OWASP Top 10

Application Security #2-3

Norbert Langner

> **The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted.**

https://owasp.org/Top10/A00-about-owasp/

> **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.**

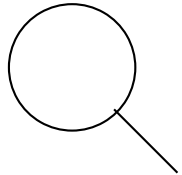
https://owasp.org/www-project-top-ten/

OWASP Top 10 in a nutshell

- Latest release: 2021-09-24, The 7th edition
- Next release is going to be publish in the 1st half of 2025
- First Top 10 published in 2003
- Risk awareness document not an oracle nor easily testable issues
- High level description
- Top 1 = most serious security risk

Methodology

- **Data and survey driven research**
  - Grouped all CVEs with CVSS scored by CWE (exploitability and technical impact)
- **Provided data covers over 500,000 applications**
- **8 categories derived from data, 2 from a survey**
- **Almost 400 CWEs considered**
- **Focus on a root cause**



Factors

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurren-ces	Total CVEs

Common Vulnerabilities and Exposures

- The mission of the CVE® Program is to identify, define, and catalog publicly disclosed cybersecurity vulnerabilities. There is one CVE Record for each vulnerability in the catalog. [3]
- Operated by The Mitre Corporation
- More info + podcasts: https://www.cve.org/About/Overview
- Example CVE: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-48454

CVE-ID

CVE-2022-48454 Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

In wifi service, there is a possible out of bounds write due to a missing bounds check. This could lead to local denial of service with no additional execution privileges needed

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:https://www.unisoc.com/en_us/secy/announcementDetail/https://www.unisoc.com/en_us/secy/announcementDetail/1719615756246777857
- URL:https://www.unisoc.com/en_us/secy/announcementDetail/https://www.unisoc.com/en_us/secy/announcementDetail/1719615756246777857

Assigning CNA

Unisoc (Shanghai) Technologies Co., Ltd.

Date Record Created

20230413	Disclaimer: The record creation date may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

Phase (Legacy)

Assigned (20230413)

Votes (Legacy)

Comments (Legacy)

Proposed (Legacy)

N/A

This is an record on the CVE List, which provides common identifiers for publicly known cybersecurity vulnerabilities.

SEARCH CVE USING KEYWORDS: Submit
You can also search by reference using the CVE Reference Maps.

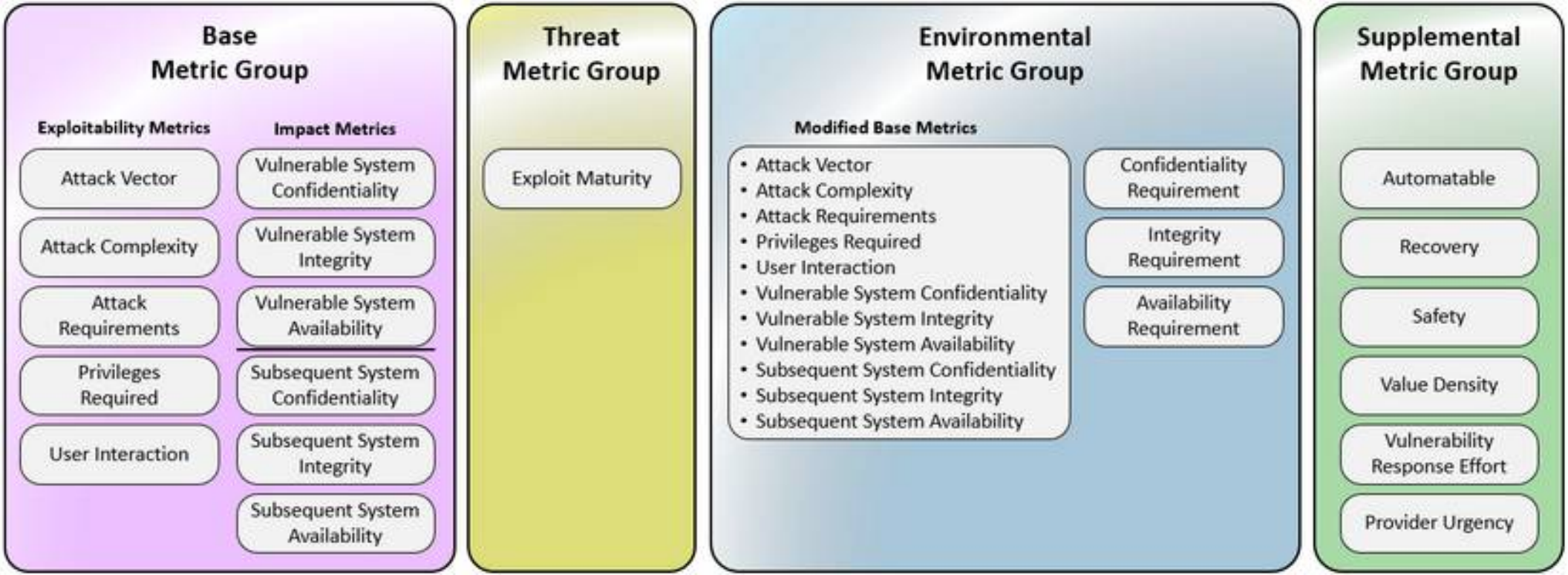
For More Information: CVE Request Web Form (select "Other" from dropdown)

