense-in-depth strategy, which, in turn, warrants having a secure SDLC practice (**figure 2**) in place. The principal intent is to build security within the life cycle of these applications from ground zero that potentially and gradually reduces the flaws in security, design, implementation and deployment. Proper adherence to such assurance best practices will result in applications devoid of vulnerabilities that might have

Figure 2—A Secure SDLC



Source: S. Subramanian and B. Swaminathan. Reprinted with permission.

been introduced accidentally or intentionally at any point of time in their life cycle.

Inception and Requirements

Effective security management of the components involved is a critical focus area because a typical IoT environment is spread widely, both physically (with numerous devices) and logically (with multiple technologies and applications). This effective management can be achieved only with a proper inventory of what is to be managed, such as:

- **Category by development type**—The category by development type depends on where the software/API was developed, e.g., in-house development, vendor/partner development, commercial/commercial off-the-shelf or open source. This is also vital in deriving the security governance and policies with which to adhere for each of these types.
- **Category by application type**—The category by application type depends on where the software or the API resides, e.g., device (nodes or gateways), cloud/data center servers, mobile or desktops. This would dictate the secure coding guidelines, checklists, security configuration applicable to each of the application types in line with the security policies.

"Awareness sessions on application security, threats and recent breaches have to be conducted early in the life cycle and on a regular basis to impart the necessity of enforcing security in the applications at all stages in the life cycle."

The S-SDLC control gates, such as design review/threat modeling in the design phase or static application security testing in the development phase, have to be mandated. The entire SDLC cycle has to be monitored and managed for continuous improvement in delivering rapid-yet-secure software to production. Such managed solutions are vital to ease the security assurance process and are highly recommended.

Depending on the risk levels perceived for the applications, the control gates can be derived. Incremental and rapid deployments require oversight throughout the development life cycle until operation, especially in the case of enhancing via DevOps, along with a well-defined acceptance criterion from a security standpoint.

Awareness sessions on application security, threats and recent breaches have to be conducted early in the life cycle and on a regular basis to impart the necessity of enforcing security in the applications at all stages in the life cycle.

Design Phase

During the design phase, the architecture and design of the IoT solution would be reviewed using threat modeling techniques. Threat actors and possible threat scenarios should be enumerated for each of the IoT components. For example, a threat scenario could be a possible data integrity issue due to a lack of authentication controls in the device. After the enumeration of threats, countermeasures can be ranked and recommended.

The standard approach for threat modeling using the STRIDE and DREAD models is as follows:

- **STRIDE**—Threat categorization considering spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privileges
- **DREAD**—Threat ranking attributes considering discoverability, reproducibility, exploitation, affected users and damage potential

This method will suffice for reviewing the IoT architecture, with an additional threat focus on physical security of the devices. Common threats applicable to physical security are device theft, device cloning and unauthorized physical access. The primary difference in the standard threat model analysis and design reviews is the application of knowledge of industry-specific threats. **Figure 3** shows some of these unique industry threats.

Figure 3—Industry-specific Threats

Industry	Threats
Automotive	Human safety, unauthorized control of self-driving vehicles, global positioning system (GPS) use
Health care	Medical data theft, manipulation of medical devices (e.g., pacemaker, blood pressure monitor, defibrillators) that could malfunction and pose a threat to patients' lives
Energy	Power theft, unauthorized metering data calibration, grid shutdown
Consumer electronics	Electronic appliance malfunction that could lead to threats on a consumer's life or unauthorized device consumption

Source: S. Subramanian and B. Swaminathan. Reprinted with permission.

The controls that need to be built into the IoT applications and the APIs are:

- **Input validation**—Handling input data from users, apps and services
- **Authentication**—Identifying and verifying users and traffic
- **Authorization**—Enforcing access control for all requests to the application
- **Configuration management**—Securing configuration data and metadata, console access
- **Data security and privacy**—Security of sensitive and confidential data, such as protected health information or other personally identifiable information
- **Session management**—Safely initiating, handling and terminating an application's sessions
- **Cryptography**—Using strong encryption, hashing and key exchange algorithms, digital certificates and signatures
- **Exception management**—Safe handling of application exceptions

- **Auditing and logging**—Recording all events with required attributes for accountability
- **Communication security**—Securing traffic to and from the application
- **Availability**—Safe handling of loads on the applications

The output of design review and threat modeling activities should enforce the incorporation of only standard and authorized frameworks, modules, APIs and design specifications, e.g., Spring Security for access control or AES 256 and above for encryption. This ensures a secure baseline is built into the design, which will flow through the SDLC, thus greatly reducing the number of security defects in later phases. Unauthorized or unverified frameworks or APIs that are pulled from public repositories should be avoided. Should there be a business need for the usage of such modules, the following steps should be completed:

- Enumerate if there are any known vulnerabilities and exploits associated with the codebase.
- Ensure only the updated version is used.
- Thoroughly analyze the code and fix the vulnerabilities (development phase).

