

A Brief Overview of the OWASP[®] Top Ten

Gabriel Kronfeld, BAsC

November 2024

Acronyms To Be Used

OWASP: Open Worldwide Application Security Project

SQL: Structured Query Language, a common database language

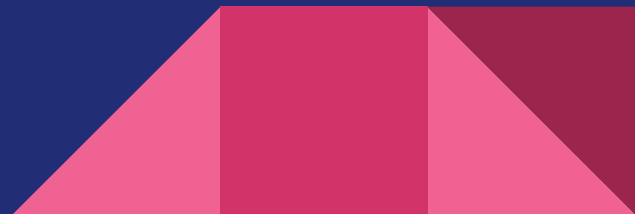
VPS: Virtual Private Server (using part of someone else's machine for your server)

API: Application Programming Interface

OS: Operating System

MFA: Multi Factor Authentication

URL: Uniform Resource Locator



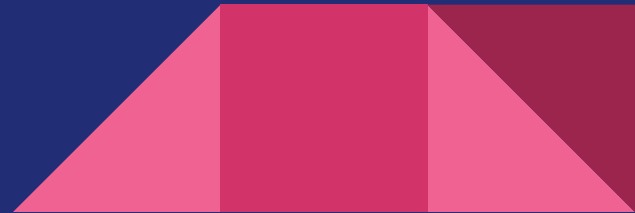
What is the “OWASP Top Ten?”

Quote: The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

The OWASP Top Ten is a list of the Top Ten MOST CRITICAL security vulnerabilities

You may notice many points on the list have overlapping causes, effects, and solutions.

Source: owasp.org/www-project-top-ten



3 Main Focuses

Vulnerabilities in:

- Your Code + infrastructure
- External code (libraries, plugins, etc)
- Client inputs



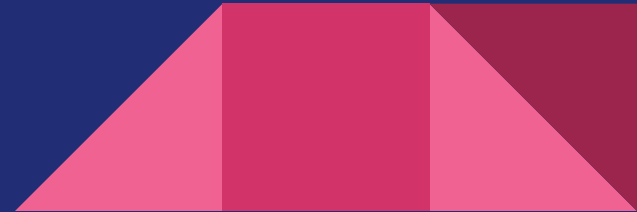
A01:2021 - Broken Access Control

Simply: controlling what people can access.

When is it broken? When they see and do more than they should be allowed to!!

Failure can cause:

- Disclosure of unauthorized information
- Ability outside user's limits (e.g. modification, deletion, performance, etc.)

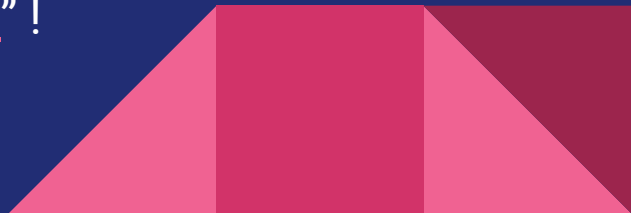


Broken Access Control Examples:

- `https://example-website.com/adminpanel`
- Navigating to `/index.html`, or other index page for admin purposes
- Allowing all users to access protected data/not verifying permissions

```
chmod 777 getmydata.sh
```

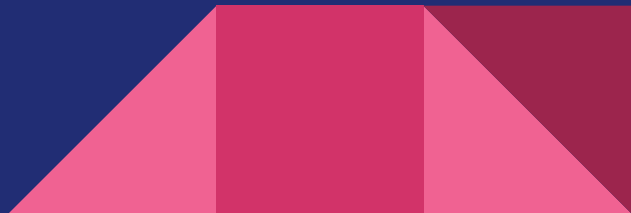
Try navigating to “<https://gabrielkronfeld.com/admin-dev>” !



