🥇

Web hacking

☰ Tags	checklist web
≡ Property	

Information gathering

- Search engine hacking

  🔎 Search engine investigation

  - We work exclusively, not inclusively. This means we start at the base site:target.com and exclude results we viewed. This ensures we don't miss things. We ALSO work inclusively, but it's only after working exclusively
- Webserver fingerprint + Source code investigations

  👇 Fingerprinting a webserver + finding new web applications
- Asset discovery

👚 User emulation

- Endpoint discovery
- Enumerate any admin interfaces
  - gobuster
  - ffuf
  - discover content from burp suite pro
  - Google dorks
  - Alternative server ports
  - Look for admin interface references in the source code
- Testing HTTP methods

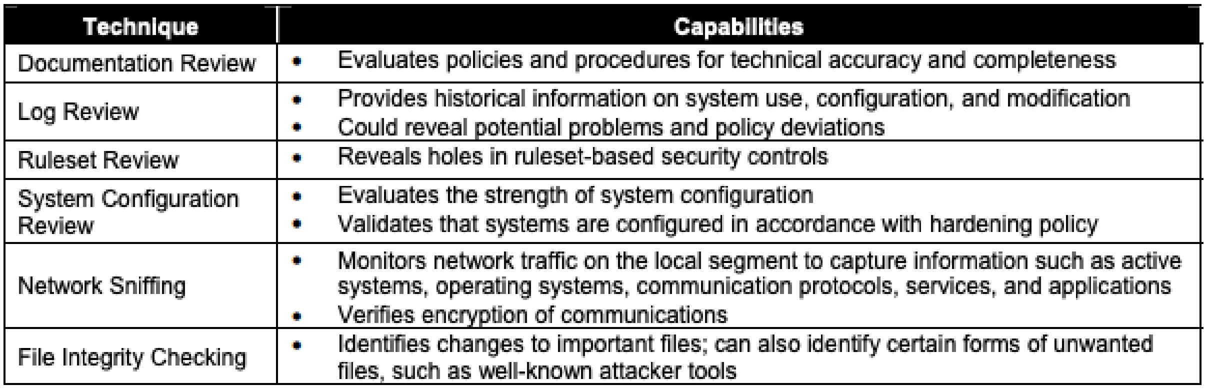
Testing HTTP methods

- Testing HTTP methods
  - If PUT is supported, try to PUT a file on the server (send OPTIONS call to find out)
    - try to GET that file
- Enumeration of errors and stack traces
- Repository recon
  - github dorking
- Fingerprint the application
- Map the application architecture
- Map out integration points
- Find
  - Data flows
  - Paths
  - Race's

Whitebox techniques

- Check in the console if anything is being logged that should not be
- Check internal logging to see if it complies to policy
  - Pay special attention to security logging
- If there is a WAF or firewall or ACL (access control list), review the ruleset
- Review the system configuration
  - Preferably a mix of automated and manual testing
- Check if a file integrity check is enabled

Table 3-1. Review Techniques



Technique	Capabilities
Documentation Review	• Evaluates policies and procedures for technical accuracy and completeness
Log Review	• Provides historical information on system use, configuration, and modification • Could reveal potential problems and policy deviations
Ruleset Review	• Reveals holes in ruleset-based security controls
System Configuration Review	• Evaluates the strength of system configuration • Validates that systems are configured in accordance with hardening policy
Network Sniffing	• Monitors network traffic on the local segment to capture information such as active systems, operating systems, communication protocols, services, and applications • Verifies encryption of communications
File Integrity Checking	• Identifies changes to important files; can also identify certain forms of unwanted files, such as well-known attacker tools

Exploits

Login system

- Test the JWT token
  - Signature Verification
    - The none signing algorithm sometimes is accepted
    - Weak HMAC Keys
    - HMAC vs Public Key Confusion

        - https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/10-Testing_JSON_Web_Tokens

- Login bypass
    - Directory brute force
    - SQLi
    - Session ID prediction
    - Remember me functionality might be able to abuse
    - XPath Injection authentication bypass
    - LDAP Injection authentication bypass
    - https://book.hacktricks.xyz/pentesting-web/login-bypass/sql-login-bypass
- Username enumeration
    - Does system return different response if username exists?
    - Does the system take longer to process if username is correct?
- Credentials transported over HTTP
- Default credentials
- Issues in the registration process
    - Tokens sent over plaintext?
    - DoS by entering too many characters
    - Register a user with XSS attack vector in every input field, use for further testing
- Weak password systems
- Test password reset systems
    - Add second email parameter with email of attacker
    - Any tokens sent over HTTP
    - Weak predictable tokens

    - Is the user forced to reauthenticate
- SessionID in URL
- Logout should invalidate session tokens
- Validate that a hard session timeout exists.
- Weak lockout
- Is there any alternative login system
    - Oauth
    - Mobile login
    - Token login with weak tokens
- Testing for session puzzling
    - https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/08-Testing_for_Session_Puzzling
- Session hijacking
    - Request from attackers website to victims bank for example to login. The bank will possibly return session vars if bad config. This leads to attackers owning session vars now.
    - https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/09-Testing_for_Session_Hijacking

Input validation

- XSS (See XSS topic)
- SQLi
- XXE
- Command injection
- SSTI/CSTI

    - Insert ${7*7} into every field you see, if it resolves, investigate further
- SSRF
    - Add burp collaborator URL in everywhere that URL resolves
- HTTP parameter polution
    - https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/04-Testing_for_HTTP_Parameter_Pollution
- LDAP injection
- SSI (server side includes injection)
- XPath injection
- Clickjacking
- GraphQL testing

