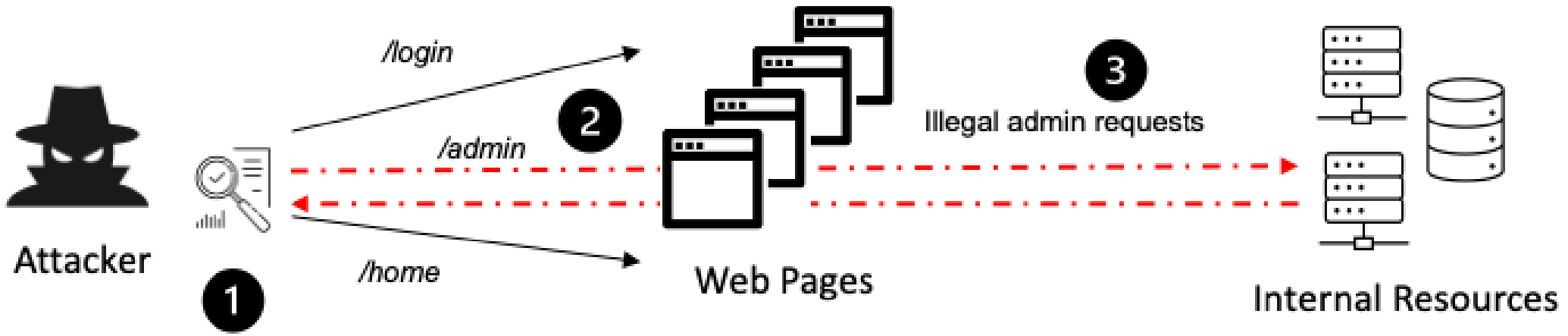
OWASP Top 10

> Enumeration of the top 10 most critical security risks to web applications.