BACK TO TOP

Common Vulnerability Scoring System

- Open framework for communicating the characteristics and severity of software vulnerabilities
- CVSS is owned and managed by FIRST.Org, Inc.
- Score range from 0 to 10
- See: https://www.first.org/cvss/calculator/4.0

CVSS Metrics



Common Weaknesses Enumeration

- Community-developed list of common software and hardware weakness types
- The CWE List includes both software and hardware weakness types
- Every CWE has its id in CWE-[1-9][0-9]* form
- Example entry: https://cwe.mitre.org/data/definitions/787.html

Example – CWE-787: Out-of-bounds Write

CWE-787: Out-of-bounds Write

Weakness ID: 787
Abstraction: Base
Structure: Simple

View customized information: Conceptual | Operational | Mapping Friendly | Complete | Custom

Description

The product writes data past the end, or before the beginning, of the intended buffer.

Extended Description

Typically, this can result in corruption of data, a crash, or code execution. The product may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent write operation then produces undefined or unexpected results.

Alternate Terms

Memory Corruption: Often used to describe the consequences of writing to memory outside the bounds of a buffer, or to memory that is invalid, when the root cause is something other than a sequential copy of excessive data from a fixed starting location. This may include issues such as incorrect pointer arithmetic, accessing invalid pointers due to incomplete initialization or memory release, etc.

Relationships

Modes Of Introduction

Phase	Note
Implementation	

Applicable Platforms

Languages

C *(Often Prevalent)*

C++ *(Often Prevalent)*

Class: Assembly *(Undetermined Prevalence)*

Technologies

Class: ICS/OT *(Often Prevalent)*

Example – CWE-787: Out-of-bounds Write

```
int id_sequence[3];


/* Populate the id array. */
id_sequence[0] = 123;
id_sequence[1] = 234;
id_sequence[2] = 345;
id_sequence[3] = 456;
```