Protect Yourself! - Broken Access Control

- Follow the principle of least privilege!
- Use **TRUSTED** server-side code and APIs,
- Disable web server listings
- Enforce record ownership (do not assume a user has execute permissions just because they can access it!)

```
chmod 640 getmydata.sh
```

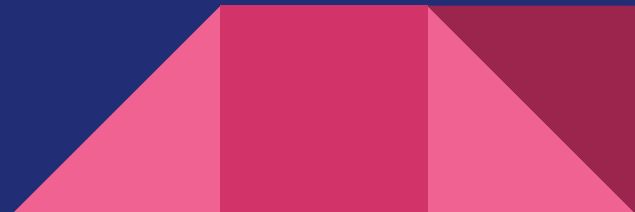


A02:2021 - Cryptographic Failures

AKA: Sensitive Data Exposure

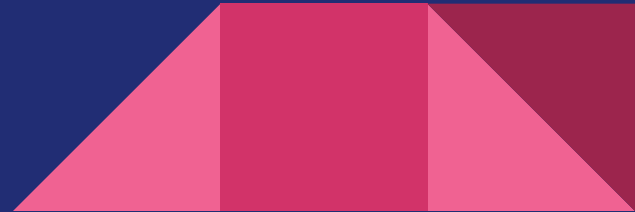
Failure can cause:

- the exposure of data
 - Passwords, credit cards, etc. being leaked
 - Transmissions being read in transit
- Broken Access Control



Cryptographic Failure Examples

- Hard Coding Passwords in your web page
- Uploading your API keys to GitHub
- Using obsolete cryptography methods (SHA1, SSL 1.0)
- Using default passwords/keys
- Not enforcing encryption keys (HTTP/HTTPS)
- Transmitting data over the Web in plain text



Cryptographic Failure Examples

- storing passwords as plain text, allowing an SQL injection flaw to have all passwords in plain text.

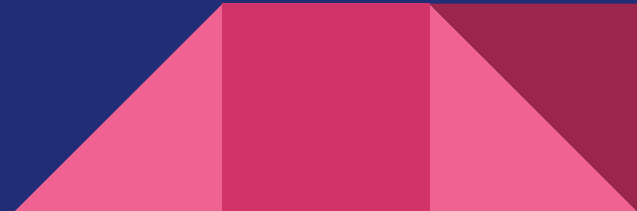
Note: Even hashing can still lead to them being cracked by a rainbow table if the algorithm is simple, or if you do not use a salt.

If you have insecure connection (e.g. insecure public network), an attacker can:

- monitor traffic,
- downgrade your connection to HTTP
- steal session cookies
- alter transported data

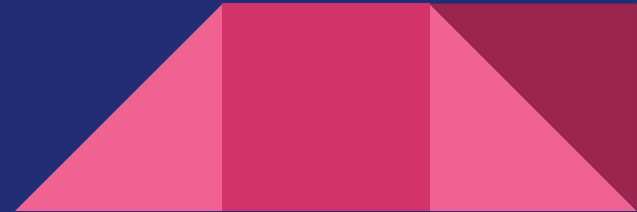
Rainbow table: a list of common hashed passwords

Salt: extra data added to your saved password



Protect Yourself! - Cryptographic Failures

- Do NOT store sensitive data unnecessarily (do not cache it client side)
- Make sure data is encrypted in transit
 - (e.g. TLS with FS, HTTPS Strict Transport Security)
- Use Authenticated Encryption
- Use up-to-date protocols and algorithms
- Store passwords using current, salted, hashing algorithms with work factors
 - Use proper hashing!
- Generate keys cryptographically

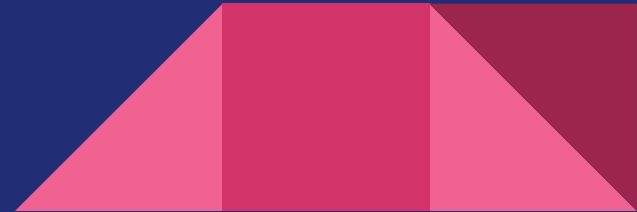


A03:2021-Injection

Data injection (famously SQL injection): The process of injecting commands to get additional data

Failure can cause:

- Data Leaks
- Modification of resources



Injection Examples

HI, THIS IS
YOUR SON'S SCHOOL.
WE'RE HAVING SOME
COMPUTER TROUBLE.



