Software Security Testing

We have covered so far the various types of software testing for quality assurance and the different methodologies for security testing. In the following section, security testing as it is pertinent to software security issues will be covered. We will learn about the different types of tests and how they can be performed to attest the security of code that is developed in the development phase of the SDLC.

Before we start testing for software security issues in code, one of the first questions to ask is whether the software being tested is new or a version release. If it is a version release, we must check to ensure that the state of security has not regressed to an insecure state than what it was in its previous version. This can be accomplished by conducting regression tests (covered earlier) for security issues. The introduction of any newer side effects that impact security and the use of banned or unsafe APIs in previous versions should specifically be tested for.

For software revisions, regression testing must be conducted and for all versions, new or revisions, the following security tests must be performed, if applicable, to validate the strength of the security controls. Using a categorized list of threats as a template of security testing is effective in ensuring comprehensive coverage of the varied threats to software. The NSA IAM threat list and STRIDE threat lists are examples of categorized threat lists that can be used in security testing. Ideally, the same threat list that was used when threat modeling the software will be the threat list that is used for conducting security tests as well. This way security testing can be used to validate the threat model.

Testing for Input Validation

Most software security vulnerabilities can be mitigated by input validation. Buffer overflows, Injection flaws, scripting attacks, etc. can be effectively reduced if the software just performs validation of input before accepting it for processing.

In a Client/Server environment, it is best recommended to perform the input validation tests for both the client and the server. Client side input validation tests are more a test for performance and user experience than it is for security. If you only have the time or resource to perform input validation tests on either the client or the server, make sure that validation of input happens on the server side for sure.

Attributes of the input such as its range, format, data type, and values must all be tested. When these attributes are known, input validation test can be conducted using pattern matching expression and/or fuzzing techniques

(covered earlier). Regular Expression (RegEx) can be used for pattern matching input validation. Some common examples of RegEx patterns are tabulated in *Table 5.2*. Tests must be conducted to ensure that the white-list (acceptable list) of input is allowed while the black-list (dangerous or unacceptable) of input is denied. Not only must the test include the validation of the white lists and black lists, but must also include the anti-tampering protection of these lists. Since canonicalization can be used to bypass input filters, both the normal and canonical representations of input should be tested. When the input format is known, smart fuzzing can be used otherwise dumb fuzzing using random and pseudo-random inputs values can be used to attest the effective of input validation.

Regular Expression	Validates	Description	Example
^[a-zA-Z"-'\s]{1,20}$	Name	Allows up to 20 uppercase and lowercase characters and some special characters that are common to some names.	John Doe O'Hanley Johnson-Paul
^([0-9a-zA-Z]([-\.\w]*[0-9a-zA-Z])*@([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA-Z]{2,9})$	E-mail	Validates an e-mail address.	mpaul@isc2.org user@mycompany.com
^(ht|f)tp(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&%\$#_]*)?$	URL	Validates a Uniform Resource Locator (URL)	http://www.isc2.org
(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,15})$	Password	Validates a strong password. It must be between 8 and 15 characters, contain at least one numeric value and one alphabetic character, and must not contain special characters.	
^(-)?\d+(\.\d\d)?$	Currency	Validates currency format. If there is a decimal point, it requires 2 numeric characters after the decimal point.	289

Table 5.2 – **Commonly used regular expressions (RegEx)**

Testing for Injection Flaws Controls

Since injection attacks take the user-supplied input and treat it as a command or part of a command, input validation is an effective defensive safeguard against injection flaws. In order to perform input validation tests, it is first important to determine the sources of input and the events in which the software will connect to the backend store or command environment. These sources can range from authentication forms, search input fields, hidden fields in web

pages, Querystrings in the URL address bar and more. Once these sources are determined, then input validation tests can be used as a test to ensure that the software will not be susceptible to injection attacks. There are other tests that need to be performed as well. These include the test to ensure that

- parameterized queries that are not susceptible to injection themselves are used.
- dynamic query construction is disallowed.
- error messages and exceptions are explicitly handled so that even boolean queries (used in blind SQL injection attacks) are appropriately addressed.
- non-essential procedures and statements are removed from the database.
- database generated errors don't disclose internal database structure.
- parsers that prohibit external entities are used. External entities is a feature of XML which allows developers to define their own XML entities and this can lead to XML injection attacks.
- white-listing that allows only alphanumeric characters is used when querying LDAP stores.
- developers use escape routines for shell command instead of custom writing their own.

Testing for Scripting Attacks Controls

Scripting attacks are possible when user supplied input is executed on the client because of lack of output sanitization. Tests to validate controls that mitigate scripting attacks should be performed. These include the test to ensure that

- Output is sanitized by escaping or encoding the input before it is sent to the client.
- Requests and inputs are validated using a current and contextually relevant whitelist that is updated with the latest script attack signatures and their alternate forms.
- Scripts cannot be injected into input sources or the response.
- Only valid files with approved extensions are allowed to be uploaded and processed by the software.
- Secure libraries and safe browsing settings cannot be circumvented.
- Software can still function as expected by the business if active scripting configuration in the browser settings is disabled.
- State management items such as cookies are not accessible from client side code or script.

Testing for Non-repudiation Controls

The issue of non-repudiation is enforceable by proper session management and auditing. Test cases should validate that audit trails can accurately determine the actor and their actions. It must also ensure that misuse cases generate auditable trails appropriately as well. If the code is written to automatically perform auditing, then tests to assure that an attacker cannot exploit this section of the code should be performed. Security testing should not fail to validate that user activity is unique, protected and traceable. Tests cases should also include verifying the protection and management of the audit trail and the integrity of audit logs. NIST Special Publication 800-92 provides guidance on the protection of audit trails and the management of security logs. The confidentiality of the audited information and its retention for the required period of time should be checked as well.