Countdown

Server-Side Request Forgery

#10

Security Logging and Monitoring Failures

#9

Countdown

Software and Data Integrity Failures

#8

Identification and Authentication Failures

#7

Countdown

Vulnerable and Outdated Components

#6

Security Misconfiguration

#5

Countdown

Insecure Design

#4

Injection

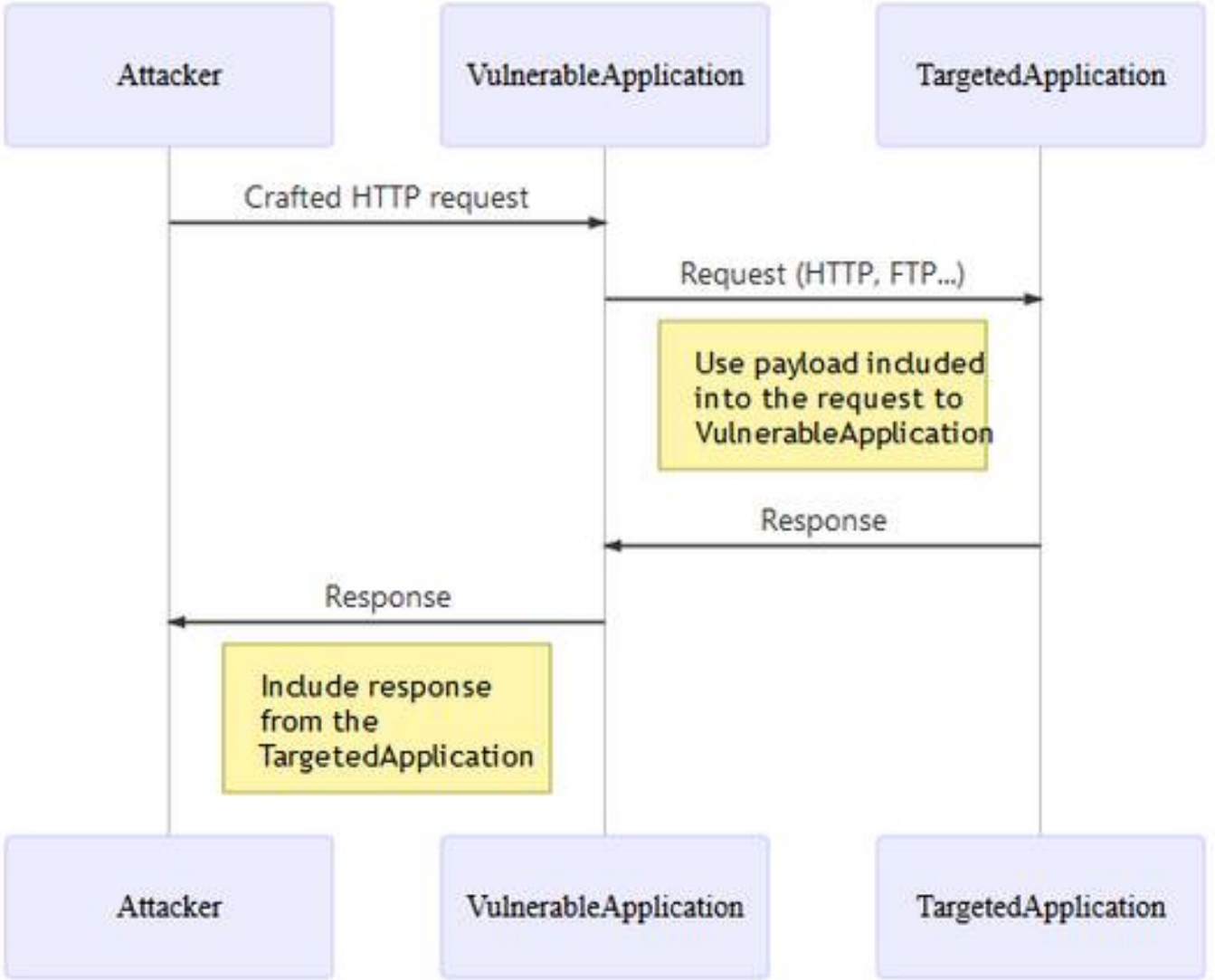
Countdown

Cryptographic Failures

Broken Access Control

#10 Server-Side Request Forgery (SSRF)

- Occurs whenever a web application is fetching a remote resource without validating the user-supplied URL
- Application can send a crafted request to an unexpected target, bypassing firewall, VPN or other ACLs
- CWE-918


Attacker
VulnerableApplication
TargetedApplication
Crafted HTTP request
Request (HTTP, FTP...)
Use payload included into the request to VulnerableApplication
Response
Response
Include response from the TargetedApplication
Attacker
VulnerableApplication
TargetedApplication

[5]

SSRF prevention

Network Layer:

- Segment components in separate networks
- Enforce “deny by default” rule

Application Layer:

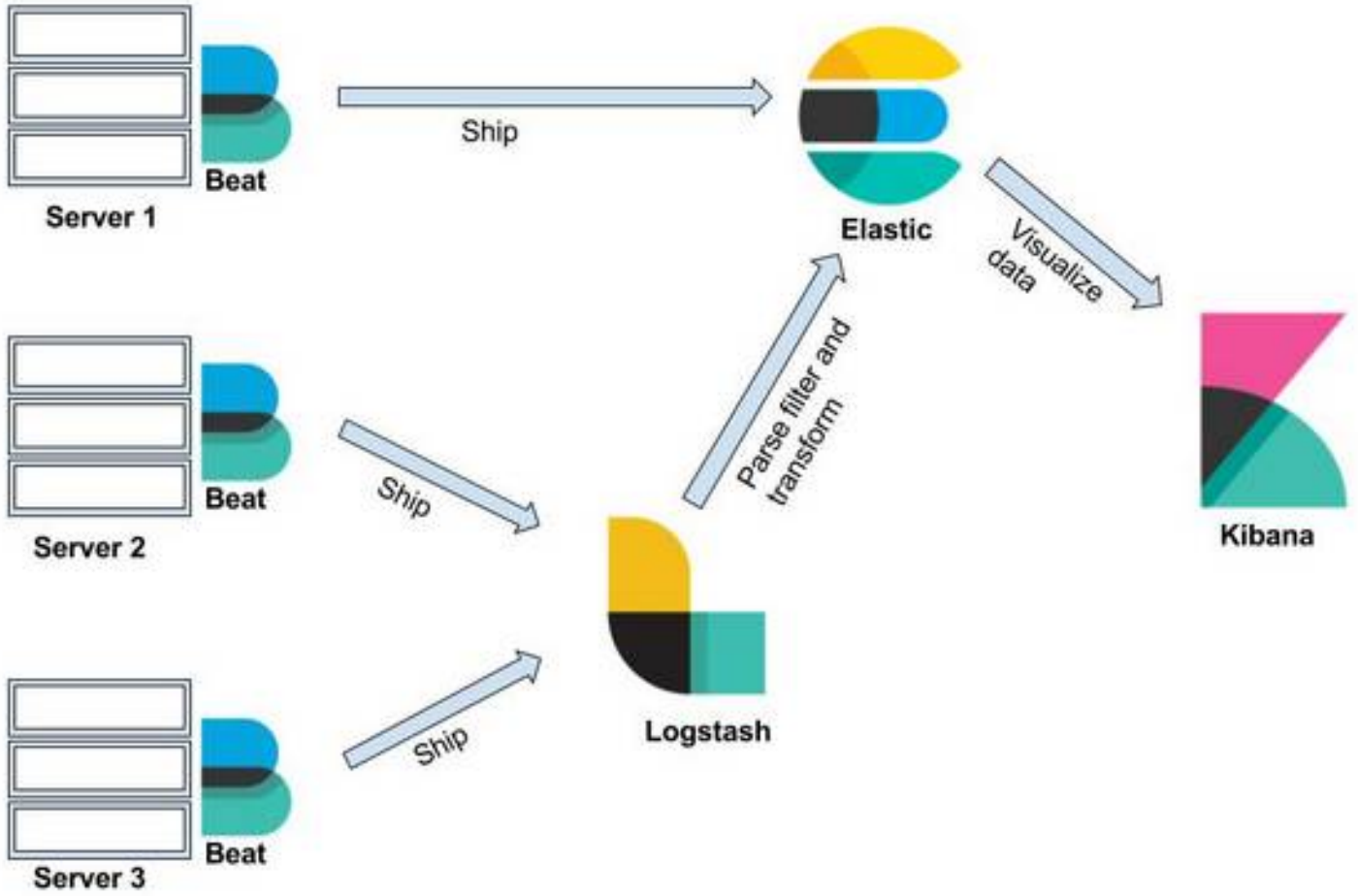
- Sanitize and validate user input

#9 Security Logging and Monitoring Failures

- Auditable events not logged...
- ...or logged not meaningful events
- Inadequate severities (e.g. errors being warnings)
- Unclear messages
- Logs stored only locally
- Lack of log analysis mechanisms
- CWE-778, CWE-223

Road to better logging...