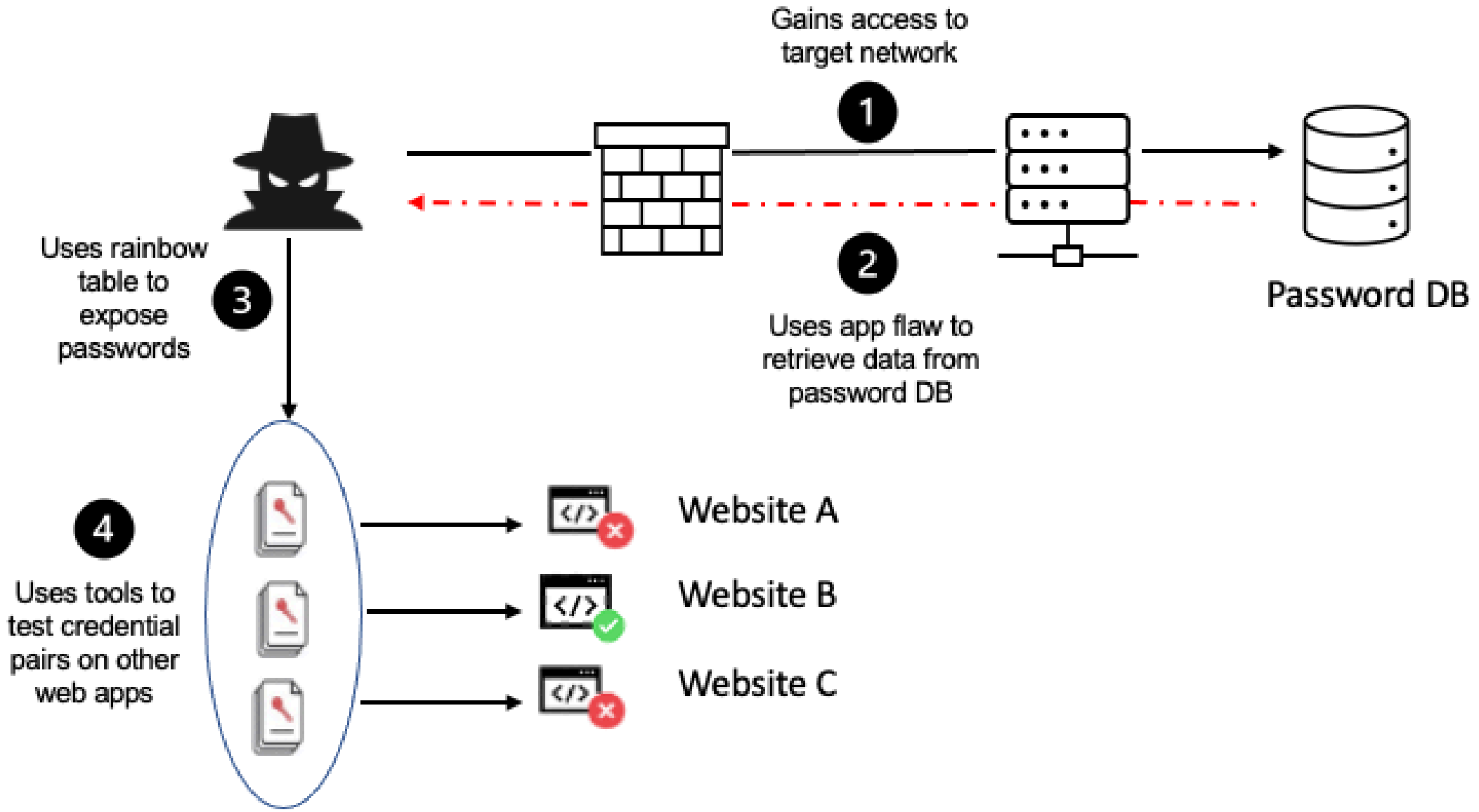
1. Broken Access Control

- **Description:** attacker was able to access a resource or perform an action that they was not supposed to have access to, due to the access control mechanisms not implemented correctly.
- **Example:** modification of a URL parameter to gain unauthorized access to an object (IDOR).
- **Countermeasures:** deny by default policy, access control checks at all layers, code reviews, implementing ownership model, disabling unnecessary functionalities, API rate limit, automatic testing and token validation.


/login
/admin
/home
2
3
Illegal admin requests
Attacker
1
Web Pages
Internal Resources

