



The **security** handbook

Documented standards and best practices for secure
product development

V2.5

June 15, 2021



Know how to address critical security issues

Molson Coors. Equifax. eBay. LinkedIn. News of data breaches, denial of service attacks, loss of service, and compromised information constantly make headlines. From SaaS platforms used by Fortune 500 companies to private healthcare patient data run through DICOM, data leaks happen, causing irreparable damage to enterprises and users. There's no shortage of reasons for enterprises to have diligent, secure coding best practices in place to remedy or avoid the risk of a security breach.

At Devbridge, our team uses a simple yet effective approach to document and implement secure coding best practices. The tactics are ingrained in the agile software development life-cycle and embedded in every engagement.

We start with education, seeking out industry standards and best practices from open standards organizations. Our team leverages the learnings from OWASP (Open Web Application Security Project) and the SANS (SysAdmin, Audit, Network, Security) Institute who set standards and regularly updates documentation based on the types of attacks occurring across industries. Then, enforcing these principles within SDLC through code reviews and DevOps automation, our engineer team ensures code compliance and high-quality software delivery.

Our team created a downloadable Secure Handbook to document our best practices. The documentation sets standards for our internal information security program and helps keep our team accountable. In the spirit of transparency, we are sharing our documented best practices to help our clients better understand the top security risks and implement controls in your software engineering organization.



**Security is a skill,
not a separate department.**

– AppSec Community

Treat security as a shared responsibility

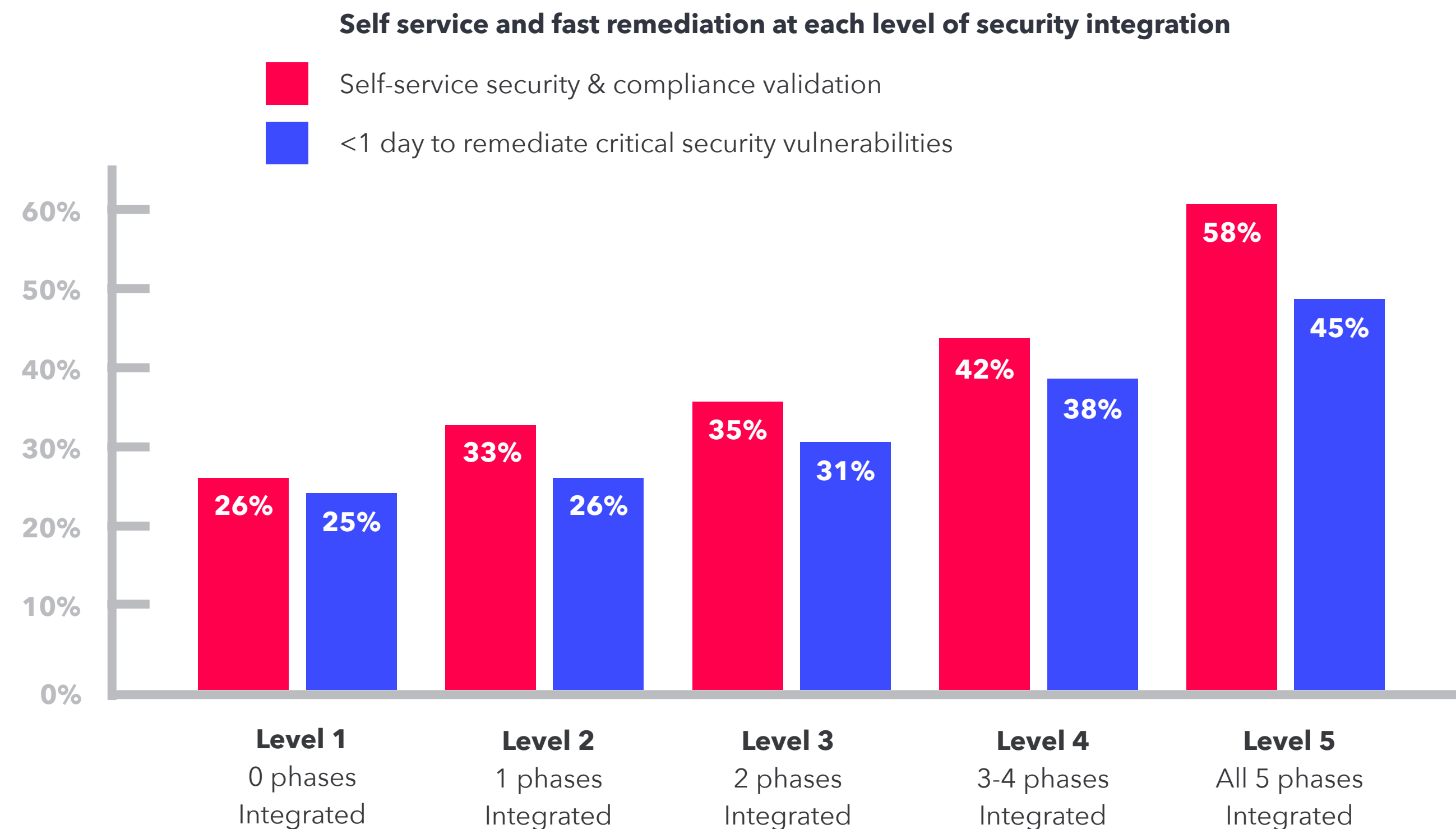
Integrating security into the software delivery lifecycle fosters an environment in which delivery teams begin to view secure code as a shared responsibility.

The benefits of shared ownership

- The team pays closer attention to potential security issues.
- The team becomes more diligent in following policy and process at all stages of the lifecycle.
- The team is open to halting deployment when a security issue warrants it.



2020 State of DevOps Report
<https://puppet.com/resources/report/state-of-devops-report/>





The Open Web Application Security Project

OWASP is a nonprofit organization dedicated to improving the security of software with hundreds of local chapters and tens of thousands of members worldwide.

As the source for developers and technologists to secure the web, the foundation offers:

- Application security tools and standards.
- Complete books on application security testing and secure code.
- Development and secure code review.
- Presentations and videos.
- Cheat sheets.
- Standard security controls and libraries.
- Local chapters worldwide.
- Cutting-edge research.
- Global conferences.
- Mailing lists.



The OWASP Top 10 List



OWASP actively maintains a running list of the most critical security risks to web applications.

Use the OWASP Top 10 as a guide to build secure code and minimize the risk of:

- 1. Injection
- 2. Broken authentication
- 3. Sensitive data exposure
- 4. XML external entities
- 5. Broken access control
- 6. Security misconfiguration
- 7. Cross-site scripting
- 8. Insecure deserialization
- 9. Using components with known vulnerabilities
- 10. Insufficient logging & monitoring

OWASP Top 10	2003	2004	2007	2010	2013	2017 RC1	2017 RC2
Unvalidated Input	A1	A1					
Buffer Overflows	A5	A5					
Denial of Service		A9					
Injection	A6	A6	A2	A1	A1	A1	A1
Cross Site Scripting (XSS)	A4	A4	A1	A2	A3	A3	A7
Broken Authentication and Session Management	A3	A3	A7	A3	A2	A2	A2
Insecure Direct Object Reference		A2	A4	A4	A4	A4	A5
Cross Site Request Forgery (CSRF)			A5	A5	A8	A8	
Security Misconfiguration	A10	A10		A6	A5	A5	A6
Broken Access Control	A2	A2	A10	A8	A4	A4	A5
Insufficient Attack Protection					A7	A7	
Unvalidated Redirects and Forwards				A10			
Information Leakage and Improper Error Handling	A7	A7	A6	A6			
Malicious File Execution			A3	A6			
Sensitive Data Exposure	A8	A8	A8	A7	A6	A6	A3
Insecure Communications		A10	A9	A9			
Remote Administration Flaws	A9						
Using Known Vulnerable Components					A9	A9	A9
Unprotected APIs						A10	



The OWASP Top 10

Understanding the risks
and how to remedy them



1. Injection

Untrusted data is sent as part of a command or query.

What it is

Websites and apps occasionally run commands on the underlying database or operating system to add or delete data, execute a script, or start other apps. If unverified, inputs are added to a command string or a database command. As a result, attackers can launch commands at will to take control of a server, device, or data.

How it works

If a website, app, or device incorporates a user input within a command, an attacker can insert a " command directly into the corresponding input. If that input is not verified, an attacker then uses " and runs their commands freely.

Why it's bad

Once attackers gain access to make commands, they can control your website, apps, and data.

Countermeasures

- Avoid string interpretation. Use well-known ORMs or APIs to retrieve data.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.
- Use a positive or allow list " server-side input validation (e.g., allow only some HTML tags in a custom WYSIWYG editor and strip down everything else).
- For any dynamic queries, remove special characters using specific syntax for the interpreter. Avoid dynamic queries. Instead, use ORMs.



1. SQLi injection example

The hacker uses SQLi injection to exploit a non-validated input on a web form or http get where the query parameters are exposed on the URI that can be used to perform malicious activity.

Example A1 showcases good tactics.

The URL and query are used to bring back results data based on the customer ID being '1' or what the user enters in the form on the web page. The result is an account view for a customer ID of 1.

Example B1 exhibits poor tactics.

The URL query parameter drops a table because the form data or URL query parameters are not validated prior to sending to the database. A secondary SQL command is added to "DROP TABLE FROM USERS" by appending a semicolon and a SQL command to the .

CODE SNIPPET:

```
String query = "SELET * FROM accounts WHERE custID=" + request.getParameter("id");
```

Example A1:

<https://www.example.com/app/accountView?id=1>

```
SELECT * FROM accounts WHERE custID = '1'
```

Example B1:

<https://www.example.com/app/accountView?id=1;DROP TABLE USERS>

```
SELECT * FROM accounts WHERE custID = '1';DROP TABLE USERS
```



2. Broken authentication

Authentication and session management functions are implemented incorrectly.