OH, DEAR - DID HE
BREAK SOMETHING?
IN A WAY-



DID YOU REALLY
NAME YOUR SON
Robert'); DROP
TABLE Students;-- ?



OH, YES. LITTLE
BOBBY TABLES,
WE CALL HIM.

WELL, WE'VE LOST THIS
YEAR'S STUDENT RECORDS.
I HOPE YOU'RE HAPPY.



AND I HOPE
YOU'VE LEARNED
TO SANITIZE YOUR
DATABASE INPUTS.

Injection Examples

How to avoid Bobby Tables

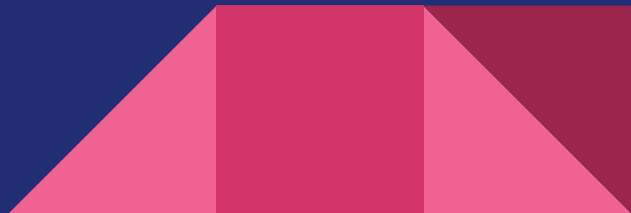
There is only one way to avoid Bobby Tables attacks

- Do not create SQL statements that include outside data.
- Use parameterized SQL calls.

That's it. Don't try to escape invalid characters. Don't try to do it yourself. Learn how to use parameterized statements. Always, every single time.

The strip gets one thing crucially wrong. The answer is not to "sanitize your database inputs" yourself. It is prone to error.

source:bobby-tables.com/about (direct quote)



Injection Examples

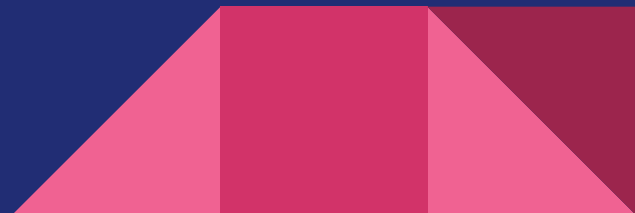
Other common vulnerabilities:

- Not validating or cleaning supplied data

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

```
Statement.executeQuery(query);
```

example.com/account?id='UNI/**/ON + SE%0bLECT SLEEP(10);--



Protect Yourself! - Injection

Prepared statements always treat client-supplied data as content of a parameter and never as a part of an SQL statement.

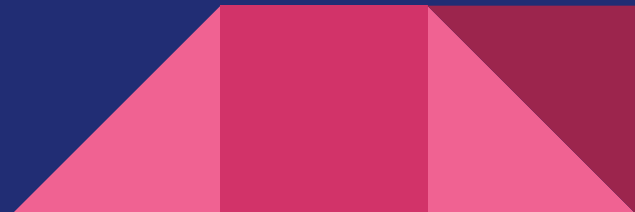
```
String custname = request.getParameter("customerName");
```

```
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
```

```
PreparedStatement pstmt = connection.prepareStatement( query );
```

```
pstmt.setString( 1, custname);
```

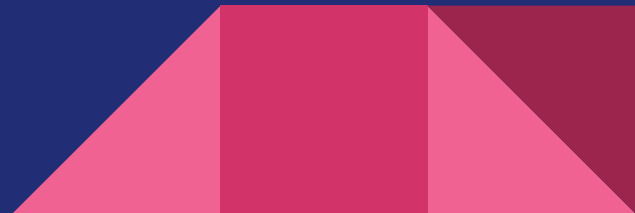
```
ResultSet results = pstmt.executeQuery( );
```



Protect Yourself! - Injection

- Use Prepared Statements!!
- Input Sanitization
 - Use Positive Validation and escape special characters when possible
 - Allowlists
- Use LIMIT so failures aren't as severe
- Again, use Prepared Statements

Use safe APIs! (see A08)