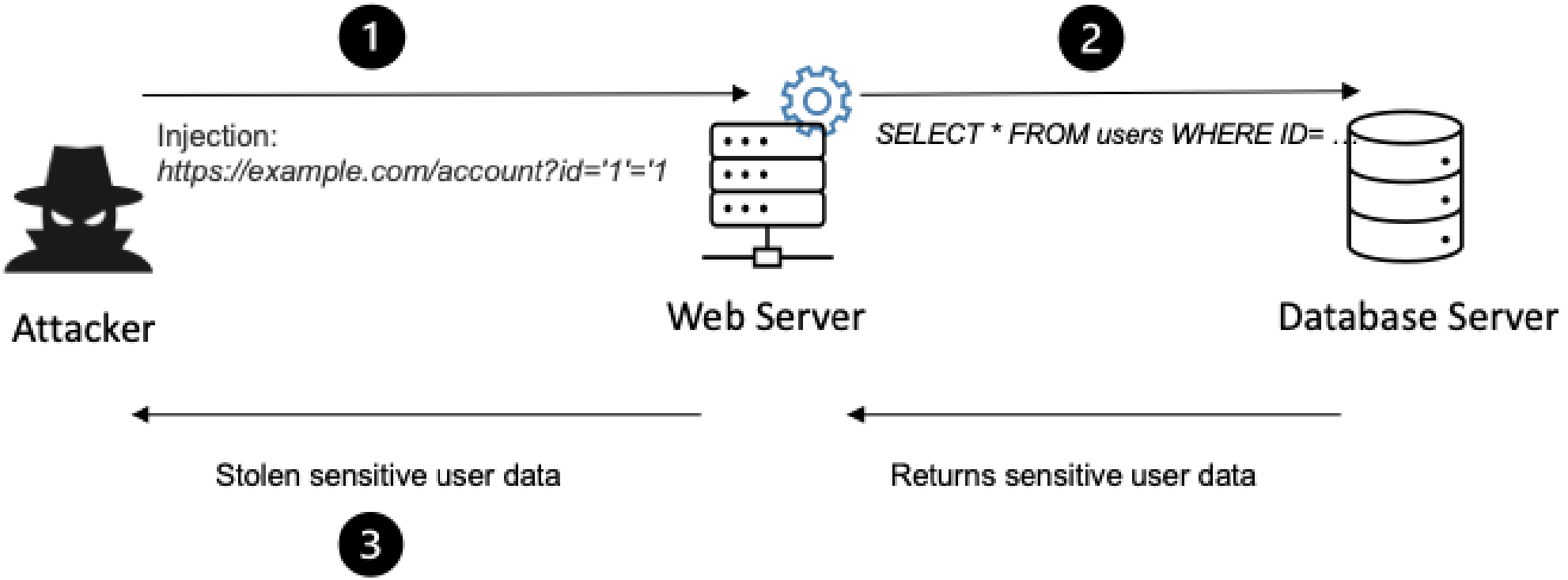
2. Cryptographic Failures

- **Description:** attacker was able to access systems/data due to the cryptographic techniques not being used correctly.
- **Examples:** weak encryption algorithms, weak/unsalted hashes, weak Pseudo-Random Number Generators (PRNG), weak/compromised/reused cryptographic keys.
- **Countermeasures:** responsible use of cryptographic functions - not rolling your own crypto, using strong keys, key rotation, secure coding, and code reviews.


Gains access to target network
1
Uses rainbow table to expose passwords
3
2
Uses app flaw to retrieve data from password DB
Password DB
4
Uses tools to test credential pairs on other web apps
Website A
Website B
Website C