What it is

Authentication is the process of ensuring it's in fact, the user accessing their accounts and data (e.g., a username and password combination). Authentication grows more complex as a site, app, or device becomes bigger, broader, and more connected to other sites, apps, or devices.

How it works

Passwords can be easily guessed or stolen if left unprotected. Attackers seek vulnerabilities where user credentials or sessions lack adequate protections. User data session IDs, when exposed via URL, are available in router logs and prone to URL rewrite attacks. Credential stuffing can be performed by utilizing leaked password databases from other sites. See <https://haveibeenpwned.com/> for a catalog of recent user account breaches.

Why it's bad

If attackers can hijack user's or administrator's session, they have access to everything available within that account, from data to account control.

Countermeasures

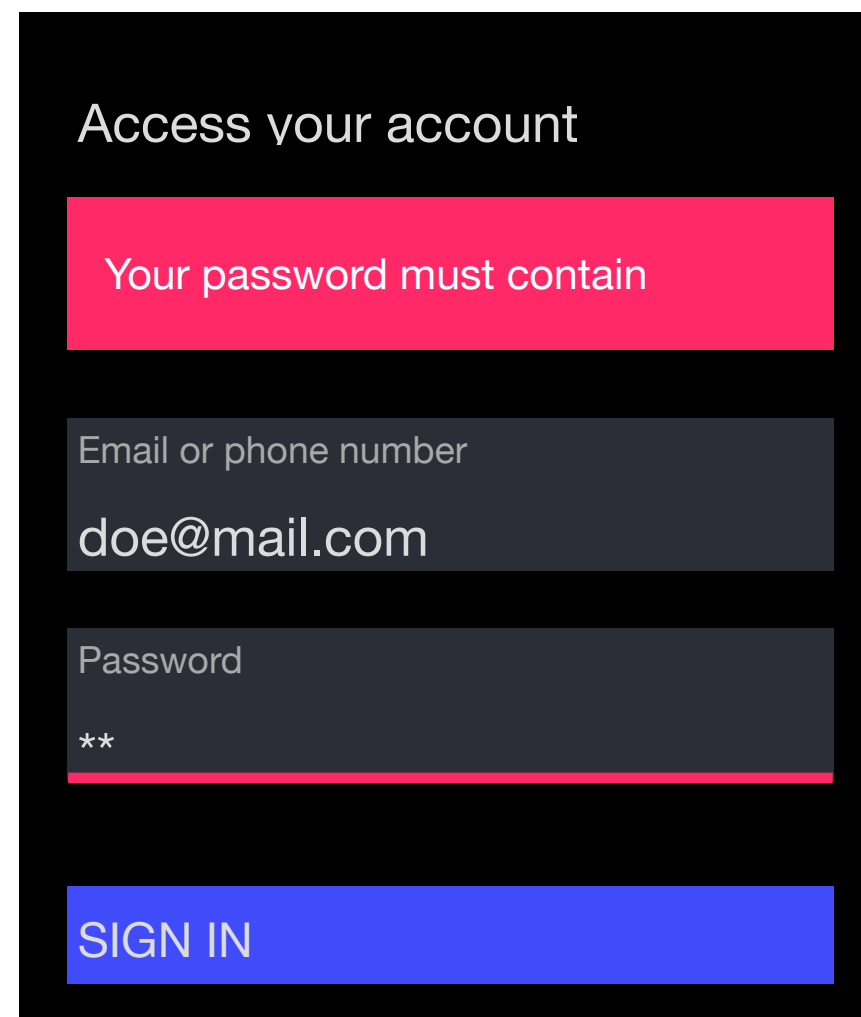
- Where possible, implement multi-factor authentication (MFA).
- Do not ship or deploy with any default (admin) credentials.
- Align password length, complexity, and rotation policies with modern password policies and algorithms.
- Implement weak password checks, such as testing new or changed passwords against a list of the top 10,000 worst passwords.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks using the same messages for all outcomes.
- Limit or increasingly delay subsequent failed login attempts. Use CAPTCHA.
- Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Web application session IDs or tokens should not be in the URL. They should be securely stored and invalidated after logout, idle, and absolute timeouts.

2. Broken authentication example

The hacker seeks out ways to modify user credentials to access the user's account.

Scenario 1

The account requires explicit or detailed password requirements, which explain the requirements that need to be met to circumvent the password.

A login form titled "Access your account" with a red error message: "Your password must contain". Below the message are two input fields: "Email or phone number" with the value "doe@mail.com" and "Password" with the value "**". At the bottom is a blue "SIGN IN" button.

Access your account

Your password must contain

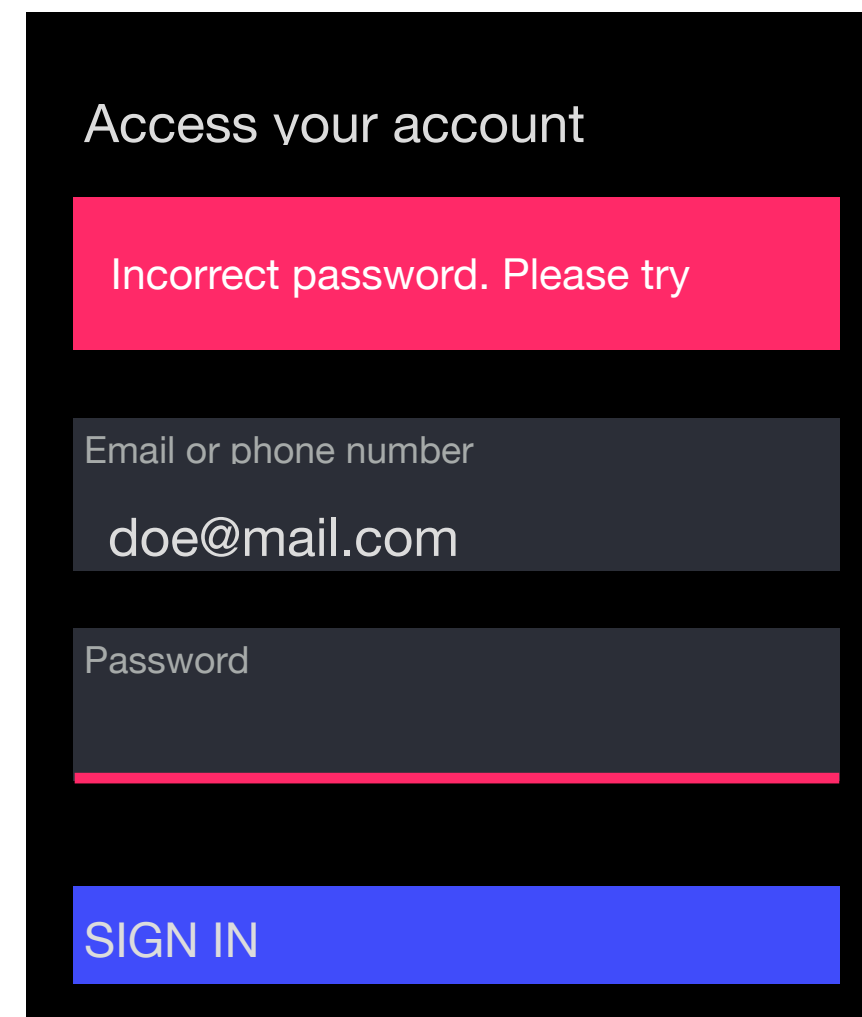
Email or phone number
doe@mail.com

Password
**

SIGN IN

Scenario 2

The account alerts the user of an invalid password entry, which inadvertently indicates that the user account is valid.

A login form titled "Access your account" with a red error message: "Incorrect password. Please try". Below the message are two input fields: "Email or phone number" with the value "doe@mail.com" and "Password". At the bottom is a blue "SIGN IN" button.

Access your account

Incorrect password. Please try

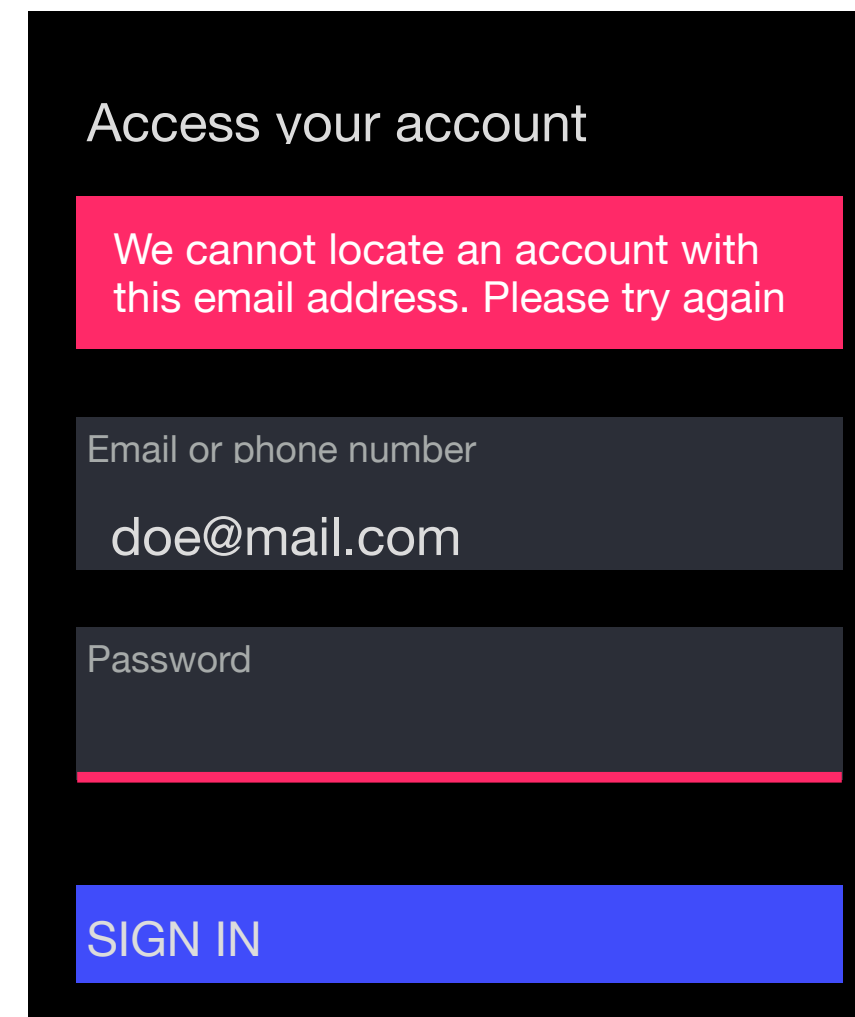
Email or phone number
doe@mail.com

Password

SIGN IN

Scenario 3

An invalid email address alerts the person entering login credentials the password is correct.