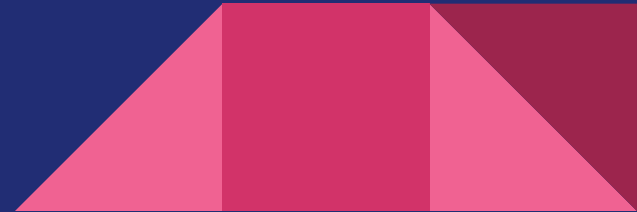
A04:2021 - Insecure Design

A very broad term encompassing different weaknesses.

design flaws != implementation weaknesses

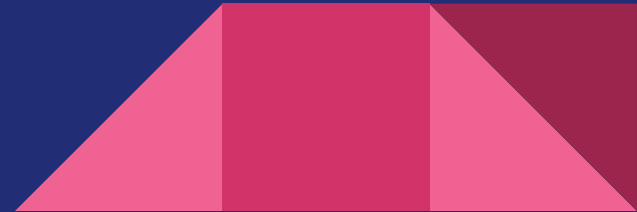
Failure can cause:

- Security vulnerabilities at an architectural level.



Insecure Design Examples

- A cinema allows group booking with a max of 15 attendees per booking before requiring a deposit. Malicious actors can make numerous accounts and reserve the entire cinema indefinitely, 15 seats at a time, preventing paying customers from buying seats, paralysing the cinema's income.
- E-commerce site having a lack of protection vs bots run by scalpers can ruin the store's reputation



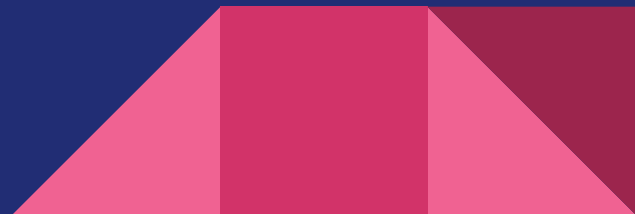
Protect Yourself! - Insecure Design

During the design process:

- Take into account how exposed the app will be
- Include security requirements in technical requirements
- Include threat modeling into refinement sessions

Establish a secure dev. lifecycle with appSec professionals

“Secure design is neither an add-on nor a tool that you can add to software”



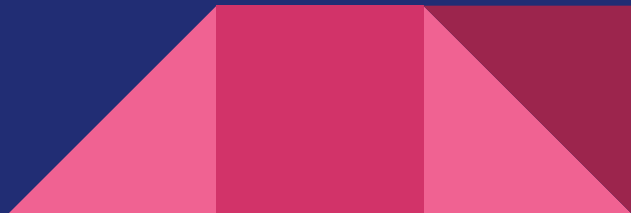
A05:2021 - Security Misconfiguration

Another largely-encompassing point,

When your configs are wrong, you open yourself up to vulnerabilities.

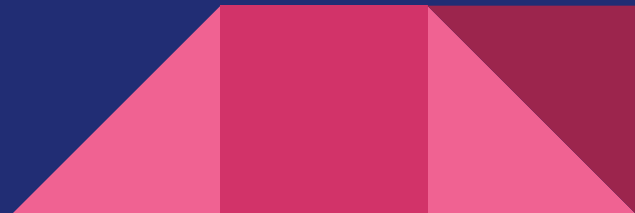
Failure can cause:

- Other vulnerabilities to become possible
 - Access control, cryptographic failures
 - etc



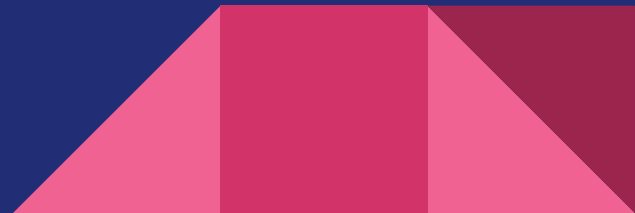
Security Misconfiguration Examples

- Error handling visible to users, including stack traces
- Unnecessary features enabled, including:
 - unneeded web pages,
 - test accounts,
 - expanded privileges,
 - unused services
- Out-of-date software (A06)
- Security settings aren't set to secure values