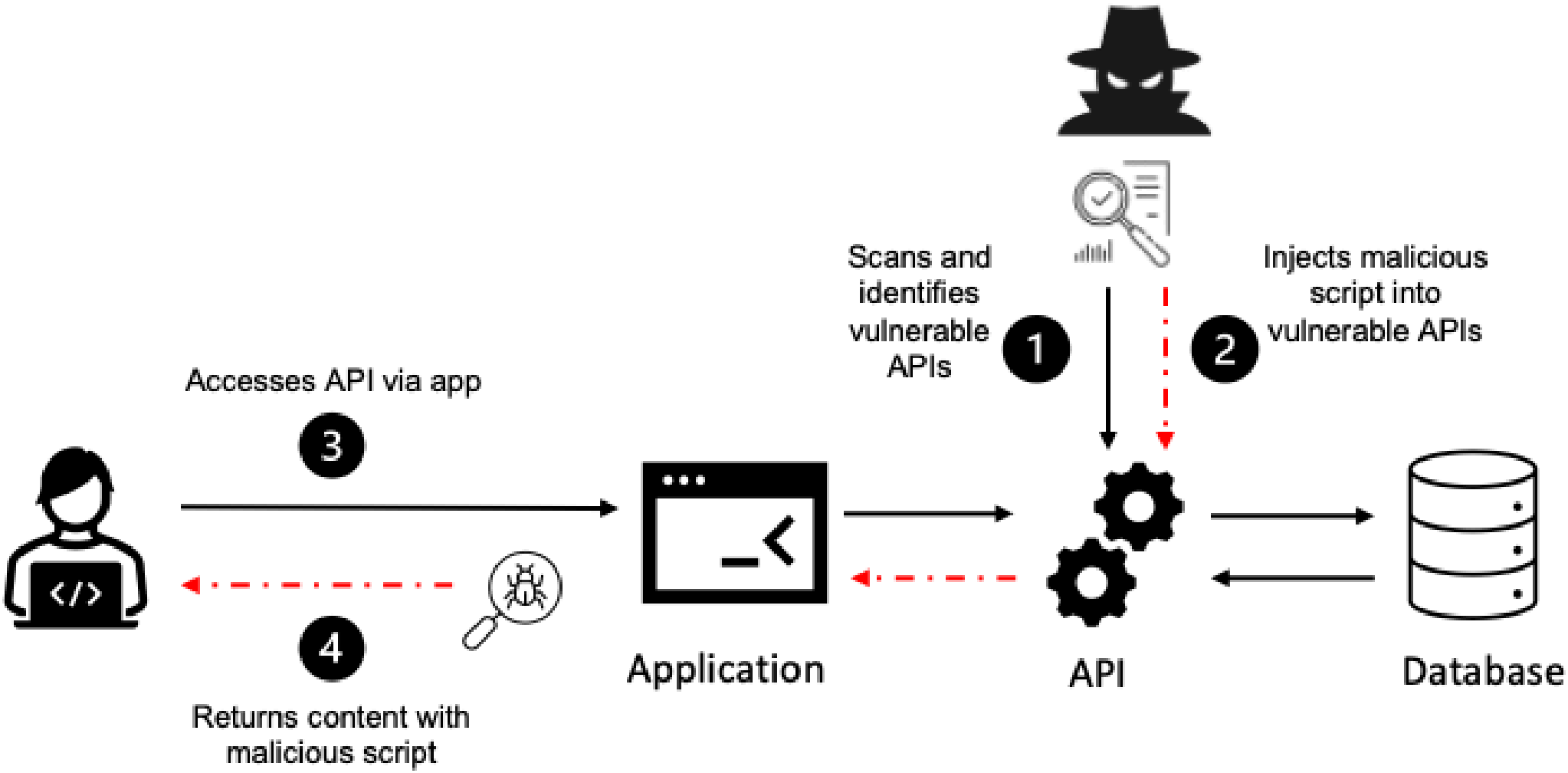
3. Injection

- **Description:** attacker was able to send malicious input to a vulnerable system, the input was processed, allowing the attacker to change the course of execution and manipulate the system into doing an unintended behavior
- **Examples:** leaking/corrupting/destroying sensitive data with SQL/NoSQL injection or other query languages, OS command injection, or CRLF injection to inject HTTP request headers. Payloads can be injected in request line, headers, or body.
- **Countermeasures:** properly sanitize input by escaping special characters and s or limiting input length, allow using commands only from a whitelist, no direct substitution of user input into code (e.g., process through an isolated API first), handle exceptions and make sure they don't expose sensitive data.


1
2
Injection:
https://example.com/account?id='1'='1
SELECT * FROM users WHERE ID= ..
Attacker
Web Server
Database Server
Stolen sensitive user data
Returns sensitive user data
3

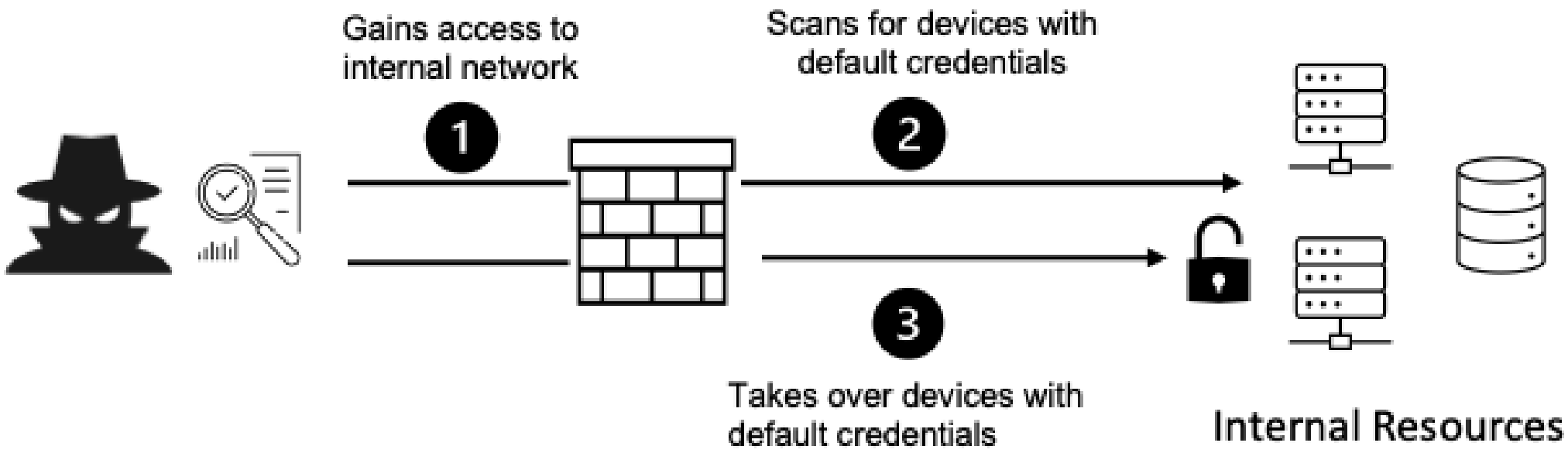
4. Insecure Design

- **Description:** attacker was able to hack into a system because it was vulnerable by design (there was an architectural flaw in the way the system is designed), there was no error from the admin side or the developers who implemented it.
- **Examples:** insufficient elaboration at the stages of analysis and planning didn't define how to handle certain use cases of the application which lead to vulnerabilities.
- **Countermeasures:** implement secure software development lifecycle, use safe design patterns, use threat modeling, data validation at all levels, write tests, limit resource consumption.