- Ensure all security events are logged
- Ensure all high-value transactions are logged
- Use consistent log format across application
- Store logs in a separate system
- Consider using log analytics tools (e.g. ELK Stack)


Server 1
Beat
Ship
Elastic
Visualize data
Server 2
Beat
Ship
Parse filter and transform
Kibana
Server 3
Beat
Ship
Logstash

#8 Software and Data Integrity Failures

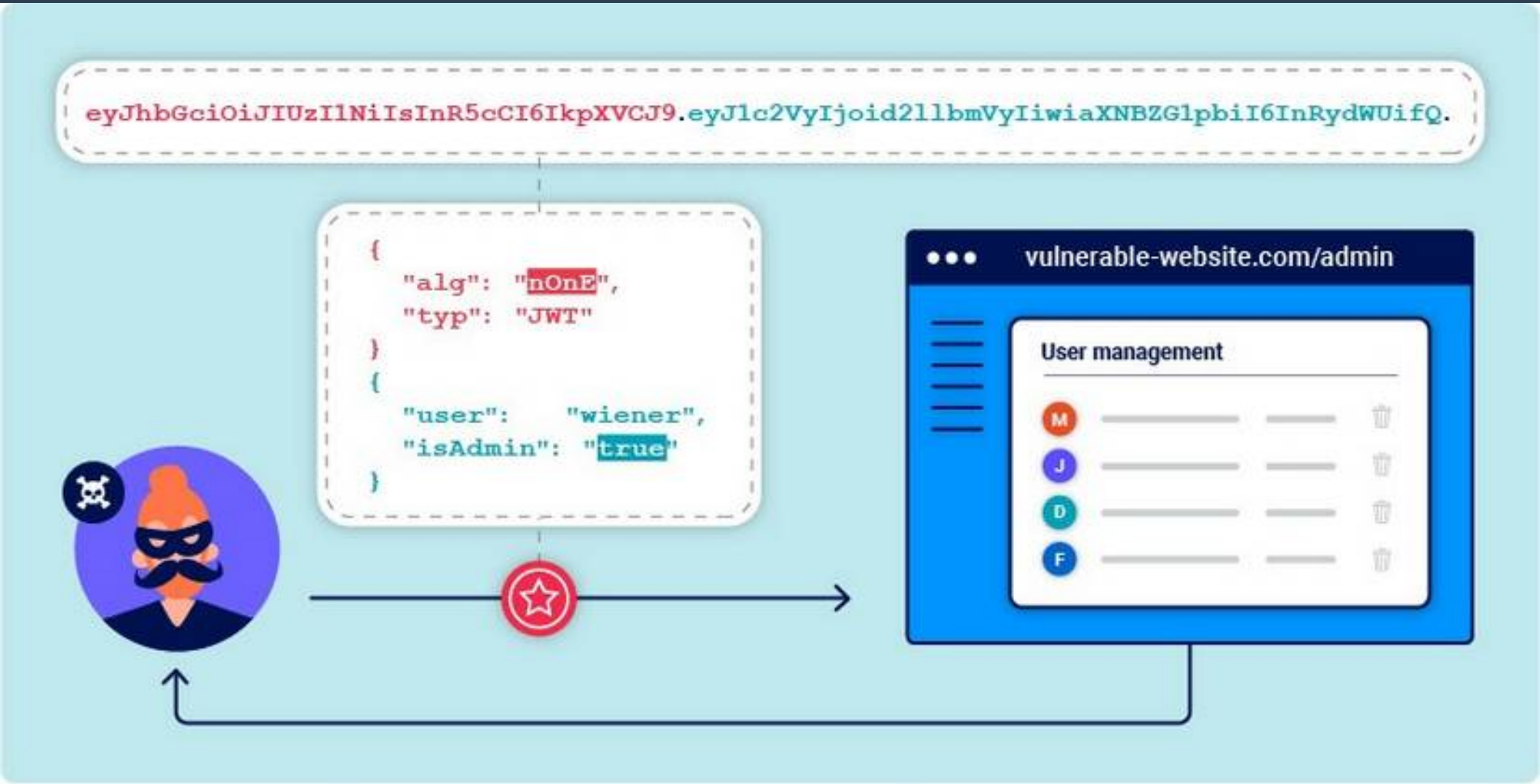
- **Relying on data or components we cannot prove integrity**
  - e.g. using front-end library by including it with an external link
- **When implementing an auto-update feature, not using digital signatures and/or SHA sum for verification**
- **Supply-chain attack**
- **Example: SolarWinds**

Prevention

- Use valid digital signatures
- Ensure libraries and dependencies are downloaded from a trusted source
- Use least possible privilege approach

