Building Security into Your Apps One Story at a Time

Bhushan B. Gupta
Gupta Consulting LLc.
Bhushan@bgupta.com

Abstract

In the agile software development, a story is the smallest element of your application and setting **an** appropriate security threshold dictates the security of your application.

This paper discusses the implementation and validation of security controls in the lifecycle of a story in the agile software development environment. The three 'Ws'; what, when, and who are emphasized with reference to WHAT security controls to implement, WHEN to verify and validate the implementation, and WHO should assure that the security control provides the intended safeguard. The role and time of engagement of product owner, security engineer, quality engineer, and test engineers are explained as a story progresses from one stage of the lifecycle to the next.

With the help of examples, the paper demonstrates the necessary security controls at the product definition. Once the security controls for a story are defined, the implementation needs verification by the security engineers and the product owner. The test team is then responsible to validate that the security controls are working as intended in the context of the application and without degrading the customer experience. The paper also highlights the post deployment security related activities and measures that should be taken for an uninterrupted operation.

Biography

A proven champion for quality, well-versed with software quality engineering, and a WebApp security researcher, Bhushan is the principal consultant at Gupta Consulting, LLC. In WebApp security, his research areas are; infusing security in SDLC, OWASP Top10, Risk Analysis and Mitigation, Attack Surface Measurement, and Static and Dynamic Application Security Analysis. As a leader of Open Web Application Security Project (OWASP) Portland Chapter, he is dedicated to driving the web application security to higher levels via technical education and training. Bhushan often provides training workshops and presentations to corporations and non-profit organizations. He is also an invited speaker and a panelist in discussions for both application security and agile software development. Bhushan serves as a Program Team member for the Pacific Northwest Software Conference and has been a member of the Program team for the Global AppSec Conference 2020 organized by OWASP.

Bhushan has been a Certified Six Sigma Black Belt (American Society for Quality and Hewlett Packard) and possesses deep, but broad experience in solving complex problems, change management, and coaching and mentoring. Bhushan has a MS in Computer Science (1985) from New Mexico Tech and has worked at Hewlett-Packard and Nike in various roles. He was also a faculty member at the Oregon Institute of Technology, Software Engineering department, from 1985 to 1995 and is currently an Adjunct Faculty member.

Introduction

Web application security is becoming increasingly important as the extent of breaches has increased. The software development community is also realizing that application security should be given due consideration throughout the development lifecycle and not be an afterthought. The Open Web Application Security Project (OWASP) has now ranked insecure design as the 4th in its Top-10 vulnerability list (OWASP 2021). In his recent book, Kohnfelder (Kohnfelder, 2022) has emphasized the importance of secure design along with threat modeling, mitigation, and design patterns.

Although not the only software development paradigm, the agile software development life cycle (SDLC) is widely used in the software industry. The basic element of the agile SDLC is a story and a collection of stories makes an epic. A story goes through a well-established transformation throughout the development as shown below:

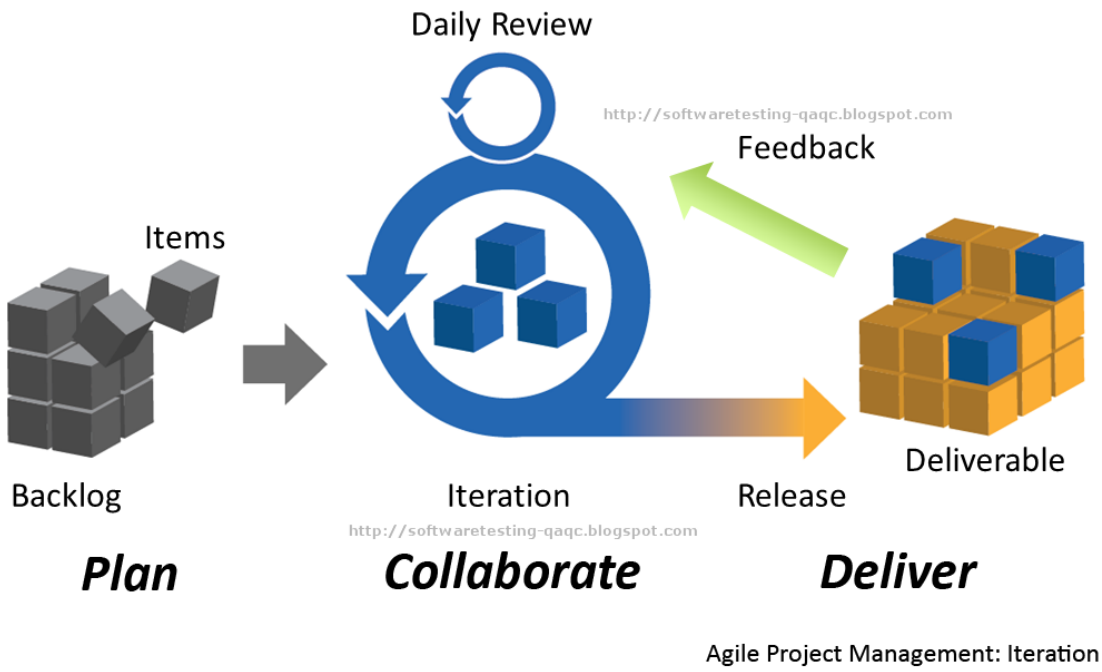

Fig 1: Lifecycle of a Story in the Agile SDLC

Like any other attribute of software quality, such as usability, security can be and should be integrated at the earliest point possible in development. The following discussion will focus on how to integrate security in each iteration and an ultimately deliver a minimally vulnerable web application.

Scope

Multiple factors influence web application vulnerability. Some of the important ones are the operating system, the platform and the framework the application is built on, cryptography, the hosting environment (local vs. cloud), and the system configuration. They all introduce their own risks and some may have lower risk than the others. For example, the Django development framework is considered more secure than others.

The scope of this article is limited to the stories, the functional aspects of the web application, and not the platform the application is running on, the development stack, and how is it hosted. However, there are scenarios where a component of the hosting environment such as the memory management scheme

provided by the operating system, effects the application vulnerability and should be given due consideration in the design and development of the story.

Story Characterization

Not all stories impact the application security at the same level, some are more susceptible to breach than the others. This section is an attempt to classify stories based upon their vulnerability.

High Vulnerability Stories

These stories are directly associated with application security. For example, “As a user of the application, my personal data should be protected”. This story explicitly dictates that the data should be protected at all levels. It is not specific to any particular data entity. Another story, “As a user, I should be the only entity to login to my account” is also a high vulnerability story. This story is also about data protection but specifically refers to a specific set of data. Stories that are more specific to security are more tangible and actionable as we design a secure solution.

Medium Vulnerability Stories

In an online banking system, a typical story would be “As a user, I should be able to make a deposit”. The typical process for this story will be login into your account, follow the deposit process and submit your transaction. A man in the middle attack may redirect the money to her/his account. This may cause the loss of money but the privacy of the account is not compromised for further data breach.

Low Vulnerability Stories

“As a user I should be able to add an item to the cart” is a story that has a very little risk associated. There is a potential that a hacker may change the price of the item. But otherwise the action will be safe.

Threat Modeling

Realizing that not all stories have the same level of threat, it is important to analyze the threat level of a story. Several threat modeling techniques exists (Shevchenko, 2019) and the one most applicable should be used to assess the risk. A popular techniques is STRIDE (an acronym for **S**poofing Identity, **T**empering of Data, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege) and has been widely published. It is possible to enumerate STRIDE for a set of stories by assigning a value to each element in STRIDE and establishing the overall threat as a numeric value. Appendix A provides examples of how to enumerate the threat for a few selected stories.

The threat modeling is helpful in evaluating the security risk associated with the story. In these times of “first to market”, a good technique to prioritize your vulnerability helps you focus on the most vulnerable stories and thus minimize the threat.

Security Controls

In the literature (Harris, 2012), the security has been defined by a triad, **C**onfidentiality, **I**ntegrity, and **A**ccessibility (CIA) and it is important to strike a balance between these elements. An extreme example will be, your application will be very secure if you lock it up. But, it will not serve any purpose because no one will be able to use the system. This section will describe the security controls specific to each element.

Confidentiality

Confidentiality implies that the data is exposed in an authorized manner during its entire life time – at creation, in motion, and while stationary. If a story does not involve sensitive data, confidentiality is not important. For example – adding items to a shopping cart does not include any sensitive data. It may be tempting to treat every data element of a story "confidential" but this approach warrants securing the entire data. In this scenario, it will be necessary to develop a robust acceptance criteria, a secure design, coding, and verification, and finally an exhaustive validation of the story.

Confidentiality is preserved by access management which is enforced mainly by two security controls identification and authentication. NIST (National Institute of Standards and Technology) article 800-53, REV. 5, (NIST Task Force, 2020) recommends a set of controls that should be given due consideration while evaluating a story for confidentiality.