Scans and identifies vulnerable APIs
1
2
Injects malicious script into vulnerable APIs
Accesses API via app
3
4
Returns content with malicious script
Application
API
Database

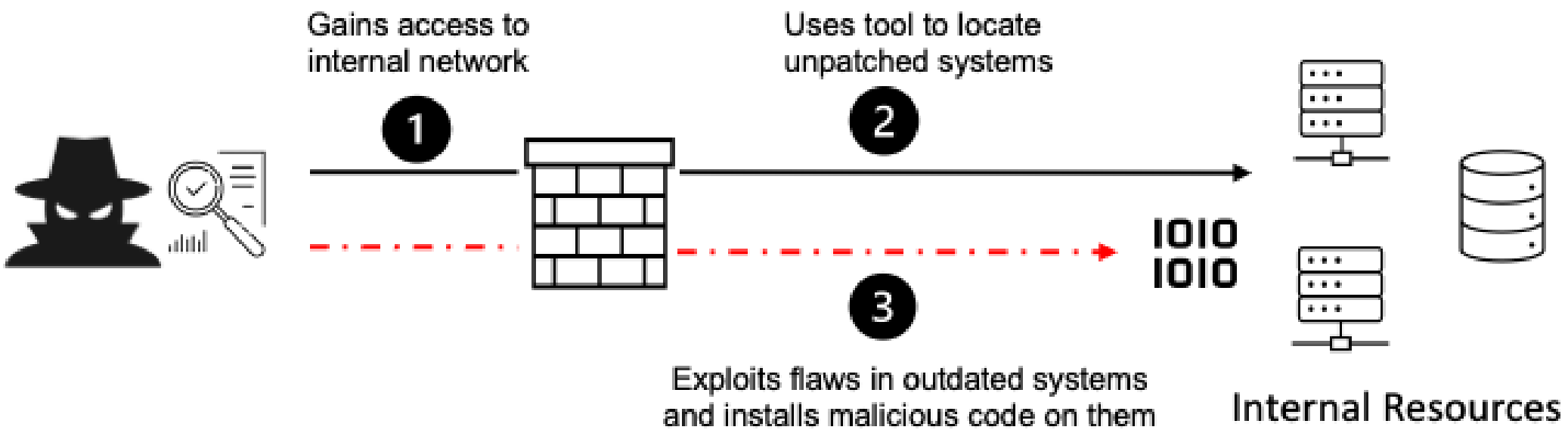
5. Security Misconfiguration

- **Description:** attacker was able to hack into a system because of an error from admin side, as opposed to the previous point.
- **Example:** https not enforced in server configuration, unnecessary features are left enabled, default credentials are not changed.
- **Countermeasures:** automatic configuration management and configuration verification that will reduce the possibility of human errors, removing unnecessary features and frameworks, the use of security headers and directives.


Gains access to internal network
1
Scans for devices with default credentials
2
3
Takes over devices with default credentials
Internal Resources