General exploits

- RFI/LFI
- BAC
- Insecure session management
- Subdomain takeover
- Check if user role is user controllable (Can you make yourself admin)
- Authentication bypass
- Default accounts
- Password quality checks
    - No blank passwords allowed
    - Ensure strong passwords are required
- Test the cookies attributes
    - Sensitive cookies should have secure and httponly flag
    - Domain and path need to be set right

    - Expires in timely manner
    - SameSite Attribute
- CSRF testing
    - Only on sensitive functions, not on login/logout
- File upload testing
    - Uploading of malicious content
    - File upload restrictions
        - Changing mimetype
        - Using nullbytes
- Test integration points for overextended priviledges
- Test if the browser caches sensitive information
    - Use the back button after logout timer
    - Click around after login timer
    - Check cache headers on sensitive pages
- There should not be any weak encryption used anywhere
    - Search for the following keywords to identify use of weak algorithms: MD4, MD5, RC4, RC2, DES, Blowfish, SHA-1, ECB

Business logic flaws

- Data validation
    - example may be if I use my credit card at multiple locations very quickly it may be possible to exceed my limit if the systems are basing decisions on last night's data.
    - Identify data injection points.
    - Validate that all checks are occurring on the back end and can't be bypassed.
    - Attempt to break the format of the expected data and analyze how the application is handling it.

- Test for hidden parameters that you can change with impact.
    - For example changing account type from consumer to business
- Check for things that are only hidden in the front-end
- Check for disabled fields that are only front-end disabled
- Integrity checks
    - Review the project documentation for components of the system that move, store, or handle data.
    - Determine what type of data is logically acceptable by the component and what types the system should guard against.
    - Determine who should be allowed to modify or read that data in each component.
    - Attempt to insert, update, or delete data values used by each component that should not be allowed per the business logic workflow.
- Test functions that can only be used a limited amount of times
    - For example a coupon code that you should only be applying one time but that's just a front-end check
- If something gets added to account and should be withdrawn again, check if it is.
    - For example if you order an item but cancel the order, your loyatee points should go down as well.

XSS

General

- All XSS must also be viewed via an admin interface if that is available
- '"><img src=x> into every field or your own attack vector as soon as you register to passively test a little bit for HTMLi, HTML tag injection and JS injection
- Blind XSS is just stored XSS that the user can not view the result of
- Cheat sheet available

Reflected XSS

- Design step
    - Find a foothold
        - Identify ALL the user controlleable parameters
            - HTTP parameters
            - POST parameters
            - POST data
            - Hidden fields
            - Predefined radio or selection value
        - Identify where a value is reflected on the page
            - Pay attention to the context
                - JS context
                    - \n (new line)
                    - \r (carriage return)
                    - ' (apostrophe or single quote)
                    - " (double quote)
                    - \ (backslash)
                    - \uXXXX (unicode values)
                - HTML injection
                    - > (greater than)
                    - < (less than)
                    - & (ampersand)
                    - ' (apostrophe or single quote)
                    - " (double quote)
                - HTML tag attribute
                    - > (greater than)

          - ' (apostrophe or single quote)
          - " (double quote)
          - ` (backtick)
        - ...
    - Craft an attack vector for the specific context
  - Attacker creates and tests an offending URI
    - Sometimes filters are in place
    - Figure out what filters exist
    - See if we can get around them
    - **Sometimes <script> might be filtered**
      - **But %3cscript%3e not**
        where
        %3c = <

        %3e = >
- Social engineering step
  - Attacker gets victim to click link and execute XSS
- Execution step
  - Make sure the XSS has impact and be realistic
  - Specific exploit gets executed
    - I.E. cookie stealing
    - I.E. executing JS function
    - I.E. Stealing data on a page

Stored XSS

- Design step
  - Find a foothold
    - Identify the stored input and where it is reflected in the client side

  - Hidden fields
  - POST parameters
  - headers
  - cookies
  - ...
  - Tester must define all user controller variables and parameters
- Identify where a value is reflected on the page
  - Pay attention to the context
    - JS context
      - \n (new line)
      - \r (carriage return)
      - ' (apostrophe or single quote)
      - " (double quote)
      - \ (backslash)
      - \uXXXX (unicode values)
    - HTML injection
      - > (greater than)
      - < (less than)
      - & (ampersand)
      - ' (apostrophe or single quote)
      - " (double quote)
    - HTML tag attribute
      - > (greater than)
      - ' (apostrophe or single quote)
      - " (double quote)
      - ` (backtick)

        - ...
    - Craft an attack vector for the specific context
  - Attacker creates and tests an offending URI
    - Sometimes filters are in place
    - Figure out what filters exist
    - See if we can get around them
    - **Sometimes <script> might be filtered**
      - **But %3cscript%3e not**
        where
        %3c = <

        %3e = >
  - Social engineering step
    - Attacker gets victim to click link and execute XSS
  - Execution step
    - Make sure the XSS has impact and be realistic
    - Specific exploit gets executed
      - I.E. cookie stealing
      - I.E. executing JS function
      - I.E. Stealing data on a page

DOM XSS

See DOM XSS

Extras

- Contact form
  - Rate limiting (spammer prevention)
  - CAPTCHA bypass (spammer prevention)

- Application level DoS
  - Enter big input one time
  - Enter decent size input many times
  - Application lockout should apply on sensitive areas such as login
  - Application lockout may not be triggered by users for others

🔴 Enumeration of errors and stack traces