**Web Application Security**

**Terminology**

**Breach**: the loss of control, compromise, unauthorized disclosure, unauthorized acquisition, or any other similar occurrence where: a person other than an authorized user **accesses or potentially accesses personally identifiable information**; or an authorized user accesses personally identifiable information for unauthorized purpose.

**Threat**: Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, reputation), organizational assets, individuals, other organizations or the nation through an info system via unauthorized access, destruction, disclosure, modification of information and/or denial of service.

**Intrusion**: A security event, or a combination of events constituting a deliberate security incident in which intruder gains, or attempts to gain, access to system or system resource without authorization.

**Incident**: An event actually or potentially jeopardizing the CIA of an info system or the info the system processes, stores or transmits.

**Event**: Any observable occurrence in a network system.

**Exploit**: A particular attack exploiting system vulnerabilities, as a part of incident.

Vulnerability:

Zero Day:

**Common Categories of Vulnerabilities**:

**Broken Authentication**: encompasses various defects in the app’s login mechanism, causing an attacker to be able to guess weak passwords, launch a brute force attack or bypass the login.

**Broken Access Controls**: Cases where application fails to properly protect access to its data and functionality, potentially exposing users’ sensitive data on server to attacker, or enable attacker to carry out privileged actions.

**SQL injection:** enables attacker to submit a well-crafted input to hamper the app’s interaction with back end databases. An attacker may be able to retrieve arbitrary data from the app, interfere with its logic, or execute commands on the database server itself.

**Cross-site Scripting (XSS):** enables an attacker to target other users of the application, potentially gaining access to their data, performing unauthorized actions on their behalf, or carry out other attacks against them.

**Information Leakage**: exposing users’ sensitive data on server to attacker which helps to develop an assault on app, through defective error handling or other behavior.

**Cross-site request Forgery**: App users can be induced to perform unintended actions on the application within user context and privilege level. This vulnerability allows a malicious web site visited by victim to interact with application to perform actions user didn’t intend.

**SSL:** Secure Socket Layer. This technology upholds the confidentiality and integrity of data in transit between user’s browser and web server. It helps defend against eavesdroppers, and assures the user of the identity of the browser being used. However, it does not prevent any of the above-mentioned attacks.

These attacks mainly take place because we can submit arbitrary input to the browser (server-side application).

* Users can interfere with all types of information being sent to server, including request parameters, cookies and HTTP headers. Any security controls on the client-side like input validation checks can easily be circumvented.
* Users can send parameters in any order, and how many ever times, or none at all, contrary to what the app expects.
* Users are not restricted to web browser only to access application. There are many tools that would make requests no browser would normally make and generate huge numbers of requests quickly to find and exploit problems.

SSL will give hacker full control on her end, and not interfere with the input being sent from client to server.

KEY PROBLEM FACTORS

Underdeveloped security awareness

Custom development

Deceptive simplicity: Use of app frameworks means that a lot of components come in built-in. This means that those without much technical knowledge can also create apps with these frameworks, so they are widely used. If a security vulnerability is found in this framework, many unrelated apps will be affected.

Rapidly evolving Threat Profile: Threats overtake ways to defend against them.

Resources and Time Constraints

Overextended Technologies: The technologies developed during WWW’s start are doing functions beyond their original requirement, leading to security flaws.

Increasing demands on Functionality: simple login pages don’t work. ‘Forget Password’, ‘Sign up with Google’ are features that may seem to increase security, but in reality they add to site’s attack surface.

New security perimeter

Earlier, servers were firewalled from external access. Now, the perimeter firewall must allow access over HTTP or HTTPS to inbound connections, and server must be allowed to connect to back-end components like databases, mainframes, financial and logistical systems. A significant part of the security perimeter is within the web apps. A single line of defective code allowing a well-crafted user input, can provide a gateway to exploit the system’s internal vulnerabilities. This also includes, third-party widgets used in apps, or apps, made with custom code integrated in the standard code.

Because of the multiple ways in which web apps receive user input and pass it to the sensitive back-ends, they are the potential gateways for many attacks, and the defenses to these attacks must be implemented in the web app itself.

Eg: through phishing by email/ Trojans in malicious websites. Also, if a hacker wants to hack in a bank to steal from personal accounts, she might have to find a vulnerability in the bank’s publicly accessible service, gain entry in the DMZ, penetrate the firewall restricting external access, decipher the protocol to gain entry in the network, map the network to find the mainframe, and then guess credentials. But this was before web apps. Now since web apps themselves are the security perimeter, it greatly assists the hackers because they can get the same outcome by modifying an account number in the hidden field of HTML form.

DMZ: In computer networks, a DMZ, or demilitarized zone, is a physical or logical subnet that separates a local area network (LAN) from other untrusted networks -- usually, the public internet. DMZs are also known as *perimeter networks* or *screened sub network*s.