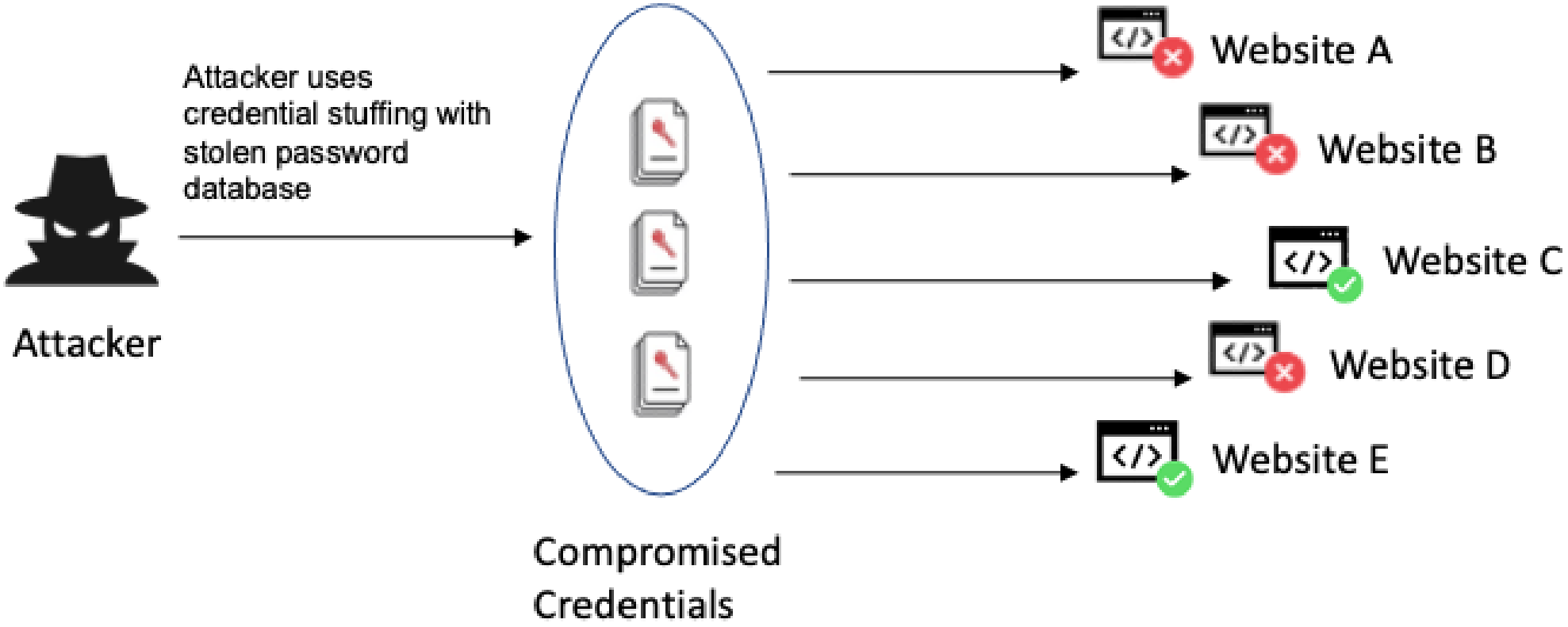
6. Vulnerable and Outdated Components

- **Description:** attacker was able to exploit a vulnerability in a certain component (OS, DBMS, API), or a dependency (software library) used by the target system, due to the component being vulnerable by design or not being updated.
- **Example:** the usage of old version of HTTP webserver.
- **Countermeasures:** regular scanning and updating of application dependencies and hosting systems, removing unused dependencies, obtaining dependencies from official trusted sources, proper monitoring.


Gains access to
internal network
1
Uses tool to locate
unpatched systems
2
1010
1010
3
Exploits flaws in outdated systems
and installs malicious code on them
Internal Resources

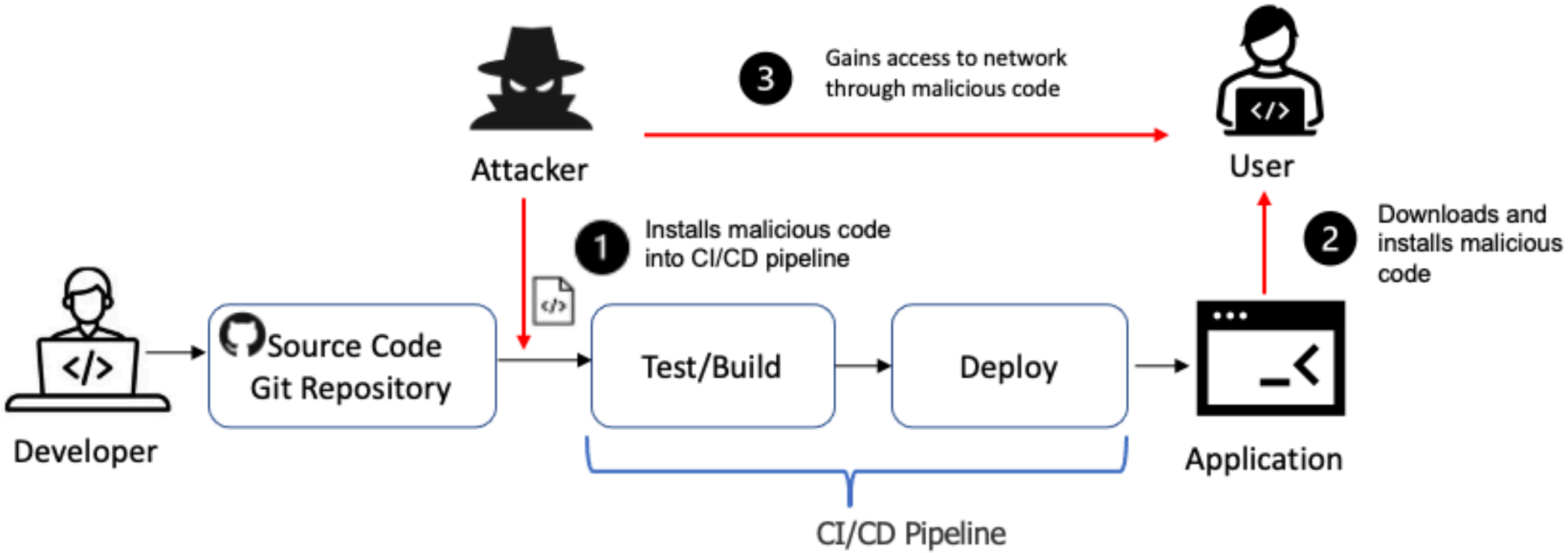
7. Identification and Authentication Failures