A login form titled "Access your account" with a red error message: "We cannot locate an account with this email address. Please try again". Below the message are two input fields: "Email or phone number" with the value "doe@mail.com" and "Password". At the bottom is a blue "SIGN IN" button.

Access your account

We cannot locate an account with this email address. Please try again

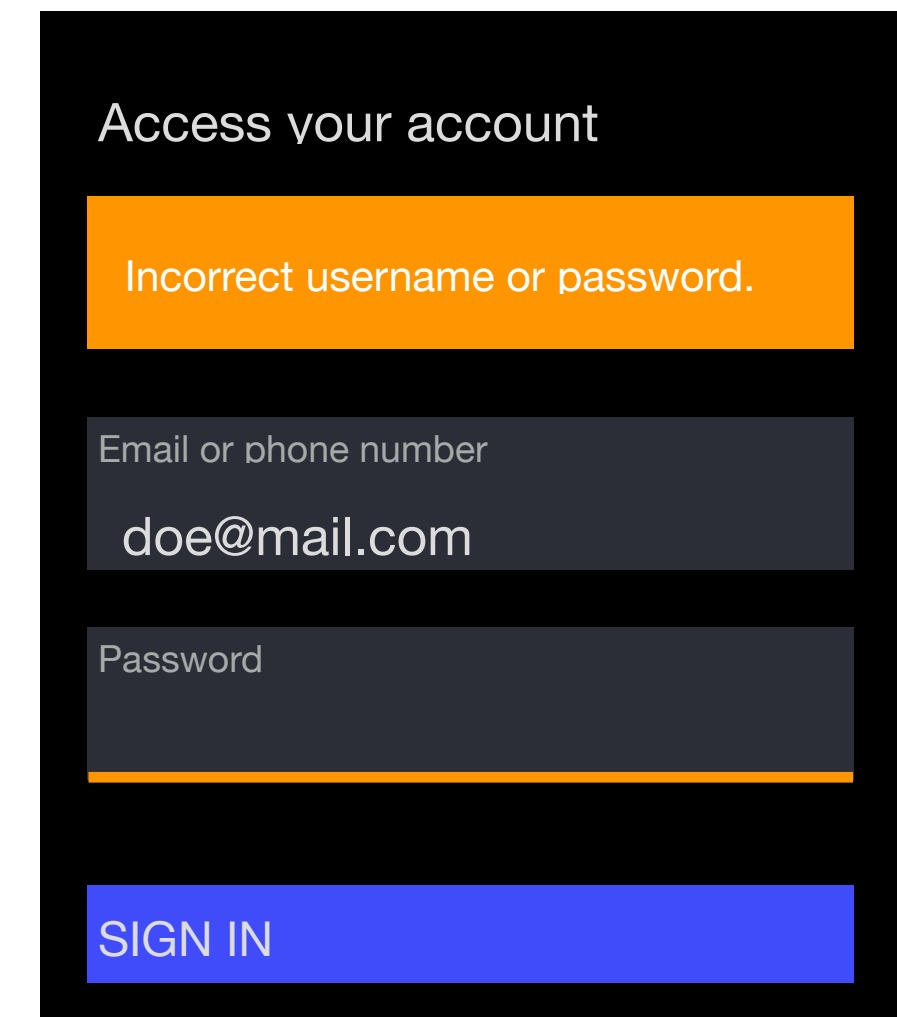
Email or phone number
doe@mail.com

Password

SIGN IN

Scenario 4

A simple, secure error message should not allude to which input fails to authenticate or why to maintain authentication security.

A login form titled "Access your account" with an orange error message: "Incorrect username or password.". Below the message are two input fields: "Email or phone number" with the value "doe@mail.com" and "Password". At the bottom is a blue "SIGN IN" button.

Access your account

Incorrect username or password.

Email or phone number
doe@mail.com

Password

SIGN IN



3. Sensitive data exposure

Financial or personal data transfers are not handled securely.

What it is

Sensitive data, such as credit card numbers, health data, or passwords, should have extra protection given the potential damage that could occur should the information fall into the wrong hands. While there are regulations and standards designed to protect sensitive data, data stored, transmitted, or protected by inadequate methods can be exposed to attackers.

How it works

Data stored or transferred as plain text, using older/weaker encryption, or can be easily decrypted gives attackers the opportunity to access to exploit the information.

Why it's bad

Once an attacker obtains passwords, credit card numbers, personal information, health records, and business secrets, they can do real damage to the person or even the company that leaked the data.

Countermeasures

- Identify and secure data deemed sensitive according to privacy laws, regulatory requirements, or business needs.
- Don't store sensitive data unnecessarily. Discard it as soon as possible, use PCI DSS compliant tokenization, or try truncation. Data that is not retained cannot be stolen.
- Instead of implementing payment gateway solutions, explore options to integrate with PCI compliant third-party payment providers.
- Do not log excessive, sensitive data in log files.
- Encrypt all data in transit with secure protocols, such as TLS. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place. Use proper key management.
- Verify the effectiveness of configuration and settings via manual performance tests and DAST tool scans.

3. Sensitive data exposure example

A request appears to retrieve credit card information in a user's wallet for viewing or editing.

A legitimate response shows that the data is not in the body of the response of encrypted JSON and is only exposed when displayed on the screen. The text in red shows bad coding methodology in which the credit card, status, and CVV are all in the response body text in plain text, allowing hackers to obtain information.

Request retrieving credit card info

Request: POST /payments/storecard/json HTTPS/1.1

DEBUG INFO:

```
Host: payments.host.com
Connection: close
Content-Length: 78
Cache-Control: max-age=0
Origin: https://payments.host.com
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0;Win64;X64) Apple WebKit/527.36
(KHTML, like Gecko) Chrome/87.0.480.88/Safari/537.36
Accept:text/html,application/xhtml+xml,application/xml;q0.9/image/
avif,image/webp,image/apng,*/*;q0.8,application/signed-exchange;v=b3;q=0.9
Referer: https://payments.host.com/methods/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: session-akljteha;359ldd906u4021c;.c75640202435t9078
```

Legitimate response:

```
HTTP/1.1 200 OK
Cachce-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cachce
Expires: Mon, 01 JAN 1990 00:00:00 GMT
Date: Wed 27 JAN 2021 15:44:39 GMT
Content-Type: application/json;charset=utf-8
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Connection: close
Content-Length: 55
```

Response with data exposure – HTTP response to the API call contains sensitive data in the message body

```
[{"PAN":"4111111122223333","status":"ok","CVV":"1234"}]
```



4. XML external entities

XML entities are used to request local data or files.

What it is

XML is a data format used to describe different data elements. XML being a rich and sophisticated standard also introduces “ to help define related data. Entities can access local or remote content, which can be as harmless as pulling schema definition or current stock price from a third-party website. Entities can, however, be used maleficently to request data or files, even if that data is never intended for outside access. If the application accepts XML directly or via XML uploads, processes SAML federated identity requests, or uses SOAP prior to version 12, it may present vulnerabilities.

How it works

An attacker sends malicious data lookup values asking the site, device, or app to request and display data from a local file. If a developer uses a common or default filename in a common location, an attacker’s job is easy.

Why it’s bad

Attackers can gain access to any data stored locally or can further pivot to attack other internal systems.

Countermeasures

- Whenever possible, use less complex data formats, such as JSON, and avoid serialization of sensitive data.
- Patch or upgrade all XML processors and libraries in use by the application.
- Most of the time, you can safely disable XML EE and DTD processing in an XML tool.
- Use the OWASP cheat sheet for prevention options: https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html.
- Implement a positive allow list “) server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
- Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or something similar.
- Use SAST tools like SonarQube, which will detect XXE in the source code.
- Utilize manual code reviews in large, complex applications with many integrations.

4. XML external entities example

An attacker interferes with an application's processing of XML or substitutes XML using specifically crafted DOCTYPE data to perform denial of service (DDOS), server-side request forgery (SSRF), or even remote code execution.

Scenario 1

A hacker attempts to extract data from the server by injecting an ENTITY XXE command to obtain access and contents of a file:///etc/passwd on the system, potentially exposing stored secrets to gain direct access to the host.

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

Scenario 2

A hacker probes the server's private network by changing the ENTITY line and injecting an ENTITY XXE command to determine a specific host and folder to see if it is exposed.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE foo [
    <!ELEMENT foo ANY>
    <!ENTITY xxe SYSTEM "http://192.168.0.1/mypasswords.txt">
]>
<foo>&xxe;</foo>
```

Scenario 3

A hacker attempts a denial-of-service attack by including a potentially endless file and injects an ENTITY XXE command to execute a denial-of-service attack using an endless file hack.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE foo [
    <!ELEMENT foo ANY>
    <!ENTITY xxe SYSTEM "file:///dev/urandom">
]>
<foo>&xxe;</foo>
```




5. Broken access control

Actions of authenticated users are enforced improperly.