Any service provided to users on the public internet should be placed in the DMZ network. External-facing servers, resources and services are usually located there. Some of the most common of these services include web, email, domain name system, and FTP and proxy servers.

|  |
| --- |
|  |

The attacks against app users are greater in magnitude and depth than those against server-side apps.

**CHAPTER 3**

**HTTP**

* Core communications protocol
* Used to access www
* Originally developed for retrieving static text based resources
* Now use is extended for supporting complex distributed apps
* Uses a message based model where client sends request message and server returns a response message.
* Connectionless.
* HTTP uses stateful TCP as transport mechanism, every exchange of request and response is an autonomous transaction using different TCP connection. A stateless system send request to server and relays response/ state back without storing any information. A stateful protocol expects the response, tracks information and resends request if no response is received.

**HTTP REQUESTS**

All http messages consist of one or more headers, each on a separate line, followed by mandatory blank line and optional message body.

|  |
| --- |
|  |

1st line of every http request has 3 parameters:

1. A verb indicating http method. Most commonly used is GET, which retrieves a resource from web server. No message body, so no further data follows blank line after message headers.
2. The requested URL: name of resource being got, along with the parameters being passed by client to resource indicated by ‘?’ as part of **query string.**

Query string=?parameters(name=value). Eg. uid=129

1. HTTP version being used. Can be 1.0 or 1.1. In 1.1, the Host request header is mandatory. Very few differences while attacking web apps.

Other points

* Referer: page from which you clicked the link to the current page
* User-Agent: provide information about the browser/ client software making the request. Historically, Mozilla is used as a browser prefix.
* Host: specifies hostname appearing in full URL being accessed.
* Cookie: used to submit additional parameters that server has issued to client.

**HTTP RESPONSES**

|  |
| --- |
|  |

Here, 1st line includes http version, numeric status code (200 🡪 success and also most commonly used), and textual “reason phrase” further describing status of response, and is of no use to current browsers.

Other points:

* Server: web server software being used and sometimes other info such as installed modules and server OS. Info may or may not be accurate.
* Set-cookie: issues the browser a further cookie, this is submitted back to cookie header in subsequent requests to server.
* Pragma header: not to store info in cache
* Expires: indicates info expired in past and so shouldn’t be cached.
* These 2 headers are used for dynamic content.
* Message body following blank line
* Content-type: indicates body of message contains html doc
* Content-length: message body length in bytes.

**HTTP METHODS**

**GET**: designed to **retrieve resources**. Can be used to send the **requested resource in the URL query string**. Can be **used to bookmark a URL** for a **dynamic resource** they can use**. URLs are visible on-screen** and on the **referrer header**. This is why **we should not include sensitive info** in the query string.

**POST**: designed to **perform actions.** Can be used to send request parameters in both **URL query string** and **body of message**. The URL can still be bookmarked and any parameters sent in message body will be **excluded from the bookmark**. The parameters are also **excluded from the logs** maintained for URL and **referrer header**. Because Post method is designed to perform action, it doesn’t automatically redirect to previous page on back button. It **gives warning to user before** doing so. This prevents user from performing action more than once. So it is advisable to use POST whenever **action is to be performed**.

**Other Methods**

**URLS**

**REST**

* Representational State Transfer
* Style of architecture for distributed systems
* Requests and responses contain representations of current state of system resources
* Core tech of WWW like HTTPS and URLs conform to REST
* URLs containing parameters within query string are REST-compliant
* REST style URLs contain parameters within file path
* Instead of query string
* Eg. <http://wahh-app.com/search?make=ford&model=pinto>
* REST style= <http://wahh-app.com/search/ford/pinto>
* Important to consider parameter styles
* While mapping an application’s content and functionality to identify key attack surface

**ENCODING SCHEMES**

* Unicode Encoding
  + Used to represent all characters of written language. 16 bit encoding is
  + %u(hex of the Unicode point of character)
  + %u2215 - /
  + UTF-8 is variable-length encoding standard that employs one or more bytes per character
  + %(byte of char in hex)
  + %e2%89%a0-> =/
  + **Of primary interest in web app hacking** to defeat input validation mechanism. An attacker may provide a system component that is not Unicode aware and use that to circumvent the filter or cause the classifying mechanism to fail to properly understand request, allowing attacker to slip malicious data past the content filter and/or cause application to route the request incorrectly.
* HTML Encoding
  + - * %quot;
      * %#34; 🡪 “
* Base64 Encoding
  + - * Processes data in blocks of 24 bits. Made into 4 chunks of 6 bits. These allow for 64 permutations.
* Hex Encoding
  + - * 1 2 3 4 5 6 7 8 9 a b c d e f