- **Description:** attacker stole user credentials or was authenticated into a system they were not supposed to access due to the system not implementing authentication correctly.
- **Examples:** the system allows weak passwords, doesn't block automated attacks, doesn't use 2FA, doesn't specify session timeout, or uses an ineffective credential recovery process.
- **Countermeasures:** enforce strong passwords, 2FA, block bruteforce attacks (e.g., CAPTCHA), use strong session manager and specify session timeout.


Attacker uses credential stuffing with stolen password database
Attacker
Compromised Credentials
Website A
Website B
Website C
Website D
Website E

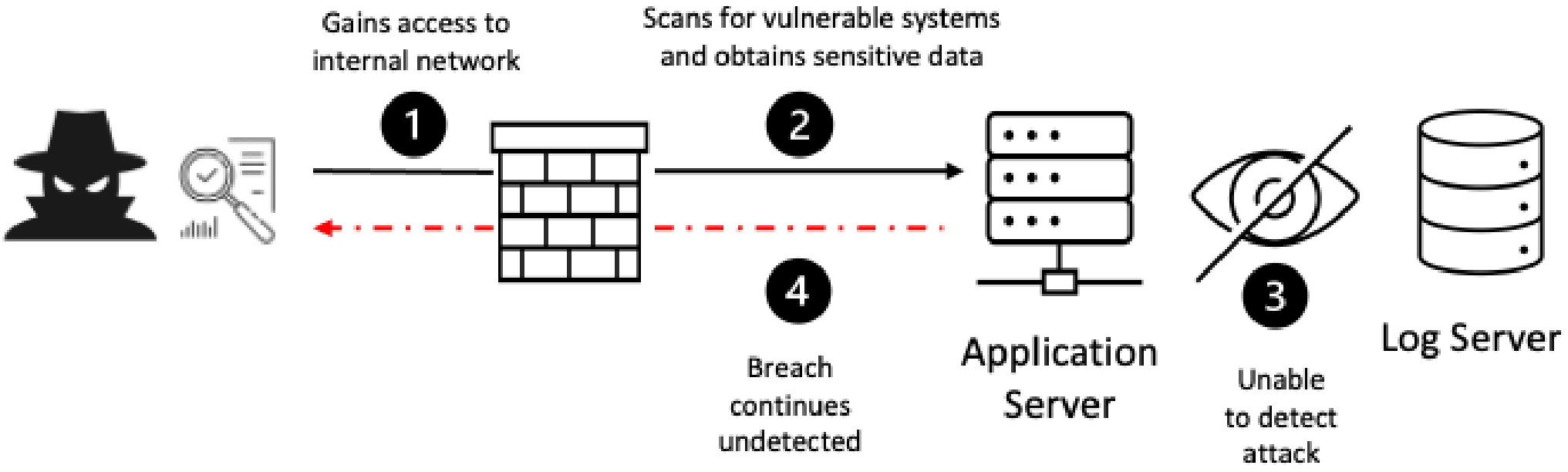
8. Software and Data Integrity Failures

- **Description:** attacker was able to modify data or software packages due to the system not verifying data integrity properly.
- **Example:** system doesn't validate digital signatures of certificates, or checksums for received software packages; a system that doesn't do these checks can be tricked into de-serializing a malicious object.
- **Countermeasures:** use digital signatures, ensure that packages are grabbed from trusted resources through a secure connection, use a tool for dependency checks.