What it is

Access control or authorization is how web apps let different users access various content, data, or functions. It's kind of how Netflix limits people on their plan view HD content while users can watch 4K. When access control is broken, malicious users can access more than they should, thus impacting system security, confidentiality, processing resources, or billed functionality.

How it works

Bypassing access controls, viewing or editing someone else's account, privileging an account, manipulating metadata, abusing CORS, or accessing authenticated pages while unauthenticated is the most common attack vectors. Sometimes, gaining unauthorized access is as simple as manually entering an unlinked URL in a browser, such as <https://example.com/admin>.

Why it's bad

Attackers can gain access to (and modify) data, accounts, and functions that they shouldn't.

Countermeasures

- Deny by default when it comes to the access controls of non-public resources.
- Minimize CORS usage and configure it correctly.
- Disable the webserver directory listing and ensure file metadata (e.g., .git) and backup files are not present within the web roots.
- Rate limit APIs and web endpoint access to minimize the harm from automated attack tooling.
- Invalidate JWT tokens after logout.
- Include unit and integration tests for functional access cases.

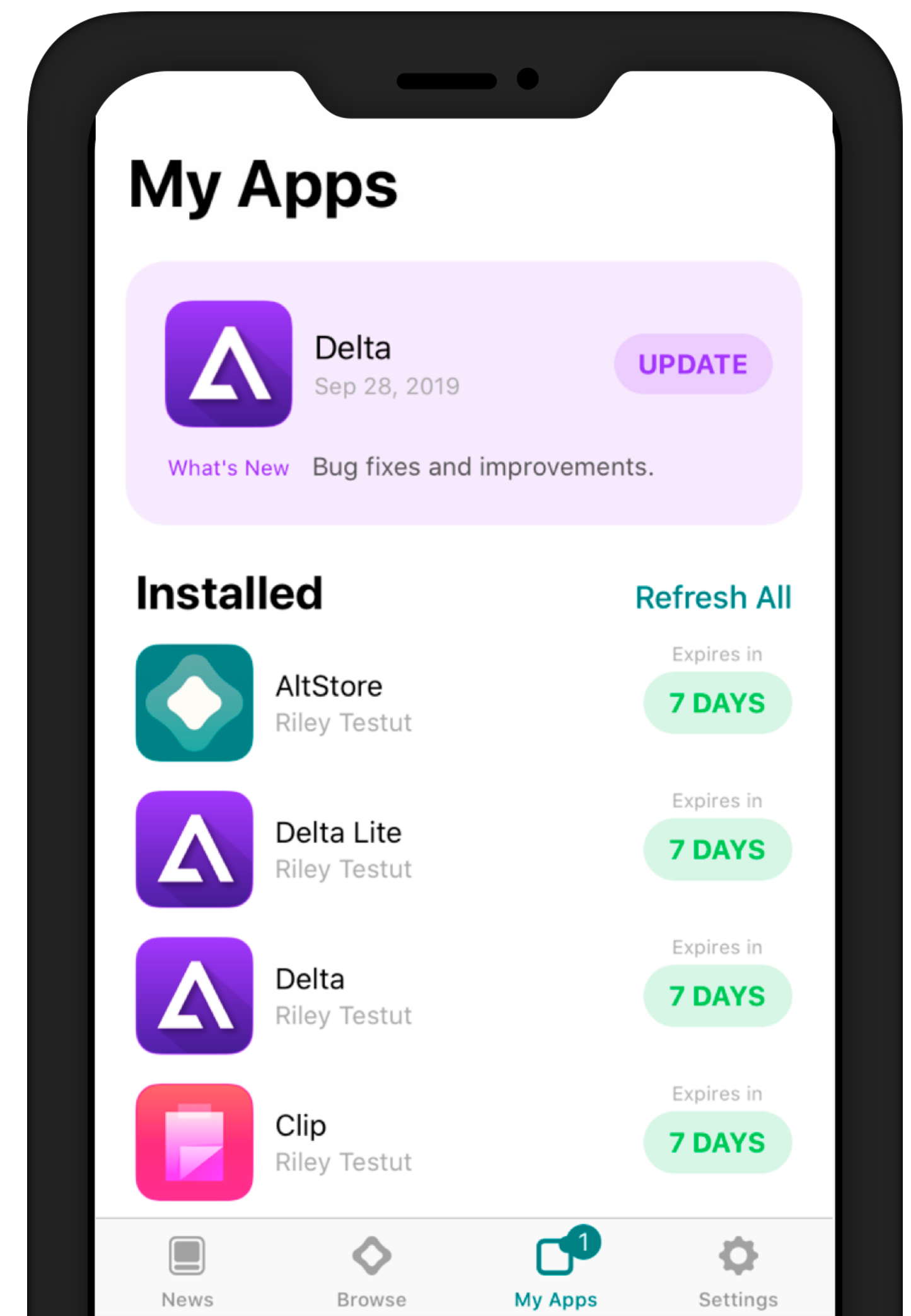
5. Broken access control example

Jailbreaking or rooting of a particular phone model and iOS is achieved via broken access control exploits.

The illustration shown depicts a broken access control in the form of an unofficial app store which looks and feels much like an actual app store. The broken access control allows sideload apps, tricking the iOS not to adhere to access control rules.

AltStore is an alternative app store for non-jailbroken devices. Unlike other unofficial app stores today, AltStore does not rely on enterprise certificates, which Apple has been cracking down on more and more recently. Instead, it relies on a lesser-known developer feature that allows you to use your Apple ID to install apps you've developed with Xcode, Apple's development toolkit. It is intended for people who might otherwise be unable to purchase a \$99/year developer account themselves. In fact, there's no technical reason why it's limited to apps installed from Xcode.

AltStore is a fully native, sandboxed iOS application that allows you to sideload apps by essentially tricking your phone into thinking it's installing apps that you made yourself. In reality, it can be installing any app.





6. Security misconfiguration

Manual, ad hoc, insecure, absence of security configurations enable unauthorized access.

What it is

Security misconfiguration is when vulnerabilities are left exposed during development, deployment, and support of an application (e.g., using default credentials, leaving files unprotected on public servers, having known but unpatched flaws, and more, and at any layer of the software stack).

How it works

Frameworks and IDEs become bigger and more complex. Engineering teams may start using them with default flaws. Once teams get into the daily routine, initial checklists and safeguards stop, things get missed, prioritization decisions are made, and vulnerabilities are left unaddressed. Luckily, most of these types of vulnerabilities are easy to find and fix.

Why it's bad

It makes it easy for even novice attackers to find and access valuable systems and data.

Countermeasures

- Development, QA, and production environments should all be configured identically, with different credentials used in each environment. The process should be automated to minimize the effort required to set up a new secure environment.
- Secure by default. Remove or do not install unused features, frameworks, and libraries.
- For multi-tenant applications, use segmentation where different tenants are separated via containerization, cloud security groups, physical databases, file system folders, etc.
- Remediate all vulnerabilities reported by SAST and DAST tools.
- Utilize HTTP security headers.