Testing for Spoofing Controls

Both network and software spoofing test cases need to be executed. Network spoofing attacks include Address Resolution Protocol (ARP) poisoning, IP address spoofing and Media Access Control (MAC) address spoofing. On the software side, user and certificate spoofing tests along with phishing tests and verification of code that allows impersonation of other identities as depicted in *Figure 5.9* need to be performed. Testing the spoofability of the user and/or certificate along with verifying the presence of transport layer security can attest secure communication and protection against Man-in-the-middle (MITM) attacks. Cookie expiration testing along with verifying that authentication cookies are encrypted must also be conducted.

The best way to check for defense against phishing attacks is to test users for awareness of social engineering techniques and attacks.

```
using System.Security.Principal;

WindowsImpersonationContext impersonationContext;
impersonationContext = ((WindowsIdentity)User.Identity).Impersonate();

//Insert your code that runs under the security context of the authenticating user here.

impersonationContext.Undo();
```

Figure 5.9 – **Code that impersonates the authenticating user**

Testing for Error and Exception Handling Controls (Failure Testing)

Software is prone to failure due to accidental user error or intentional attack. Not only should software be tested for quality assurance so that it does not fail in its functionality, but failure testing for security must be performed. Requirement gaps, omitted design and coding errors can all result in defects that cause the software to fail. Testing to determine if the failure is a result of multiple defects or if a single defect yields multiple failures must be performed. Software security failure testing includes the verification of the following security principles:

Fail Secure (Fail safe)

Tests to verify if the confidentiality, integrity and availability of the software or the data it handles when the software fails must be conducted. Special attention should be given to verifying any authentication processes. Test cases to attest the proper functioning of account lockout mechanisms and denying access by default when the configured number of allowed authentication attempts has been exceeded must be conducted.

Error and Exception Handling

Errors and Exception handling tests include testing the messaging and encapsulation of error details. Tests conducted should attempt to make the software fail and when the software fails; error messages must be checked to make sure that they do not reveal any details that are not necessary. Assurance tests to verify that exceptions are handled and the details are encapsulated using user-defined messages and redirects must be performed. If configuration settings allow displaying the error and exception details to a local user but redirects a remote user to a default error handling page, then error handling tests simulating the user to be local on the machine as well as if they are coming from a remote location must be conducted.

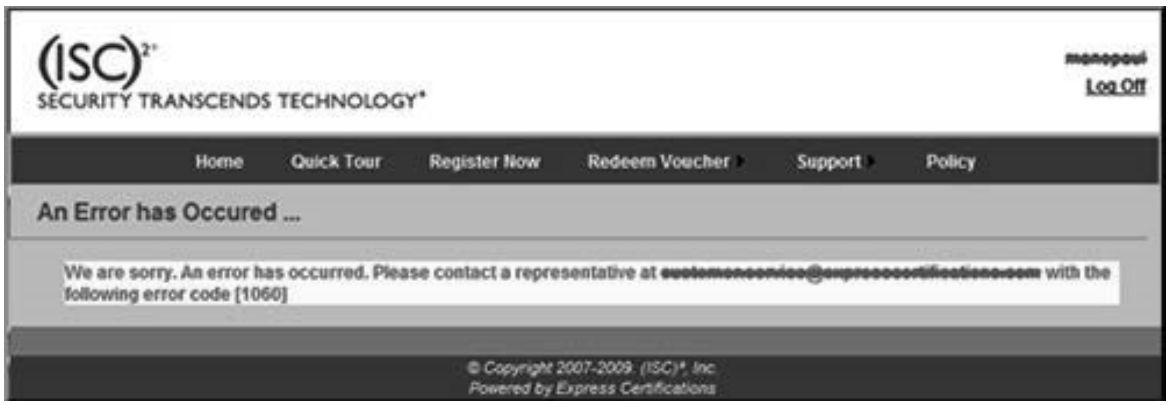

Figure 5.10 – **Reference identifier used to abstract actual error details**

If the errors and exceptions are logged and only a reference identifier for that issue is displayed to the end-user as depicted in *Figure 5.10*, then tests to assure that the reference identifier mapping to the actual error or exception is protected need to be performed as well.

Testing for Buffer Overflow Controls

Since the consequences of buffer overflow vulnerabilities are extremely serious, testing to ensure defense against buffer overflow weaknesses must be conducted. Buffer overflow defense tests can be both black box as well as white box in nature. Black box testing for overflow defense can be performed using fuzzing techniques. White box testing includes verifying

- that the input is sanitized and its size validated
- bounds checking of memory allocation is performed
- conversion of data types from one are explicitly performed
- banned and unsafe APIs are not used
- that code is compiled with compiler switches that protect the stack and/or randomize address space layout.

Testing for Privileges Escalations Controls

Testing for elevated privileges or privilege escalation is to be conducted to verify that the user or process cannot get access to more resources or functionality than they are allowed to. Privilege escalation can be either *vertical* or *horizontal* or both. Vertical escalation is the condition wherein the subject (user or process) with lower rights gets access to resources that are to be restricted to subjects with higher rights. An example of vertical escalation is a non-administrator gaining access to administrator or super user functionality. Horizontal escalation is the condition wherein a subject gets access to resources that are to be restricted to other subjects at their same privilege level. An example of horizontal escalation is an online banking user being able to view the bank accounts of other online banking users.

Insecure direct object reference design flaws and coding bugs with complete mediation can lead to privilege escalation thus parameter manipulation checks need to be conducted to verify that privileges cannot be escalated. In web applications both POST (Form) and GET (QueryString) parameters need to be checked.

Anti-Reversing Protection Testing

Testing for anti-reversing protection is particularly important for shrink wrap commercially off the shelf (COTS) software but even in business applications,