Attacker
3
Gains access to network through malicious code
User
1
Installs malicious code into CI/CD pipeline
2
Downloads and installs malicious code
Developer
Source Code Git Repository
Test/Build
Deploy
Application
CI/CD Pipeline

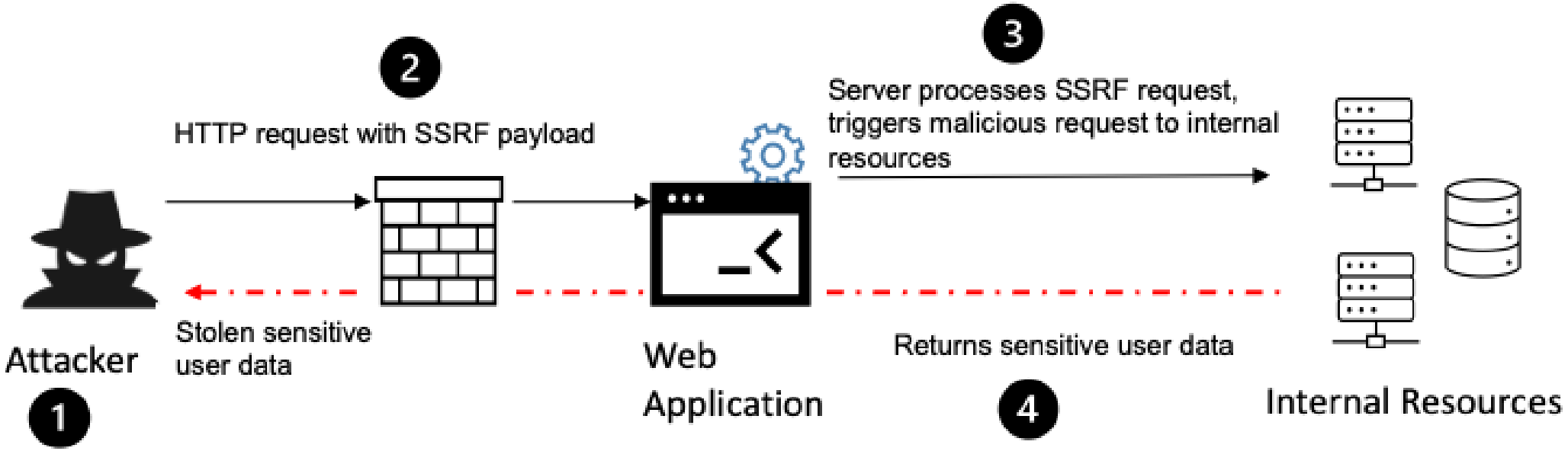
9. Security Logging and Monitoring Failures

- **Description:** attacker was able to hack into a system undetected because the system lacks extensive and explanatory logging or the logs were not inspected/monitored carefully, or the alerting mechanism was blind to the attack or very slow to report.
- **Example:** Data can be stolen from a system deployed on the cloud, it will take time for the provider to notify the organization.
- **Countermeasures:** implementing the necessary logging, auditing, monitoring, and alerting systems, having a defined incident response process.


Gains access to internal network
1
Scans for vulnerable systems and obtains sensitive data
2
3
4
Application Server
Log Server
Unable to detect attack
Breach continues undetected

10. Server-Side Request Forgery

- **Description:** attacker was able to abuse the server to access or modify internal resources (or fetch data from untrusted sources), by deceiving the server into communicating with a user-supplied URL.
- **Example:** the system takes user-supplied URLs and grabs resources from them, an attacker can supply `localhost` or some private IP, this will make the server expose internal data to the attacker
- **Countermeasures:**
  - **At the network layer:** internal servers should be separated from the ones fetching remote resource, firewalls should deny by default (only allow essential intranet traffic based on known data lifecycle).
  - **At the application layer:** sanitize and validate all client-supplied input data, fetch resources only from trusted URLs through HTTPS.


2
HTTP request with SSRF payload
3
Server processes SSRF request, triggers malicious request to internal resources
Attacker
Stolen sensitive user data
Web Application
Returns sensitive user data
Internal Resources
1
4