Chapter 4 - Mapping the Application

Mapping the application

The first step of attacking a web app is discovering your attack surface

Enumerating Content and functionality

- click every link and navigate to every page

Web spidering

- These tools request a page, get the links on that page, and repeat recursively
- They can also manually submit HTML forms with preset or random values
- some parse javascript in an attempt to find more URLs
- Example tools: Burp suite, WebScarab, Zed attack proxy, and CAT
- always check robots.txt - Automated limitations:
    - cant do unusual navigation mechanisms, eg dynamic menus made with js)
    - links burried in compiled client-side objects, eg flash or java applets
    - fields that require specific details will not work with random values eg, an email box may need @ and a valid email provider.
    - spiders do not follow links they have already visited, but submitting a form may make a new URL each time to show a different response, this causes an infinite loop
    - The spider must have access to a token in order to visit most of the site. Auth sessions can be break for reasons:

        - at some point, the spider will do a get request for logging out
        - the program may see this spider as malicious and terminate its session
        - if the site uses per-page tokens, the spider will likely fail
- spiders can cause massive damage, deleting accounts, dropping databases, etc.

User-Directed Spidering

- Manually navigate the site, but the proxy stores the requests and responses for later
- benefits:
    - the user knows how to follow unusual navigation mechanisms
    - the user controls data input, and can give correct info
    - the user can log in normally and let the browser manage the tokens
    - dangerous links can be avoided

Discovering Hidden Content

- Things spidering won't find but is useful:
    - Backup copies of live files can leak source code
    - backup archives containing sitemaps, code snapshots etc
    - New functionality that has been deployed on the server but not linked to the public
    - Default application functionality
    - Old versions of files
    - configs could contain sensitive data
    - Source files
    - Comments in source code talking to devs, eg “remove this func after done”
    - Log files, could contain senstive info, usernames, session tokens, URLs etc

Brute-Force techniques

- Do **not** assume 200 means found and 404 means not found
- 302 Found, if redirects to a login page it means that the content needs an account access
- 401 / 403 - cannot be accessed by your user
- 500 Internal Server Error, during content discovery usually indicates certain parameters we're not submitted

Inference from Published Content

- most apps have a naming scheme, by following their scheme and adapting your wordlist, you will find more
- look at how the devs abreviate words or phrases and use that style

Use of Public Information

- Links may have existed in the past, but not now.
- To find these you can use:
  - Search engines
  - waybackmachine
- Look for third parties that interact with your target
- Compile a list of devs, their emails and google dork for them on sites like Github or stack overflow.

Leveraging the web server

- vulns exist that allow you to map the web server itself
- default files, and CMS may lead to mapping and old vulns

Discovering Hidden Parameters

- debug=true
- other common debug parameter names, debug, test, hide, source, etc. and common values, true, yes, on, 1, etc.

Analyzing the application

Key areas to investigate:

- The applications core functionality
- Off-site links, error messages, admin/logging functions, use of redirects
- security mechanisms: session management, access controls, auth mechs, account changes
- all inputs the user can change.
- client-side techs
- server-side techs, web host, db, email, etc

Identifying entry points for user inputs

- URLS
- URL get params
- POST params
- cookies
- every HTTP header

URL File Paths

- REST-style URLs can act as query data, not just files or directories
- EG: `http://domain.com/browse/electronics/Iphone`, `electronics` and `Iphone` should be treated as inputs

Request Parameters:

- get parameters are not always in `name=value` syntax
- alternate nonstandard parameter formats:
  - `/dir/file;foo=bar&foo2=bar2`
  - `/dir/file?foo=bar$foo2=bar2`
  - `/dir/file/foo%3dbar%26foo%23dbar2`
  - `/dir/foo.bar/file`
  - `/dir/foo=bar/file`

- `/dir/file?param=file:bar`
- `/dir/file?data=%2cfoo%3ebar%3c%2ffoo%3e%3cfoo2%3ebar2%3c%2ffoo2%3e`

HTTP Headers

- lots of applications log HTTP headers such as `Referer` and `User-Agent`
- further processing may be done on `Referer` if you came from specific sites, such as web browsers
- processing on `User-Agent` is done to find which device the user is on
- Apps sometimes try to get your IP, if you use the `X-Forwarded-For` request Header you can input your own string

Out-of-Band Channels

- Webmail application that processes emails
- publishing application that can retrieve content from another server
- An intrusion detection application that gathers data and presents this using a web app interface
- Any kind of app that provides an API interface for use by non-browser user agents, eg cell phone apps

Identifying Server-side Technologies

Banner Grabbing

- Servers disclose lots of info in the `Server` header:
- other than the `Server` header, in other locations the following can be found:
  - Templates usedto build HTML pages
  - Custom HTTP Headers
  - URL Query string parameters

Http Fingerprinting

- Any info got from the server can be accidently, or purposefully wrong and misleading

File extensions

- ASP = Microsoft Active Server Pages
- ASPX = Microsoft ASP.NET
- JSP = Java Server Pages
- CFM = Cold fusion
- PHP = The PHP lang
- D2W = WebSphere
- PL = the Perl lang
- PY = python lang
- DLL = usually compiled native code (c or c++)
- NSF or NTF = Lotus Domino
- Even if the file extension is not shown it can be found through default files, error messages, etc

Directory names

- If you see any of the following directories, you can associate it with a framework:
  - servlet = Java servlets
  - pls = Oracle Application Server PL/SQL gateway
  - cfdocs or cfide = Cold Fusion
  - SilverStream = The SilverStream web server
  - WebObjects or {function}.woa = Apple WebObjects
  - rails = Ruby on Rails

Session Tokens

- If you can see these token names you can associate it with a framework:
  - `JSESSIONID` - The java platform
  - `ASPSESSIONID` - Microsoft ISS server
  - `ASP.NET_SessionId` - Microsoft ASP.NET

- `CFID/CFTOKEN` - Cold Fusion
- `PHPSESSID` - PHP

Third-Party Code components

- Many web apps use third party code for things like, shopping carts, login mechanisms, message boards, etc.
- This code can be downloaded and tested locally.

Identifying Server-Side Functionality

Dissecting Requests

- Urls can give away web frameworks, debug options, tell about db, etc.

Extrapolating Application Behaviour

- code reuse means assumptions can be made across areas of a platoform
- eg global sanitization means if you can find an exploit in one location, it may be applicable elsewhere
- Errors are often handled inconsistently, some areas handle them gracefully, others dont and return information.

Isolating Unique Application Behaviour

- If they are using a secure framework look for “bolted on” parts of the app to exploit.

Mapping the Attack Surface, check for the following:

- Client-Side validaiton - may not be done on the server
- Db interaction - SQLi
- File upload/download - Path treversal, stored XSS, SSRF
- Display of user-supplied data - xss
- Dynamic redirects - Redirection and header injection attacks
- Social networking features - username enum, stored xss
- Login - username enum, weak passwords, rate limiting checks

- Multistage login - Logic flaws
- Session state - predictable token, insecure handling
- Access controls - Horizontal/vertical priveledge escalation
- User impersonation functions - priv esc
- Use of cleartext communications - session hijacking, capture of creds, other data
- Off-site links - Leakage of query string parameters in the `Referer` header
- Interfaces to external systems - session control
- Error messages - data leak
- E-mail interaction - email/command injection
- Native code components or interacton - Buffer overflows
- Use of third-party application components - known vulns
- Identifiable web server software - common config weakness, known vulns