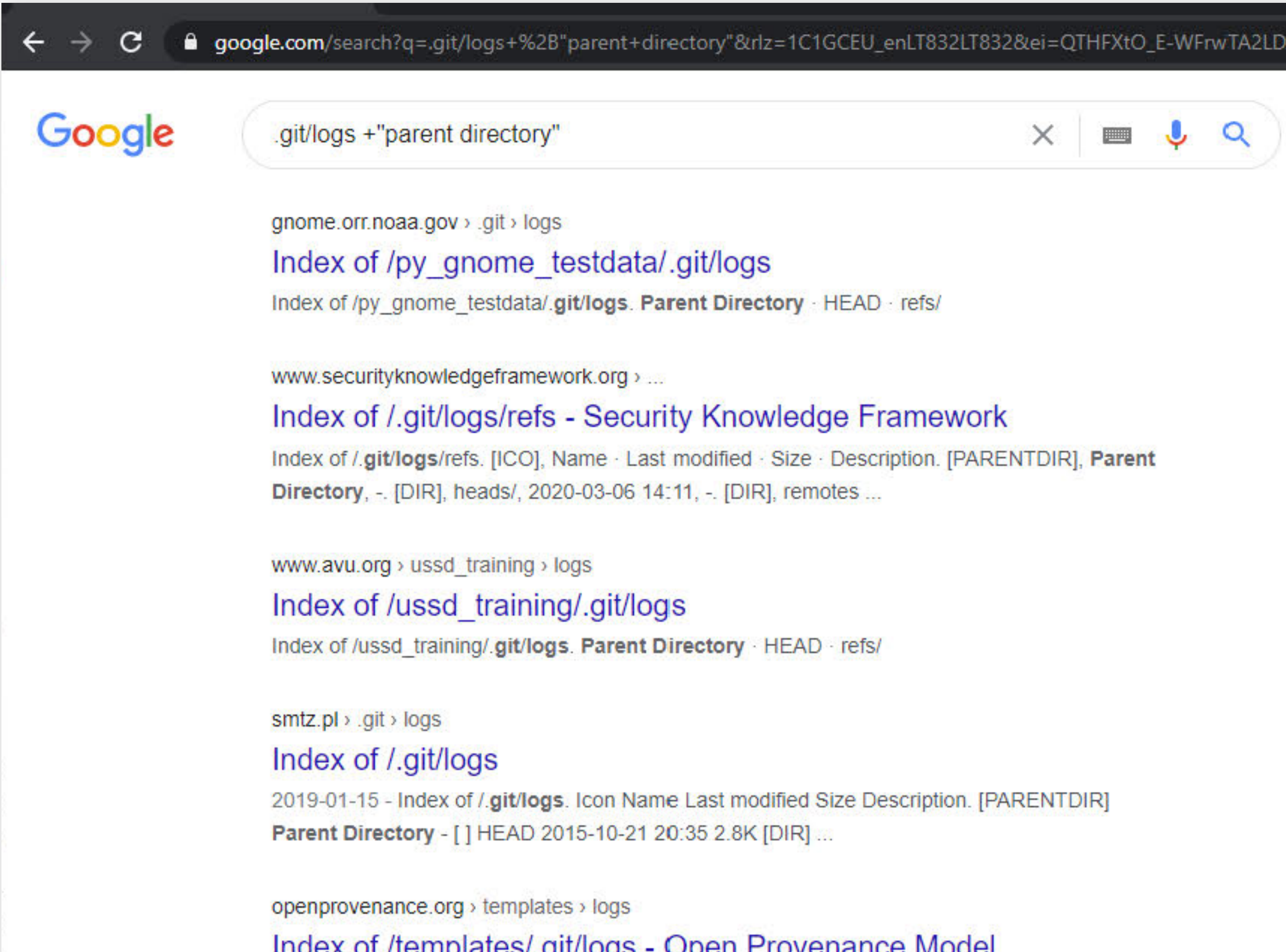
6. Security Misconfiguration example

Engineering or DevOps teams misconfigure an application or container unknowingly expose them by allowing users access to a parent directory.

As seen here, using simple search and index shows a parent directory access.

Index of /home/000~Root~000

Name	Last modified	Size
Parent directory	2015-11-24 15:40	35K
Git.om		
aquota.group	2019-01-19 01:22	0
aquota.user	2019-01-19 01:22	0
backup/	2018-03-01 22:55	-
bin/	2018-10-10 01:23	-
dev/	2018-09-24 14:13	-
etc/	2019-01-23 08:00	-
home/	2019-01-22 12:03	-





7. Cross-site scripting (XSS)

A web application includes untrusted data in a new web page without proper validation.

What it is

XSS allows malicious code to be added to a web page or app (e.g., via user comments or form submissions used to define the subsequent action). Since HTML mixes control statements, formatting, and requested content into the web page's source code, it allows an opportunity for unsanitized code to be used in the resulting page. XSS is the second most prevalent issue in the OWASP Top 10 and is found in around two-thirds of all applications.

How it works

When a web page or app utilizes user-entered content as part of a resulting page without checking for bad input, a malicious user could enter content that includes HTML entities.

Why it's bad

Attackers can change the behavior of an app, direct data to their systems, and corrupt or overwrite existing data.

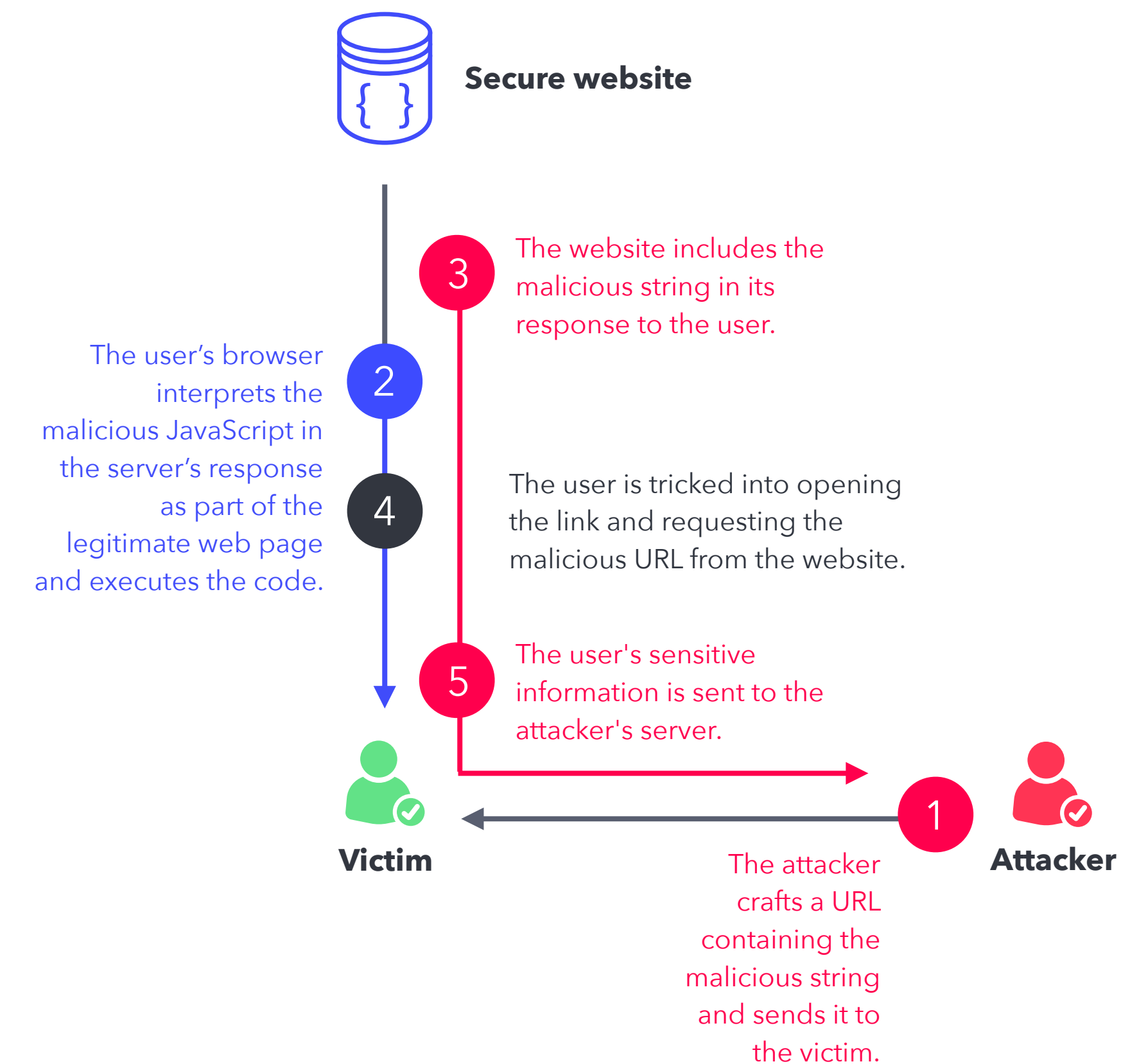
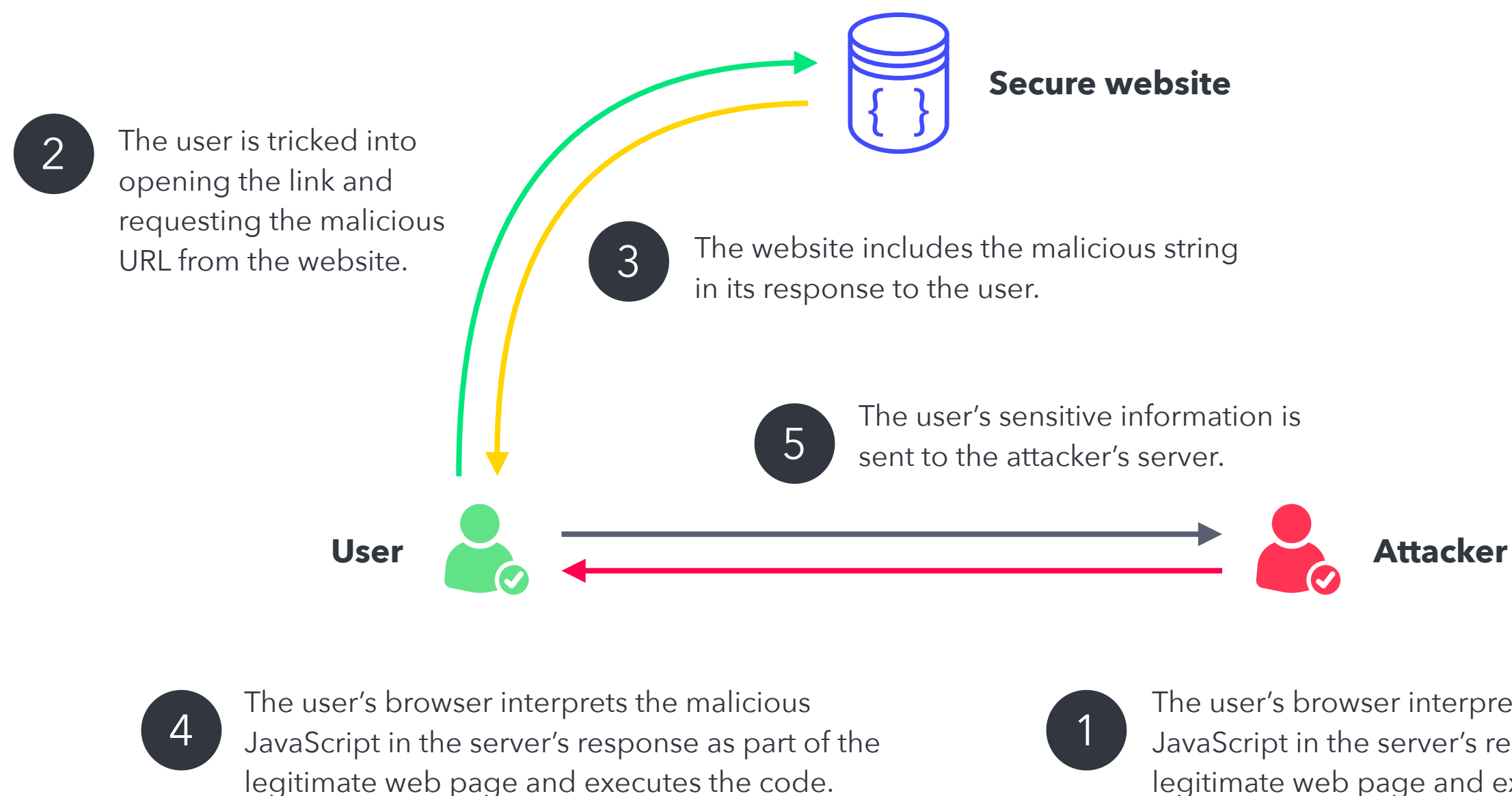
Countermeasures

- Use frameworks that automatically escape XSS by design. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
- Escape untrusted HTTP request data based on the context in the HTML output (e.g., body, attribute, JavaScript, CSS, or URL) to resolve reflected and stored XSS vulnerabilities.
- Apply context-sensitive encoding when modifying the browser document on the client side acts against DOM XSS.
- Enable a Content Security Policy (CSP) against XSS. It is effective if no other vulnerabilities exist that allow placing a malicious code file (e.g., compromised CDN, an unsecured file system on the server).