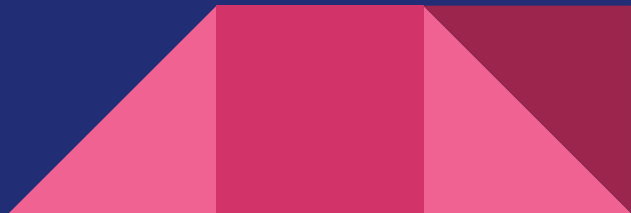
Security Misconfiguration Examples

- A VPS comes with pre-installed sample apps with known vulnerabilities (A08)
- Admin consoles with default password left unchanged (admin/admin) (A02)
- Directory listings visible to users (.htaccess, index.html) (A01)
- Not hiding sensitive files in .gitignore (symlinks?) (A04+)



Protect Yourself! - Security Misconfiguration

- Repeatable Security Hardening
 - Fast and easy
 - Automated and identical between dev, Quality Assurance, and production
- Minimize the platform
- Review configs frequently during patch management
- Automated security testing



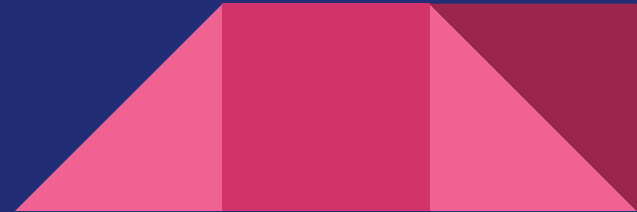
A06:2021 - Vulnerable and Outdated Components

You are likely vulnerable if:

- You don't know the versions of components + dependencies you use
- You don't scan for vulnerabilities
- If your software is vulnerable, outdated, or unsupported
 - From your APIs all the way to OS

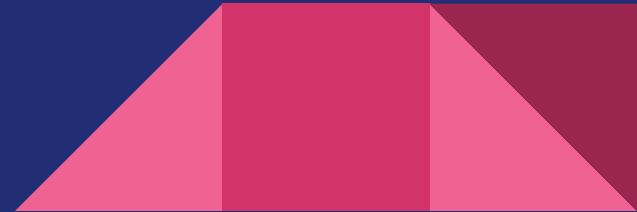
Failure can cause:

- Opening yourself up to known, discoverable, and preventable vulnerabilities



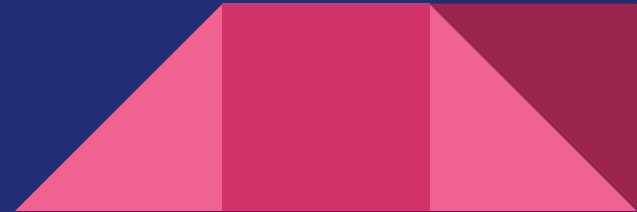
Vulnerable and Outdated Components Examples

- If you rarely update your software libraries, an external vulnerability (A08) can remain present for months
- IoT devices
 - Biomedical devices
- Log4j “Log4Shell” 0-day vulnerability
- SHODAN



Protect Yourself! - Vulnerable and Outdated Components

- Remove unused dependencies
- Keep an inventory of all components
 - OWASP Dependency Check
 - Retire.js
- Obtain components from OFFICIAL sources, over secure links
- Watch for used components becoming unmaintained



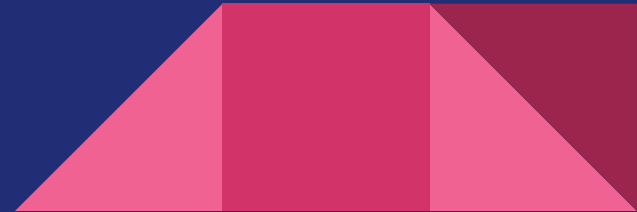
A07:2021 - Identification and Authentication Failures

What it is: the failure to properly confirm a user's identity.