Integrity

Integrity is defined as keeping data safe from unauthorized modification and removal. While we tend to think of integrity as a threat from the external elements, due consideration must also be given to malicious and disgruntled internal users. The integrity of the data may also be compromised by an unintended action of a normal user. Some of the necessary security controls are using secure socket layer to transmit data, encrypting data both in motion and while at rest, certificate management, and controlling authorization including privilege escalation.

Availability

Is your web application available for use during the intended hours of operation? DOS (Denial of Service) vulnerability can be very damaging to your organization. You not only lose revenue, it also drives customers away causing further revenue impact. Security controls such as "no single point of failure" are more applicable at the systems level. This control requires an existence of an additional instant of the data that can be activated when the existing data instant is impacted by at DOS attack.

Testing Evolution in a Story Development

This section will elaborate on how a story progresses through its agile life cycle and the actors who play important roles in each phase. RACI framework has been utilized to simplify the discussion and keep is organized. RACI is a responsibility assignment matrix and an acronym for different roles in the matrix as described below:

R – Responsible: those who do the work to complete the task
A – Accountable: the one who is ultimately answerable to complete the task
C – Consulted: those whose advice should be sought to develop the solution
I – Informed: those who are kept up to date about the progress of the task

From the RACI perspective, in an agile environment, the scrum master, product owner, development engineer, security engineer, and test engineer are the most active roles. In the RACI matrix, the informed could apply to numerous roles and therefore it is not critical to highlight unless it is absolutely necessary in the context of the story. A manager role has been used where it is more reflective than the scrum master. Otherwise, the scrum master can very likely be treated "accountable" for all activities.

Story Context

To illustrate the essential elements of story development, we will consider the following story; it is generic in nature and essential for almost every commercially intended web application.

Story: Sign Up
As a customer, I want to setup an account, so that I can login to the portal and make a purchase.

This story is a high impact story as it contains not just the personal identifiable information (PII), but also has login credentials, data in motion and finally at rest at the server side.

In a software development organization that follows best practices, this story will go through the following agile iteration steps:

Fig 2: Typical Story Progress in an Agile Iteration

While the first three steps are self-explanatory, in the web security community, static testing refers to any type of testing that is performed when the code is raw or compiled and is termed as SAST (Static Application Security Testing). Dynamic testing is when the code is in the executable state and is known as DAST (Dynamic Application Security Testing). The rest of the section details each iteration step with using the RACI matrix.

Acceptance Criteria

Establishing an acceptance criteria is an important activity as it defines the boundaries for the story and sets the user expectations. In the context of security, it establishes which data elements should be protected and what security controls should be used to protect it. We will be using a list format for documenting the criteria and bear in mind, this list is not exhaustive by any means.

Criteria ID	Criteria Description
01	Collect only essential PII from the user
02	Hide sensitive data at submission (login ID, password)
03	Protect user data exposure when in motion
04	Protect user data exposure when stored
05	Protect user data from unauthorized changes when in motion and stored

Table 1: List of Acceptance Criteria for Sign Up

The RACI matrix to establish the acceptance criteria is:

R	A	C	I
Product Owner	Scrum Master	Security Engineer	Need based

Once established, the security team can add the security controls to the acceptance criteria. The updated list looks as follows:

ID	Description	Security Control
01	Collect only essential PII from the user	None/security policy
02	Hide sensitive data at submission (login ID, password)	Mask LoginID and Password at the data entry
03	Protect user data exposure when in motion	Encryption and Secure Transmission
04	Protect user data exposure when stored	Encryption

05	Protect user data from unauthorized changes when stored	Prevent privilege escalation Implement lowest level of privilege

Table 2: List of Acceptance Criteria with Security Controls for Sign Up

This sets up the security requirements for this story. The RACI matrix for this activity will be:

R	A	C	I
Security Engineer	Scrum Master	Product Owner	Need based

These security controls will provide guidance to the developers on how to design and develop the story and to the test engineers on how to validate it. The test case design team is ready to engage at this point.

Design/Code Development

These two activities, although serial in nature, are performed by the development team. When it comes to security there are some best practices that the team should put to use. Some of the common best practices with examples are listed below.

Best Practice	Example
Defense in depth	Two-factor authentication
Fail secure	System lockup if an unsafe activity is suspected
Protect weakest code	
Principle of least privilege	Allow lowest level of privilege
Use of prepared statement	Building database queries

Table 3: Best Practices for developing a Secure Story

The responsibility matrix for this phase is:

R	A	C	I
Development Lead Engineer	Development Manager	Product Owner, Security Engineer	Need based

Static Testing (SAST)

As described earlier, static testing is performed when the code is raw. The simplistic approach for SAST is to perform code reviews and compiling code with debug on. The typical vulnerabilities that can be detected from this method are buffer overflow, verification of encryption and secure transmission algorithms which would take an extra-ordinary amount of effort if performed by executing the code. A story may warrant a specific system configuration (also refers to as system hardening) which can be efficiently verified using the SAST approach. This approach is particularly valuable to verify individual stories such as the one being considered in this discussion.