Security design solutions for the perceived and standard threats have to be carefully weighed and provided based on multiple factors, e.g., use cases involved, input and output, application type and technology, and device specifications. Educating the application architects and developers on secure design guidelines and incorporating these guidelines into the design will also greatly improve the security baseline.

Development Phase

During the development phase, whether it is Agile or waterfall development, secure coding training based on the Open Web Application Security Project (OWASP) Top 10[3] or Top 9,[4] SANS Top 25,[5] or CERT principles[6] should be imparted to all the developers and has to be enforced at regular intervals to stay abreast of the latest developments against new attacks and threats. Developers should also be trained on embedded systems programming, or embedded systems programmers used to design the apps that specifically sit on the devices/hardware. Such apps can either be full-fledged or enhancements to existing apps.

> **“Continuous integration of security best practices, tools and assessments to aid in continuous delivery must be practiced and implemented with automation.”**

In addition to the standard vulnerabilities and insecure coding practices in the common programming languages (e.g., Java, J2EE, PHP, .NET), hardware-specific code and code governing the embedded systems (Embedded C/C++) should be inspected based on the secure coding standards mentioned previously. Custom vulnerability signatures and test cases can also be developed depending on the nature of the logic implemented, APIs and libraries that lie on the devices that govern the underlying hardware, persistent and nonpersistent storage, firmware life cycle, coding techniques, and defects that reside in open sources.

Awareness should also be imparted to the developers on the IoT attack and threat surfaces, because the software they are expected to develop is supposed to interact with real-world objects. Their software integrated development environments (IDEs) should be enabled with secure code review plugins to do security checks during the checkout time. An independent security analyst could review the stable code for any of the security

flaws mentioned previously, unauthorized access to secrets and secret-key session issues.

For rapid and incremental deployments, exercising formal control gates (as in traditional waterfall models) might not be practically feasible. Continuous integration of security best practices, tools and assessments to aid in continuous delivery must be practiced and implemented with automation. In other words, continuous development is and should always be accompanied by continuous automated assessments to ensure that all software and API changes are security-vetted and only a signed-off build is propagated to the next phase. Failed builds should be automatically fed back for remediating vulnerabilities.

Assessing a software or API can be more effectively achieved when the source code is available, as it can be directly inspected for vulnerabilities. When blackbox products, whose design and source code are not available, are used only a standard vulnerability assessment can be performed on the apps or APIs in the test phase. Nonstandard open source software and APIs that are leveraged from public repositories should be given due attention, as specified in the design phase. Since such software and APIs might not have undergone a secure development, they should be thoroughly examined for known and custom vulnerabilities. This examination should use tools that are specialized in identifying vulnerabilities in open-source software.

While the requirements serve as user stories (in Agile modes), developers can leverage the plugins and tools that integrate with the build servers (e.g., Jenkins, Bamboo) and conduct assessments on the fly based on check-ins. Test cases and the vulnerabilities list should be continuously revised and fed back to the cycle, and the cycle should move to the next phases based on the acceptance criteria. The same mode of continuous delivery enhanced via continuous assessments can be leveraged for the testing phase as well.

Testing Phase

Depending on the type of IoT application and the APIs in place, the necessity for and the type of security assessments to be performed can be decided accordingly. Some typical assessments that have to be performed include:

- **Vulnerability assessment or dynamic application security testing (DAST)**—Ensure the input and traffic to/from the application is thoroughly tested to enumerate the vulnerabilities that are prevalent as dictated by standards such as OWASP (Web Top 10,[7] IoT Top 10,[8] Mobile Top 10,[9] thick client), Web Application Security Consortium (WASC)[10] and SANS Top 25.[11] These have to be performed on all of the following applications used in the IoT environment:
  - Web apps
  - Mobile apps
  - Device apps
  - Thick clients
  - Web services and APIs (plain and representational state transfer)

“Assessing a software or API can be more effectively achieved when the source code is available, as it can be directly inspected for vulnerabilities.”