A.K.A. Broken Authentication

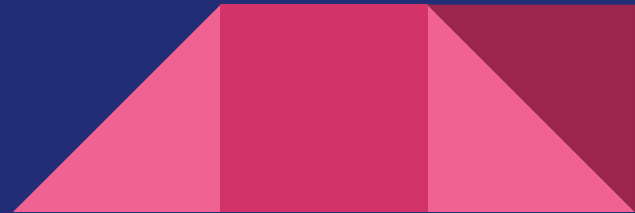
Failure can cause:

- Credential stuffing
- Brute force
- Session identifier exposure



Identification and Authentication Failure Examples

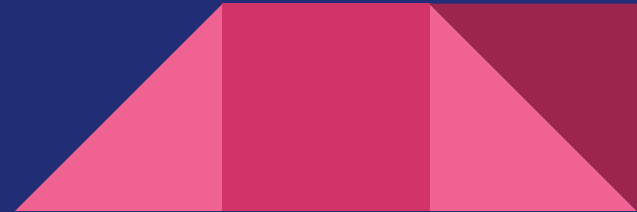
- Not invalidating session IDs
- Credential stuffing
- Forcing frequent rotating passwords with simple login
- Authentication timeout isn't set or is too long
- Missing MFA



Protect Yourself! - Identification and Authentication Failures

- Implement MFA
- NEVER deploy with default credentials
- NIST 800-63B 5.11 password guidelines
- Limit & delay failed login attempts
 - Warning: don't DOS yourself!
- Use the same messaging on all outcomes
- Implement “weak password” checks

Note: DOS: Denial of Service

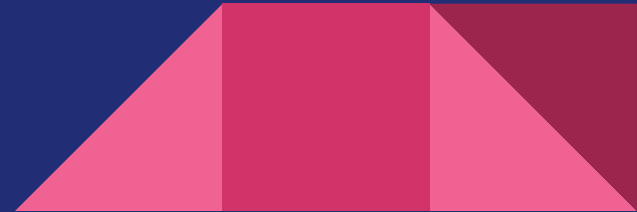


A08:2021 - Software and integrity Failures

What it is: The use of unverified, untrusted, or vulnerable external components

Failure can cause:

- Opening yourself up to other vulnerabilities and attack vectors
- Backdoors being added to your system



Software and integrity Failure Examples

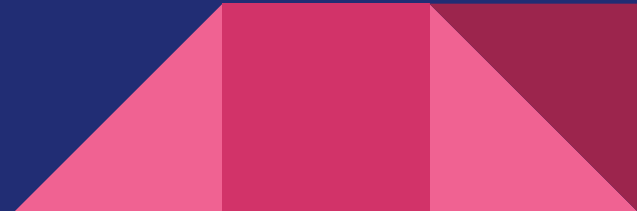
- SolarWinds Orion:

Nobelium attackers inserted a supply chain attack into software, giving them remote access to SolarWinds customers' system files

- UMN Linux Incident

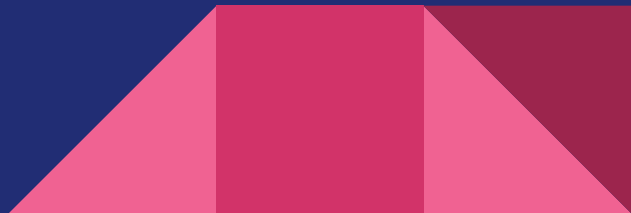
University of Minnesota attempted to intentionally insert bad patches to linux kernel github repo

- Unsigned updates for set-top boxes



Protect Yourself! - Software and integrity Failures

- Use Software from verified sources, with digital signatures when possible
 - See: linkin_park_numb.mp3.exe
 - shasum <filename>
- Use OWASP dependency-check or other software version tooling
- Ensure unsigned or unencrypted data isn't sent to untrusted clients



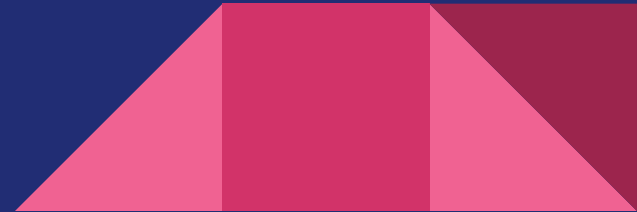
A09:2021 - Security Monitoring and Logging Failures