The SAST, when performed manually, is time consuming and requires a subject matter expert to be effective. OWASP (OWASP, 2022) has provided a list of numerous open source tools that are capable of performing SAST. Commercial tools are also available with a limited history. While considering a tool, one should be aware of false negatives that will be harmful. The responsibility matrix for the SAST is:

R	A	C	I
Development Engineer (s)	Development Lead Engineer	Product Owner, Security Engineer	Need based

Dynamic Testing (DAST)

Dynamic testing is the validation of a story while executing the code. The story may be independently verified or may need other functional stories. As an example,

As a customer, I should be able to pay by credit card or PayPal.

The pre-requisite for the story will be another story that will prepare the cart to be ready for checkout. DAST testing is a combination of manual and automated testing. There are open source tools such as ZAP or Burpsuite that will facilitate a significant part of this testing. The manual testing is absolutely necessary to achieve a high level of confidence. There also exist commercial tools that can be deployed for DAST. A systematic approach to evaluate an open source DAST tools has been described by Brian Myers (Myers, 2019). These tools look for prominent vulnerabilities and are not meant for catch all type testing.

The responsibility matrix for the DAST is:

R	A	C	I
Test Engineer(s)	Test Manager	Product Owner, Security Engineer, Development Engineer	Need based

The following figure brings these aspects together for an agile development practice.

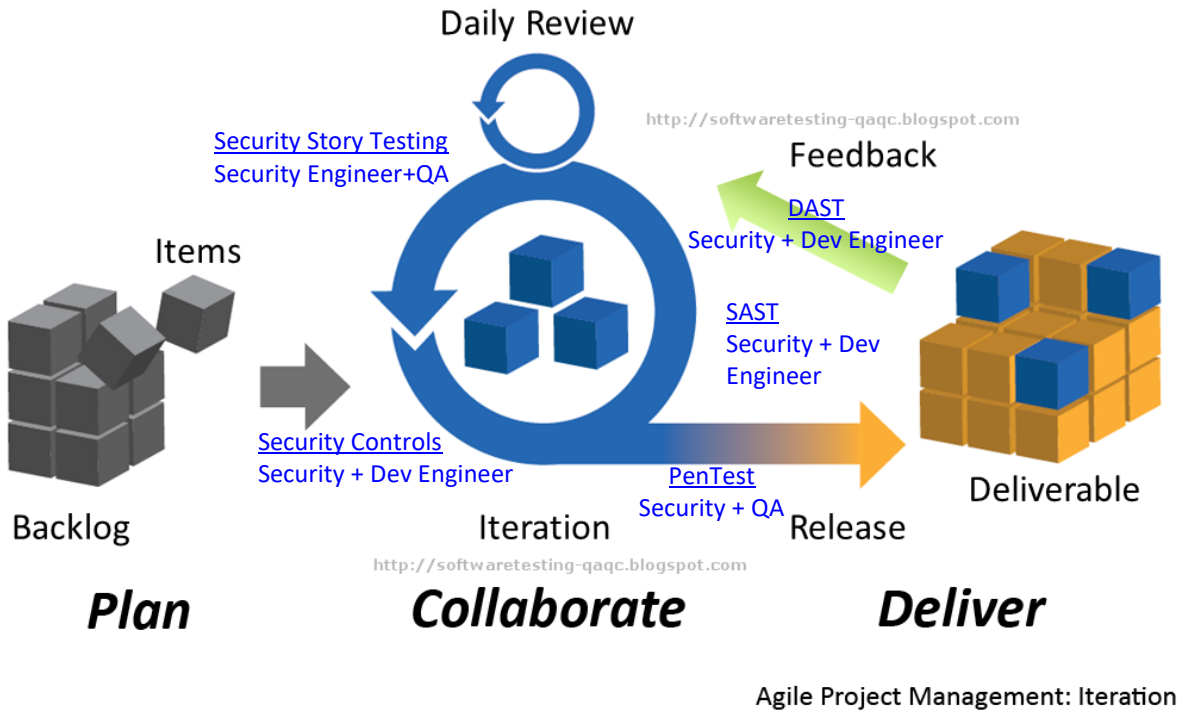

Figure 3: An Agile Iteration Cycle for a Story

Test Planning

The test planning for any story can begin as soon as the story acceptance criteria has been finalized. This section describes the essential elements of the test planning.