- **Reverse engineering and debugging**—Reverse the applications from their binaries; interpret any hidden logic, controls and secrets; and repack to their original state after bypassing and altering the hidden logic. This will be more applicable to mobile apps and device apps, where much of the application logic and controls are housed.

As with every SDLC phase, the testing phase associated with the applications should consist of the assessment activities pertaining to the physical endpoints (e.g., sensors and nodes).

Physical risk could range between any extremes depending on the use cases involved. Some sample abuse cases are:

- Bypassing device enrollment and registration
- Cloning and stealing devices
- Simulating physical movements to bypass sensors and actuators
- Abusing protocols, e.g., ZigBee, Z-wave, 6LoWPAN

Depending on the use cases or the functional flows perceived for each device and application, the vulnerability test cases should be designed as appropriate. The test methodology should consider all use cases pertaining to the complete IoT environment, and every such use case should have one or more misuse case (security test case) associated with it.

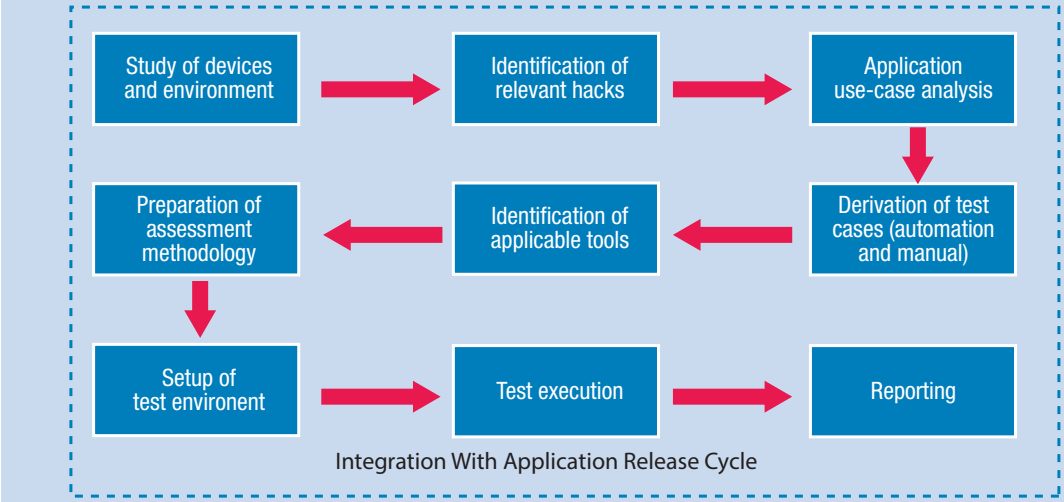
Wherever possible, vulnerability test cases and test scripts have to be initiated via automation, as and when the applications are signed off by functionality and performance testing. More logical and business use cases must be targeted manually in parallel. The standard approach for security testing is detailed in **figure 4**.

The test environment should be scalable enough to account for the data generation and aggregation to be fed to and consumed by these applications to mimic the real world. The live environment where the apps and devices will be deployed is widespread and will receive data from many endpoints.

It is worth noting that security testing for an IoT environment does not stop just with the applications. The scope is as broad as the number of components involved, e.g., sensors, actuators, gateways and the underlying infrastructure. Key target areas should include:

- **Smart devices**—Device disassembly and review, memory extraction, attacks on buses and fuzzing through physical ports
- **Firmware**—Static and dynamic analysis, reversing, malicious firmware injection and signing
- **Communication**—Traffic analysis, protocol decoding and fuzzing, packet replays, and cryptographic attacks

Figure 4—Security Testing Approach



Source: S. Subramanian and B. Swaminathan. Reprinted with permission.

- **Infrastructure (deployment phase)**—Vulnerability assessment, penetration testing and hardening

> **“Early in the inception phases, a centralized management and monitoring solution is imperative to track the IoT environment and its components.”**

Deployment Phase

While a majority of off-device apps adhere to the common environment hardening guidelines and are subjected to penetration testing, on-device apps require special considerations for a secure deployment.