7. Cross-site scripting (XSS) example 1

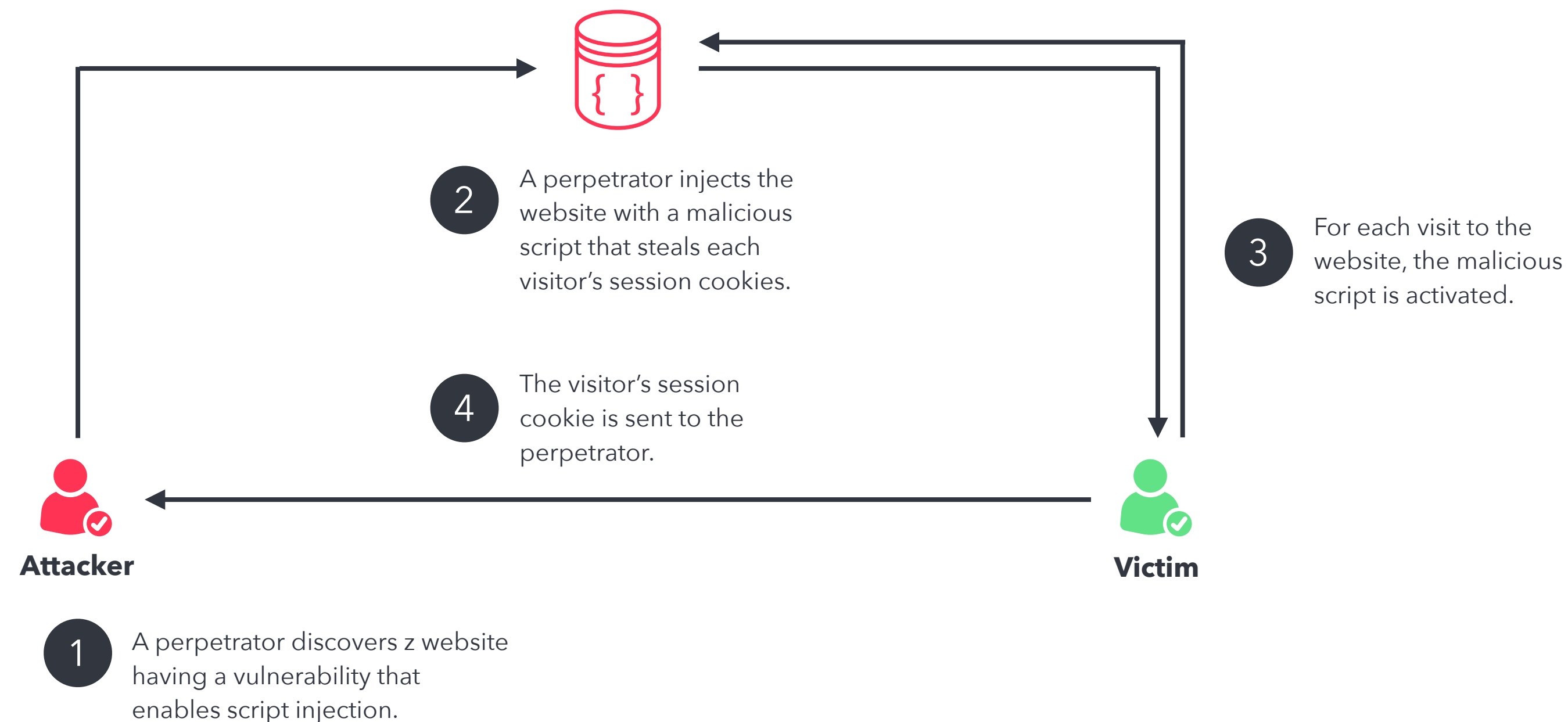
Reflected XSS: The application or API includes unvalidated and unescaped user input as part of HTML output.

The attacker executes arbitrary HTML and JavaScript in the victim's browser. Typically, the user needs to interact with a compromised link that points to an attacker-controlled page, such as malicious watering hole websites or advertisements.



7. Cross-site scripting (XSS) example 2

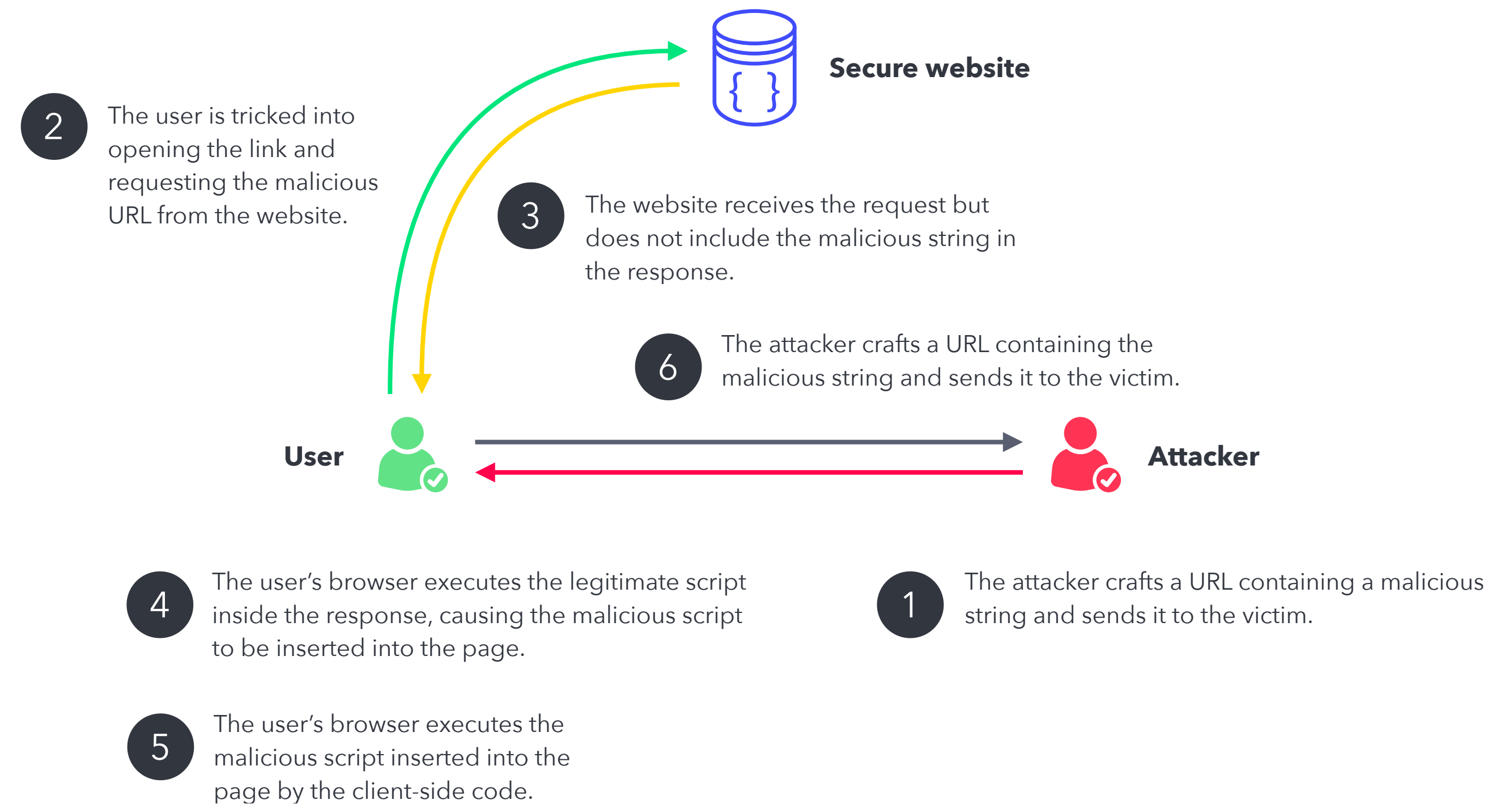
Stored XSS: Often considered a high or critical risk, the application or API stores unsanitized user input that is viewed later by another user or an administrator.



7. Cross-site scripting (XSS) example 3

DOM XSS: JavaScript frameworks, single-page applications, and APIs that dynamically include attacker controllable data to a page.

DOM-based XSS is a variant of both stored (i.e., persistent) and reflected XSS. As web applications become more advanced, an increasing amount of HTML is generated by JavaScript on the client-side rather than by the server. Even with completely secure server-side code, the client-side code unsafely includes a user input in a DOM update after the page has loaded.





8. Insecure deserialization

Hostile serialized objects are accepted, resulting in remote code execution.