Test Case Development

The initial step will be to develop test cases, one or more, for each acceptance criterion. For example, test cases for the acceptance criteria 02 will be:

Test 01	Verify that the entry of login ID is not human readable
Test 02	Verify that the entry for the password is not human readable
Test 03	Verify that the login ID can be set to be human readable if desired
Test 04	Verify that the login password can be set to human readable if desired

Of course, the elaboration of the story at the design time should include the human readable requirement.

Type of Testing

A serious consideration should be given to what type of testing will be most effective and efficient for a story. If a story requires a specific system configuration such as access control at multiple levels, static testing will be most effective. Another example will be SQL injection where a code review can verify that a "prepared statement" has been used to build the SQL query and the query has not been created from the user input.

Efficient Use of Dynamic Testing

Dynamic testing is an effective way to gauge an overall understanding as it gives you a fairly detailed view of the vulnerabilities that exist in the system. A test plan should be enhanced to retest these vulnerabilities both using the dynamic and manual testing to eliminate any false negatives.

Testing Priority

The testing priority for a story in an iteration is driven by the vulnerability of the story. Needless to say, the high vulnerability stories must be given the top priority. The prudent consideration while prioritizing stories for testing is: the developer's confidence level. Any prior history of breach for a similar story either by the organization or the industry will also influence the testing priority of a story. Testing order for the stories in an iteration can be best achieved by a strong collaboration between the product owner, development engineer, and the test engineer. Here is the responsibility matrix:

R	A	C	I
Test Engineer(s)	Test Manager	Product Owner, Security Engineer, Development Engineer	Need based

The Team

Security testing function has evolved over the past few years given the damaging impact of a breach to an organization. With the evolution of different levels of testing, the organization of test teams has also changed over time. Described below, are the test teams that help achieve a high level of confidence in building a secure product.

Red Team

The concept of a red team was the first evolution of a security test team. The team attacks the application from the hacker perspective and its main objective is to breach the system in a realistic way. The team members are skilled in reconnaissance and have a deep knowledge of test tools and techniques. They are normally referred to as attackers.

Blue Team

The second evolution of test team organization was a “blue team” and includes security personnel (defenders) who work with the development team (builders). The developers know the code and joined with the expertise from the security engineer, this team can effectively explore vulnerabilities.

Purple Team

Purple team is a relatively new trend in security testing and it’s a cooperation between the blue and the red team to find vulnerabilities.

There have been thoughts in the industry about other teams such as yellow and green teams (Miessler, 2021). The functional aspects of these teams are not yet well established.

Conclusion

Multiple factors influence web application vulnerability and the development team has limited control on these factors. The most important of these factors, is secure development which requires an understanding of security controls and their implementation at the proper stages of development in the SDLC. Developing a secure web application is a team effort and cannot be left entirely up to a security engineer or security team. Actors such as product owner and development engineer along with the quality assurance engineer play an important role at various stages. Testing methods and tools are constantly evolving and organizations such as OWASP are providing guidance that, when followed, will yield web applications with a high level of security confidence.

References

OWASP, A04:2021 – Insecure Design, https://owasp.org/Top10/A04_2021-Insecure_Design/

Kohnfelder, Loren, 2022, Designing Secure Software, no starch press, San Francisco

Shevchenko, Nataliya, 2019, Evaluating Threat-Modeling Methods for Cyber-Physical Systems**,** https://insights.sei.cmu.edu/authors/nataliya-shevchenko/

Harris, Shon, 2013, CISSP, McGraw Hill Education, New York, USA

NIST Task Force, 2020, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf

OAWSP, Dave Witchers et al, https://owasp.org/www-community/Source_Code_Analysis_Tools

Myers, Brian, Starting a Security Program on a Shoe String, Pacific Northwest Software Quality Conference, Portland, OR, October 2019

Miessler, Daniel. 2021, The Difference between Red, Blue, and Purple Teams, https://danielmiessler.com/study/red-blue-purple-teams/

Appendix A – Enumerating Threat with STRIDE Example

Story ID/ Vulnerability	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privileges	STRIDE Score
Browse to Web Site	No	No	No	No	YES	No	1
Create user Profile, login, security questions	Yes	No (data stationary)	Yes	Yes – social engineering	No	No	2
Submit form	Yes	Yes	Yes	Yes – social engineering	Yes	Yes	5
Receive Confirmation	No	Yes	Yes	Yes (MitM)	No	No	3