#7 Identification and Authentication Failures

- **Permits automated attacks for login actions**
  - Using list of valid credentials to check permissions
  - Brute-force passwords
  - Allow to use weak phrases like “Admin1”
- **Stores passwords in a plain text**
- **Exposes session identification in URLs**
- **Improper invalidation of expired sessions**
- **Example: https://portswigger.net/web-security/jwt**

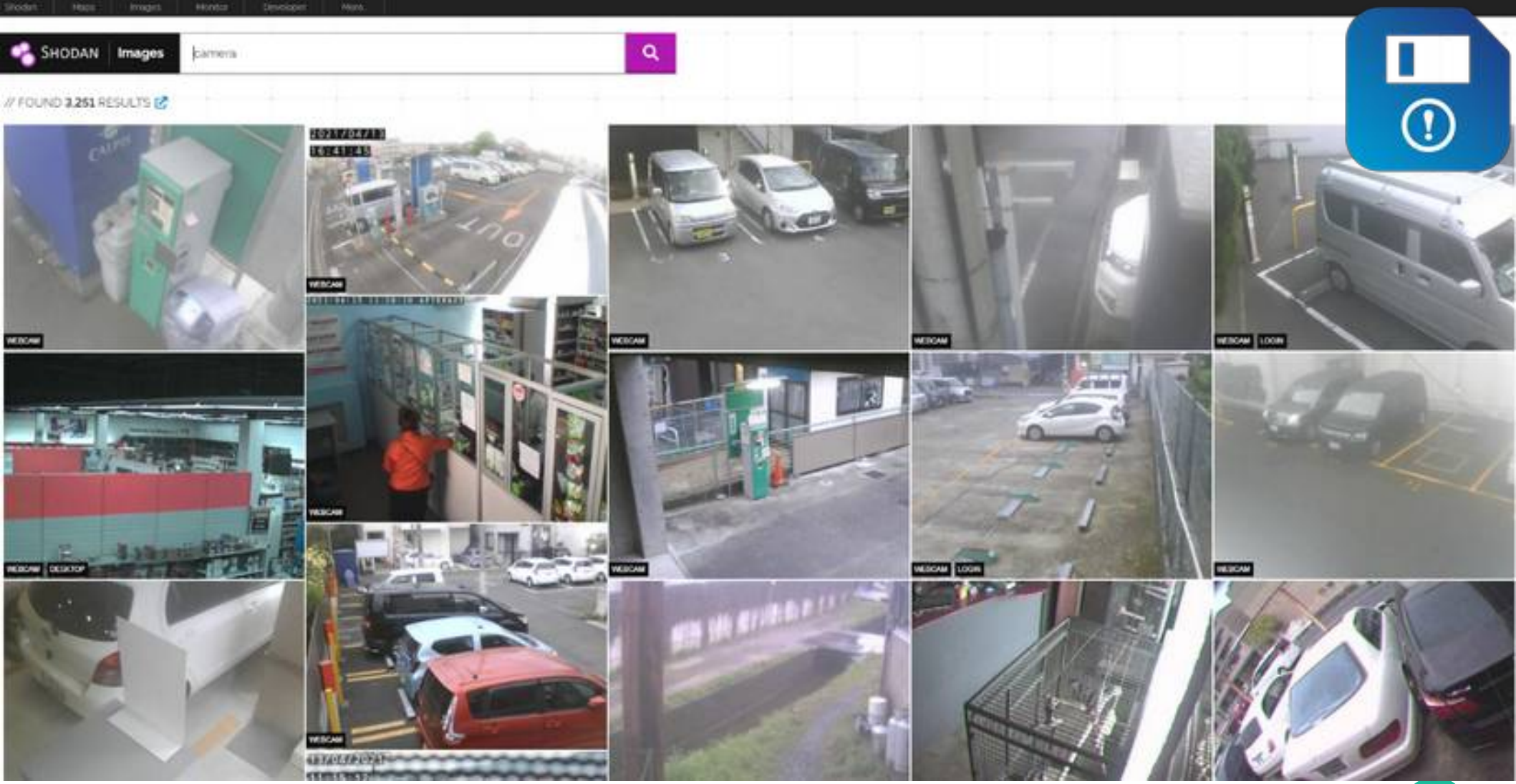

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoid2llbmVyIiwiaXNBZG1pbiI6InRydWUifQ.
{
"alg": "nOnE",
"typ": "JWT"
}
{
"user": "wiener",
"isAdmin": "true"
}
vulnerable-website.com/admin
User management
M
J
D
F