What it is

Before data is stored or transmitted, bits are serialized so that it can be restored later to the data's original structure. Reassembling a series of bits back into a file or object is called deserialization.

How it works

Data expected to be deserialized can be tampered with while in disk storage or in transit over HTTP/TCP and include malicious code. Issues occur when the application does not verify the data's source or contents before deserialization.

Two primary types of deserialization attacks:

1. Object and data structure-related attacks modify application logic, buffer overflows, or remote code execution.
2. Data tampering attacks use existing data structures but change the content (e.g., for access control related attacks).

Why it's bad

Attackers can build illegitimate objects that execute commands within an infected application.

Countermeasures

- Use serialization mediums that only permit primitive data types.
- Do not accept serialized objects from untrusted sources.
- Implement integrity checks, such as digital signatures.
- Enforce strict type constraints during deserialization.
- Isolate running code that deserializes in low privilege environments.
- Log deserialization failures and anomalies.
- Restrict incoming and outgoing network connectivity from services or containers with deserialization.
- Implement alerting and monitoring on suspicious cases for all the countermeasures noted.



8. Insecure deserialization example

Hostile serialized objects occur from poor handling of the source code.

The source code here does not protect the application from a hacker to push an insecure object into the code, which invokes a malicious action rather than the original intent of the program.

Sample JSON payload

```
{"$type": "System.Configuration.Install.AssemblyInstaller.System.Configuration.Install, Version=4.0.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" "Path": "file:///c:/somePath/MixedLibrary.dll")
```

Source code

```
// System.configuration.Install.AssemblyInstaller
public void set_Path(string value)
{
    If (value == null)
    {
        this.assembly = null;
    }
    this.assembly = Assembly.LoadFrom(value);
}
```



9. Components with known vulnerabilities

Known vulnerabilities are found and exploited before they are fixed.

What it is

When vulnerabilities become known, vendors generally fix them with a patch or update. The process of updating the software should eliminate or mitigate a specific vulnerability. Component-heavy development patterns can lead to development teams not understanding which components they use in the application or API, much less keeping them up-to-date.

How it works

Organizations sometimes fail to keep software up-to-date, especially when stacks are large or complex or require a significant undertaking to validate systems or products after an update. When an exploit is made public, or a patch is released, attackers know that some organizations do not act immediately. Hackers now have a window, ranging from days to years, to search for systems or applications for known vulnerabilities left unaddressed.

Why it's bad

As CVEs and security bulletins are public information, attackers have a recommended path to exploit vulnerabilities. Organizations have little excuse for leaving the path open.

Countermeasures

- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Fix and upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion.
- Use software composition analysis tools to automate the process of monitoring systems for vulnerabilities published in CVE and NVD lists.
- Only obtain components from official sources over secure links.
- Prefer signed packages to reduce the chance of including a modified, malicious component.
- Avoid libraries and components that are unmaintained or do not create security patches for older versions.
- Secure the components' configurations by applying countermeasures for security misconfiguration.

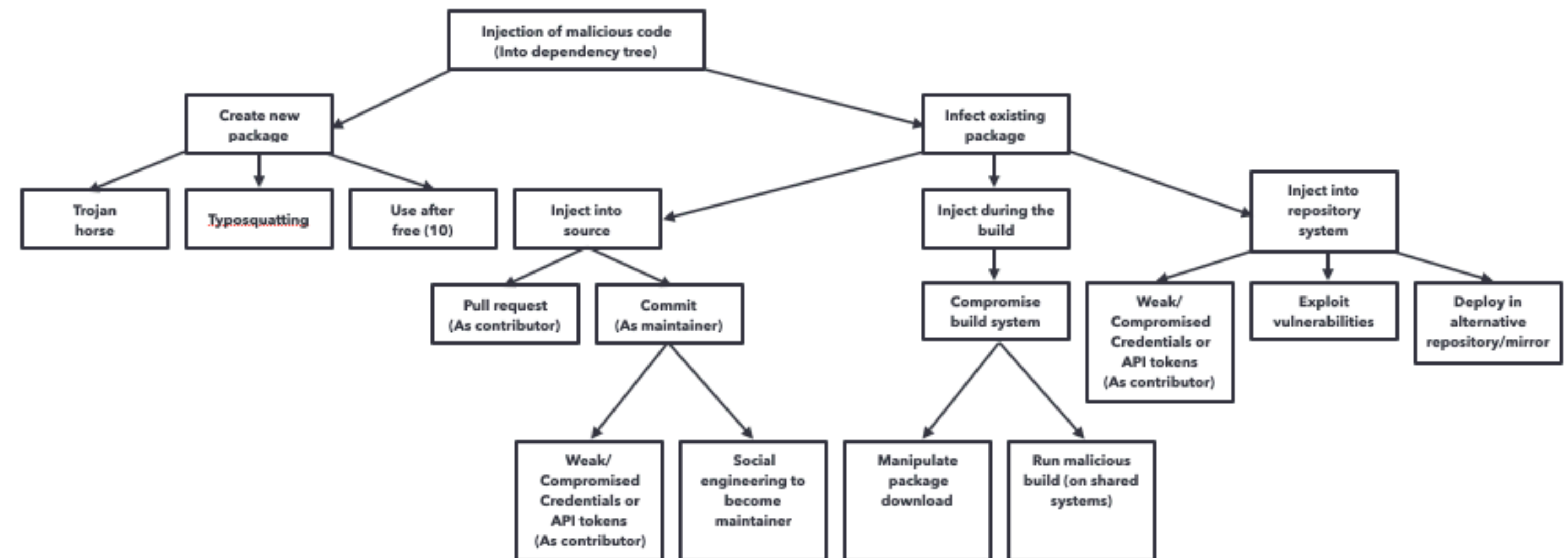
9. Components with known vulnerabilities example 1

Code libraries, framework, and component exploits occur.

Coding involves the use of open-source libraries, frameworks, and components from trusted sources. Problems arise when trusted sources can no longer be trusted. Verifying all dependency libraries and ensuring any known vulnerabilities go unresolved when upgrades, patches, or fixing to a specific version of the using a private binary repo would help.

```
NPM install <package name>
```

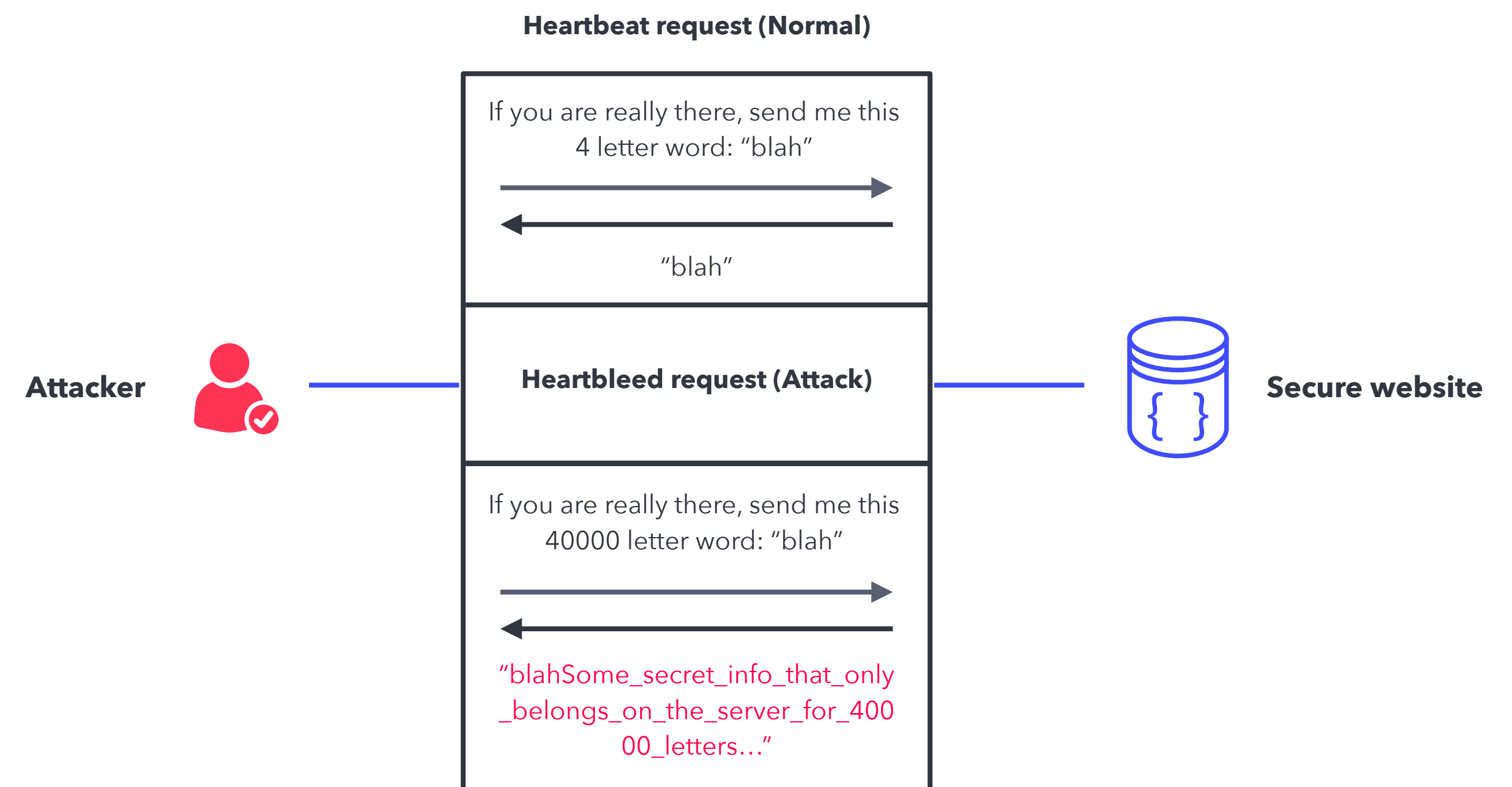
```
"dependencies": {  
  "express": "^4.3.0",  
  "dustjs-helpers": "~1.6.3",  
  "continuation-local-storage": "^3.1.0",  
  "pplogger": "^0.2",  
  "auth-paypal": "^2.0.0",  
  "wurfl-paypal": "^1.0.0",  
  "analytics-paypal": "~1.0.0"  
}
```



9. Components with known vulnerabilities example 2

The discovery of the Heartbleed vulnerability represented a turning point for OpenSSL with attackers using an OpenSSL session.