What it is: The failure to properly monitor and log events, including:

- Insufficient logging (omission of relevant info)
- Inserting sensitive information into log files (risking A01)

Failure can cause:

- An absence of information to be made aware of issues
 - Loss of auditable event recording (logins, transactions)
 - Unmonitored API activity
- Info leakage



Security Monitoring and Logging Failure Examples

A health plan provider operator did not keep ANY log information; an eventual post-incident review revealed a data breach present in the system for over 7 years

-OWASP

Air India suffered a data breach by cloud hosting provider; notified a month after the breach, leading to over a month of new clients needlessly being put at risk (on top of the decade worth of breached data)

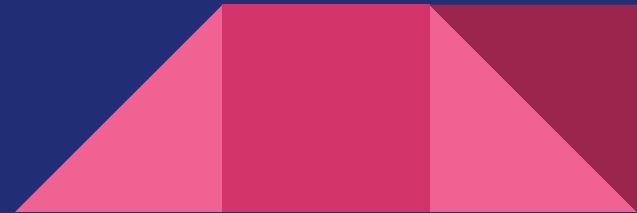
“While we had received the first notification in this regard from our data processor on 25.02.2021, we would like to clarify that the identity of the affected data subjects was only provided to us by our data processor on 25.03.2021 and 5.04.2021”.

Hindustan times:”Air India says 10 years of passenger data breached”, May 22, 2021



Protect Yourself! - Security Monitoring and Logging Failures

- Make sure all logins, access control, and input validation failures are logged with enough context to be actionable, and held long enough for analysis
- Ensure logs are in an easily consumable format
- Ensure logs are encoded correctly to prevent attacks
- Establish an incident response and recovery plan (NIST 800-61 rev. 3)