Avoiding identification and auth failures

- Implement MFA where possible
- Do not ship default credentials
- Enforce using good passwords
- Limit/throttle failed login attempts
- Store session IDs on a server side

#6 Vulnerable and Outdated Components

- No track of used dependencies versions
- Lack of scan of used dependencies
- Using unpatched version of an OS
- Lack of maintenance (especially IoT devices)


SHODAN
Images
camera
// FOUND 3,251 RESULTS

Prevention

- Remove unused components, keep other up to date
- Continuously scan your dependencies (e.g. GitHub does this automatically now)
- Obtain components from official sources (mostly)
- If a component is not maintained by its original author, consider creating a custom patch or reimplement feature without it

#5 Security Misconfiguration

- Improperly configured permissions
- Unnecessary features enabled
- Using default credentials
- Disabled security updates
- Server not sends security headers when possible
- Examples: https://brightsec.com/blog/misconfiguration-attacks/

Security ~~Mis~~configuration

- Audit used and configured permissions
- Disable/remove unused features
- Always change default credentials
- Keep OS software up to date
- Configure HTTP Security Header for your application

#4 Insecure Design

- Most fuzzy category describing all architectural weaknesses possible to introduce
- General output: think about application security eagerly and through all the development, and maintenance process

#3 Injection

- Probably we all know this
- This was on the list back in 2003, 2023 still valid :(
- All kind of introducing data other than expected
- Lack of sanitizing, filtering user inputs

Fixing injection is easy?

- General idea: validate user input
- Use well-tested libraries for checking user inputs
- Minimize opportunities to introduce a raw parameter from a user into a query

#2 Cryptographic Failures

- Is data really encrypted during transfer?
- Use of weak, compromised algorithms
- Lack of certificate validation
- Insecure modes (e.g ECB in AES)
- Use of old hash functions
- Too eager decryption

Scenario #1

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

Scenario #2

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, and steals the user's session cookie. The attacker then **replays this cookie and hijacks the user's (authenticated) session**, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g., the recipient of a money transfer.

Scenario #3

- https://github.com/wybory2014/Kalkulator1
- PL ONLY: https://niebezpiecznik.pl/post/caly-swiat-oglada-i-komentuje-kod-zrodlowy-obslugujacy-polskie-wybory/

Prevention

- Encrypt all sensitive data at rest
- Enforce TLS and use features making harder to downgrade secure session e.g HSTS in HTTPS
- Don’t use insecure protocols like FTP to transfer data
- Always validate certificates validity (validity date, issuer, CRLs, etc.)
- Classify data transferred by application

#1 Broken Access Control

- Violation of least possible privilege principle / deny by default for resources which should be protected
- Bypassing access control check
- Insecure direct object references
- Privilege escalation
- CORS
- Hidden but publicly accessible resources

How to prevent BAC?

- Deny by default for non-public resources
- Avoid exposition of specific resources (like .git directory)
- Implement resource ownership for data to avoid editing data owned by an other user
- Disable web directory listing
- Log all authentication events, alert admins when needed
- Use tokens (e.g. JWT) properly (short lived / revocation)

Summary

Sources

[1] OWASP Top 10, https://owasp.org/www-project-top-ten/ (accessed 2023-10-26)

[2] OWASP Top 10 – 2021, https://owasp.org/Top10/ (accessed 2023-10-26)

[3] CVE, https://www.cve.org/ (accessed 2023-10-27)

[4] CVSS Specification Document, https://www.first.org/cvss/v4.0/specification-document(accessed 2023-10-27)

[5] OWASP Cheat Sheet Series, https://cheatsheetseries.owasp.org/index.html (accessed 2023-10-28)

[6] PortSwigger - Server-side request forgery (SSRF), https://portswigger.net/web-security/ssrf (accessed 2023-10-28)