Typically, a session uses in a normal call for the word “blah” and four letters. However, in an attack, the OpenSSL exploit allows a hacker to ask for additional 3996 letters, enabling the server to expose additional data.





10. Insufficient logging & monitoring

Insufficient monitoring allows attackers to work unnoticed.

What it is

Organizations aren't actively looking for attackers or suspicious activities, and hackers go undetected.

How it works

Software and systems have monitoring abilities for organizations to see logins, transactions, traffic, and more. By monitoring for suspicious activity, such as failed logins, organizations can potentially see and stop suspicious activity.

Why it's bad

Attackers rely on the lack of monitoring to exploit vulnerabilities before they're detected. Without monitoring and the logging to look back to see what happened, attackers can cause damage now and in the future.


Countermeasures

- Ensure all login, access control failures, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for a sufficient time to allow for delayed forensic analysis.
- Examine the logs following penetration testing. The testers' actions should be recorded sufficiently to understand what damages may have been inflicted.
- Establish effective monitoring and alerting such that suspicious activities are detected and responded to in a timely fashion.
- Do not make systems more vulnerable by exposing logging and alerting events to a user or an attacker (see sensitive information exposure section).


10. Insufficient logging & monitoring example

Improperly setup logging, monitoring, and alerting at the operating system, application, authentication, and security logs unknowingly leaves the door open for hackers to enter.


Ensuring physical, logical, and other security measures are in place is your first line of defense. Just like the lock on your front door, if you leave it unlocked, someone is going to eventually come in.




Recording security incidents and policy violations.




Detecting and alerting to possible intrusions.




Maintaining evidence for legal proceedings.



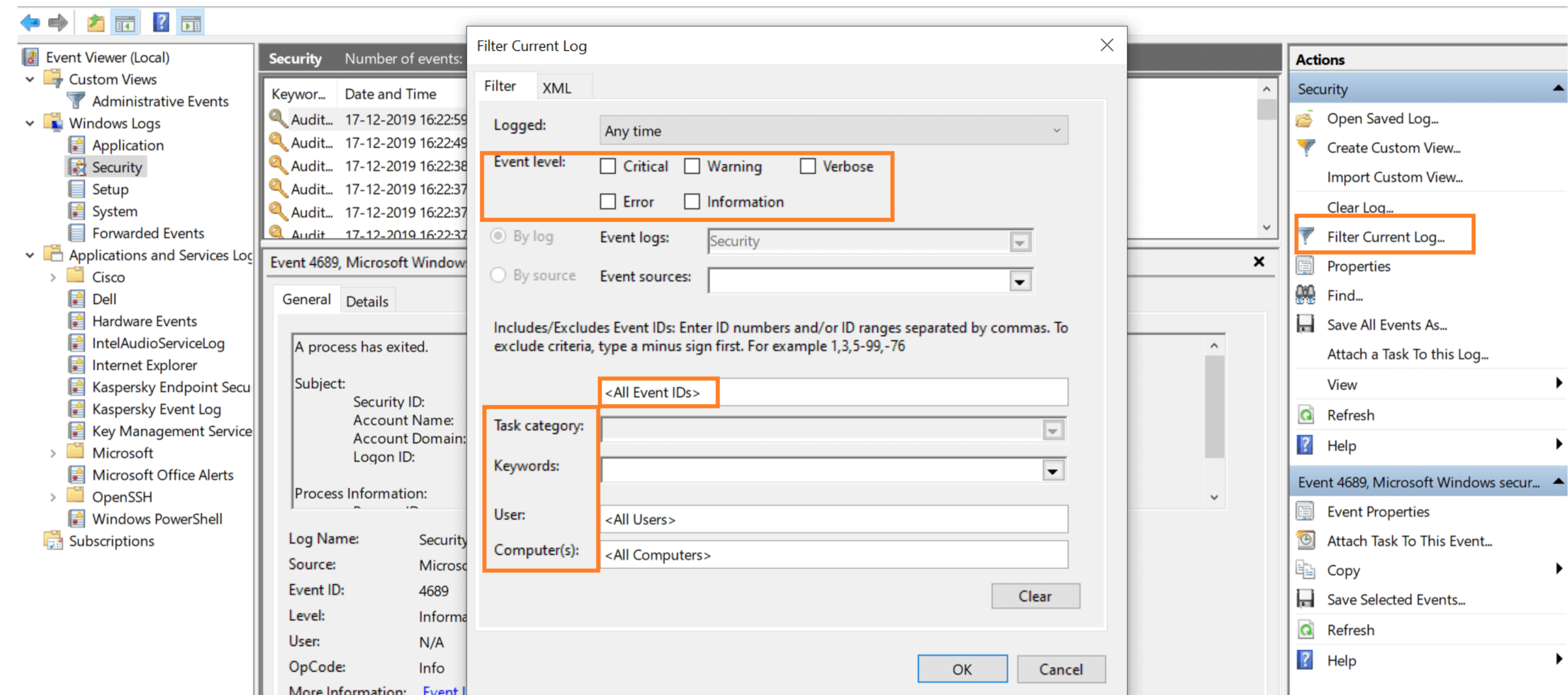
Measuring application performance.



Gathering information on application errors.




Maintaining an audit log for investigation and forensics





Locations and size

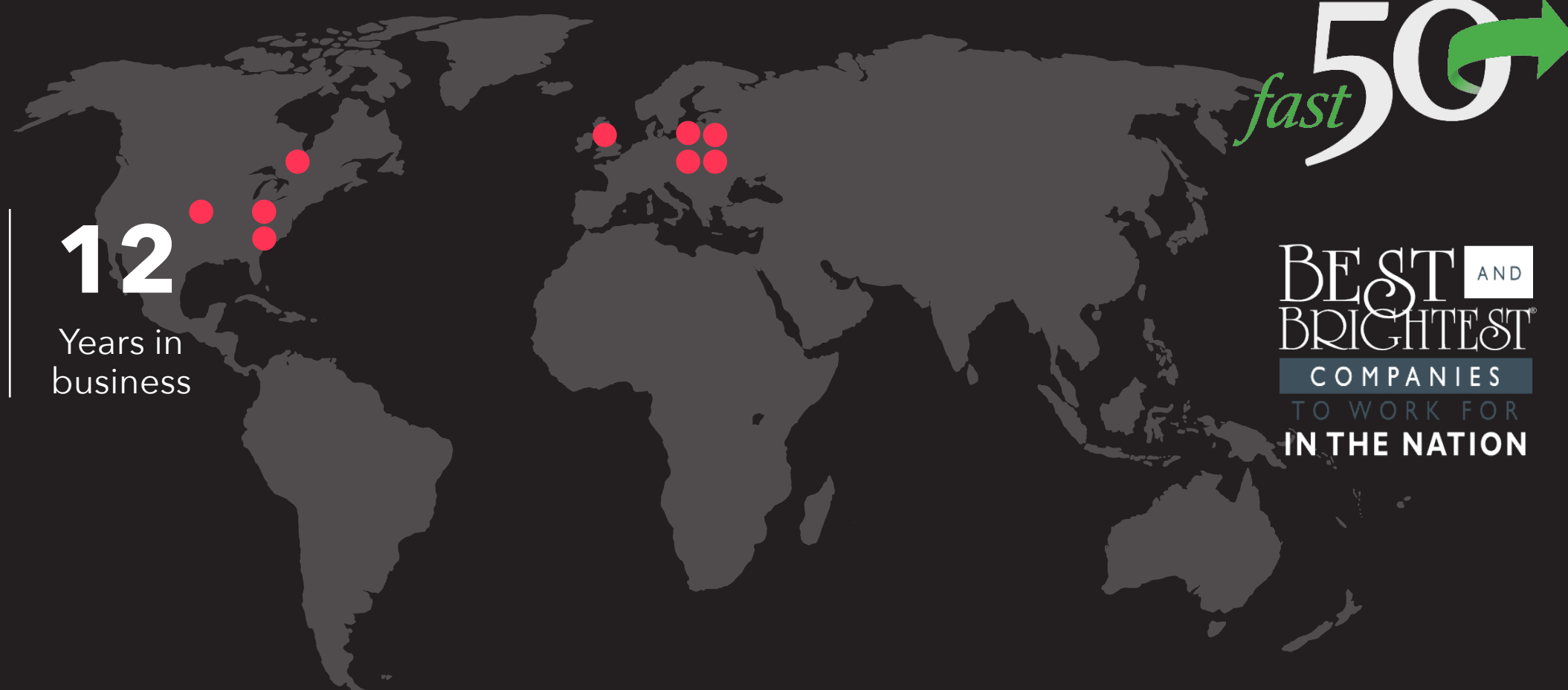
Large enough to handle digital transformations, small enough to provide exceptional service

 Atlanta, Chicago, Denver, Kaunas, Klaipeda, London, Toronto, Vilnius, Warsaw.

600
Full time employees

6
Offices

12
Years in business



Managing Directors



Laura Graves

Chicago



Will Kelly

Chicago, Toronto



Ian Spatz

London



Thomas Lukareski

Chicago



Tyson Stewart

Chicago