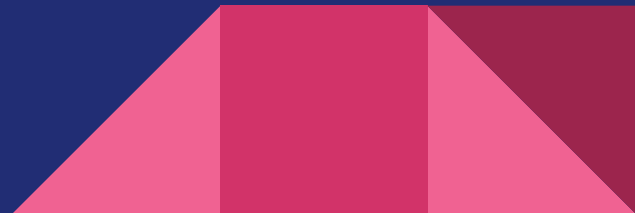


A10:2021 - Server-Side Request Forgery

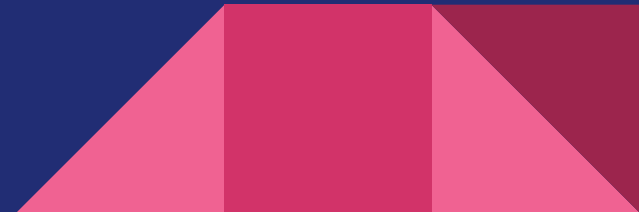
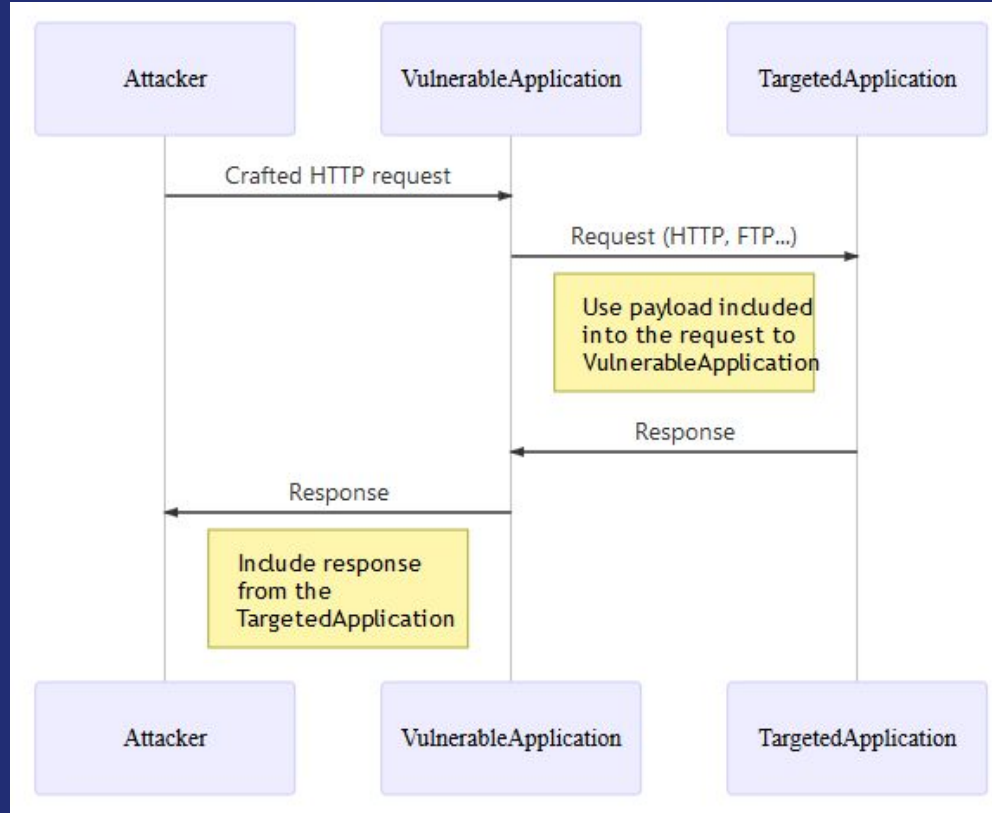
What it is: A security flaw occurring when an attacker manipulates a web app or API to return typically internal-only information/resources, caused by fetching a remote resource without validating the user-supplied URL.

Failure can cause:

- Data exposure
- Unauthorized access (A01)
- Remote code execution (A03)



Server-Side Request Forgery Examples



Server-Side Request Forgery Examples

- If a web app allows users to upload images by providing a URL to an image they want to process, a client may insert an input with a malicious URL pointing to an internal resource on the server, like a database or internal file. If the server blindly processes the message, it will follow the input making a request to the internal resource specified and return the data to the client.

```
POST /product/stock HTTP/1.0
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 118
```

```
(Vulnerable:)
```

```
stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3Fproduct
```

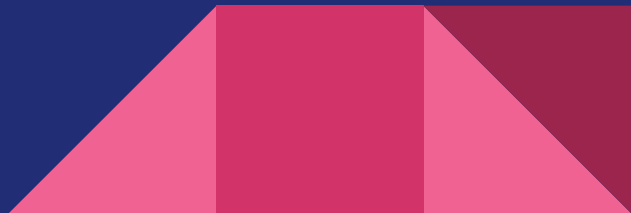
```
stockApi=http://localhost/admin
```



Protect Yourself! - Server-Side Request Forgery

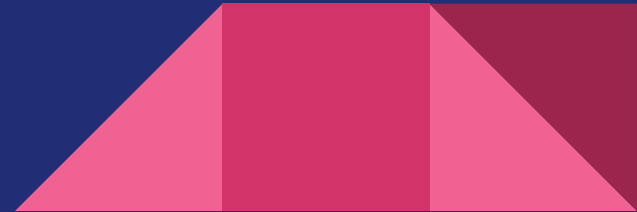
- Segment remote resource functionality to reduce the impact of SSRF
- Enforce deny-by-default policies

- Sanitize and validate ALL client supplied inputs
- Enforce URL schemas with a positive allow list
- Do not send raw response to clients



Future Reading

- [CWE.mitre.org](https://cwe.mitre.org)
- [top10proactive.OWASP.org](https://top10proactive.owasp.org)
- bobby-tables.com
- [OWASPsamm.org](https://owasp.org/OWASPSAMM)
- [OWASP.org/top10](https://owasp.org/top10)



The top right corner of the slide features a decorative graphic consisting of several overlapping squares and triangles in various shades of blue, creating a modern, geometric pattern.

Any Questions?