Device manufacturers should comply with security guidelines mandated by the respective industry consortium groups (e.g., Institute of Electrical and Electronics Engineers [IEEE] for energy, US Food and Drug Administration [FDA] for medical devices, Society of Automotive Engineers [SAE] for automotive devices, Consumer Electronics Association [CEA] for consumer electronic devices). Common devices prevalent in the industries are:

- **Energy**—Smart grid, smart meters, relays, programmable logic controllers (PLCs)
- **Medical devices**—Smart pacemakers, defibrillators
- **Automotive**—Smart or driverless vehicles
- **Consumer electronics**—Smart home appliances

Just as APIs and applications undergo a secure SDLC, manufacturers should ensure that these devices are subjected to secure development life cycle from device circuit designing, assembling and setup, to rolling out to customers.

Unwanted logical and physical ports should be turned off in such devices and servers, and physical security procedures should be tightly employed to detect and prevent against attacks such as device/sensor theft, tampering and unauthorized access. All such attempts should be monitored, alerted, logged and defended effectively.

For leveraging cloud-based services (especially in the case of Software as a Service), where the apps are exposed as services and APIs, customers should work with the service providers for obtaining the assessment and audit results of the apps and the infrastructure being relied upon to provide assurance.

Standard vulnerability assessment and penetration testing have to be conducted on all the IoT environment components, especially the servers hosting the applications and APIs to identify and mitigate the vulnerabilities pertaining to the operating system and platform that could lead to a compromise. Real-world penetration testing should also focus on serving load traffic to the devices and the applications.

Operations and Steady State

Early in the inception phases, a centralized management and monitoring solution is imperative to track the IoT environment and its components (applications, devices, sensors).

Automation of vulnerability, patch and configuration management must be exercised. Every single application and node must be subjected to continuous monitoring to aid in automated threat and attack detection and response, which will fuel additional confidence in security assurance. In addition, continuous vulnerability assessments, penetration testing and security maintenance have to be carried out to cope with the ever-increasing attacks and threats, and defended accordingly.

Business Use Case

A typical vehicle parking system consisting of device and upstream applications underwent security assurance via secure SDLC. The high-level IoT architecture is depicted in **figure 5**.

The device sensors identify the availability of a parking spot by detecting the electromagnetic field created by a vehicle's presence or absence. These devices feed the parking data to a local gateway through the data bus wired from them, and then the gateway communicates with the customer's cloud applications. From the cloud, users can pull, query and reserve the parking spot by using mobile apps. Applications that were subjected to security assurance are:

1. **Device applications**—D1, D2, D3, D4
2. **Node.js-based thick clients**—Managing the electromagnetic sensors attached to them
3. **Gateway application**—Java middleware logic
4. **Cloud application**—Java/J2EE web and mobile
5. **Mobile apps**—Android and iOS apps

All of the aforementioned applications were subjected to secure SDLC, and vulnerabilities identified in each stage in the life cycle were addressed before moving to the next phase. The mobile applications were also reverse-engineered to disassemble and debug them to enumerate vulnerabilities pertaining to data storage security, access control and sensitive secrets.

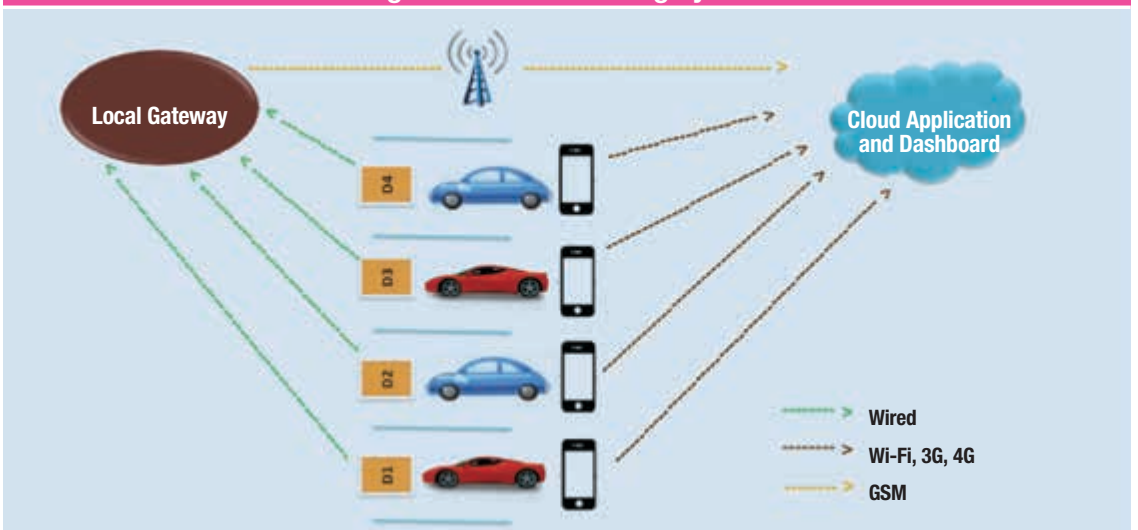
The gateway and cloud servers holding the application tiers underwent network penetration testing, and configuration hardening was conducted on the gateway and cloud servers hosting the application tiers to identify and prevent the vulnerabilities in the platforms and operating systems.

In addition, physical penetration testing was also performed, e.g., tricking the sensors, cloning or impersonating sensors, and tampering with the hardware. Abuse cases were designed based on the business rules laid out by the customer and executed as appropriate in the testing phase.

The following are the number of security defects enumerated in the respective SDLC stages:

- Design: 41
- Development: 26
- Test: 17
- Deployment: 11

Figure 5—Smart Parking System



Source: S. Subramanian and B. Swaminathan. Reprinted with permission.

The following are typical data for cost per defect ranges[12] (in US dollars):

- Requirements: $250
- Design: $500
- Coding, testing and implementation: $1,250
- Post-release: $5,000

Total cost incurred in defect fixing: (41 flaws*$250) +([26 flaws+17 flaws+11 flaws]*$1,250) = $77,750.

Had the apps not been subjected to secure SDLC, all the defects would have propagated to the final build and would have accumulated in the production:

- Total number of defects: 41+26+17+11 = 95
- Total cost that would have been incurred for post-production fixes: 95 defects*$5,000 = $475,000

Total cost savings reached: $475,000 – $77,750 = $397,250

In other words, the cost that would have been incurred for fixing defects postproduction would be approximately five times the cost incurred for fixing the defects in individual SDLC phases. Note that the previous amounts are an approximate indicator on the costs involved and not the actual figures.

Conclusion

Security assurance for IoT applications and APIs has to be embedded throughout every stage of the SDLC and must be continuous. It is critical to maintain a proper inventory and configuration of all the application components building up the IoT environment. Security oversight or verification must be part of the requirements stage. Proactive measures such as training sessions and usage of security standards, guidelines and checklists have to be mandated in all applicable stages, and validation measures are achieved by performing reviews and assessments in all stages.

Security assessment tools should be integrated into the development and quality assurance cycles to trigger assessments on the fly and, wherever possible, employ automation. Only authorized

frameworks, standards and APIs should be used, and due care exercised on open-source components. To counter the most recent vulnerabilities and threats, continuous assessments, threat monitoring and security patching must be conducted once the IoT devices, applications and APIs are exposed to production. Having security embedded in the IoT development cycle ensures that known security issues are fixed and new ones prevented with the most effective measures, thus providing security assurance for end users.

Endnotes

1 Dark Reading, "IoT Village at DEF CON 24 Uncovers Extensive Security Flaws in Connected Devices," press release, 16 September 2016, *www.darkreading.com/attacks-breaches/iot-village-at-def-con-24-uncovers-extensive-security-flaws-in-connected-devices/d/d-id/1326928*

2 Jones, C.; *Software Quality Metrics: Three Harmful Metrics and Two Helpful Metrics*, Project Performance International, 6 June 2012, *www.ppi-int.com/systems-engineering/free%20resources/Software%20Quality%20Metrics%20Capers%20Jones%20120607.pdf*

3 Open Web Application Security Project, Top 10 2013-Top 10, *https://www.owasp.org/index.php/Top_10_2013-Top_10*
4 Open Web Application Security Project, The OWASP Code Review Top 9, *https://www.owasp.org/index.php/The_Owasp_Code_Review_Top_9*
5 Martin, B.; M. Brown; A. Paller; D. Kirby; *2011 CWE/SANS Top 25 Most Dangerous Software Errors*, The MITRE Corporation, 13 September 2011, *http://cwe.mitre.org/top25/*
6 Confluence, SEI CERT Coding Standards, Software Engineering Institute at Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, *https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards*
7 *Op cit,* OWASP Top 10 2013-Top 10
8 Open Web Application Security Project, OWASP Internet of Things Project, *https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Vulnerabilities*
9 Open Web Application Security Project, Mobile Top 10 2016-Top 10, *https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10*
10 Web Application Security Consortium, "The WASC Threat Classification v2.0," Threat Classification Wiki, *http://projects.webappsec.org/w/page/13246978/Threat%20Classification*
11 *Op cit,* Martin
12 *Op cit,* Jones