Haavoittuvuustutkimus sivustolle www.example.com

Tämä on aito raportti, ainoastaan tutkimuksen kohde on piilotettu.

Raportissa on kaksi osaa:

- *yhteenveto*
- *yksityiskohtaiset löydökset*

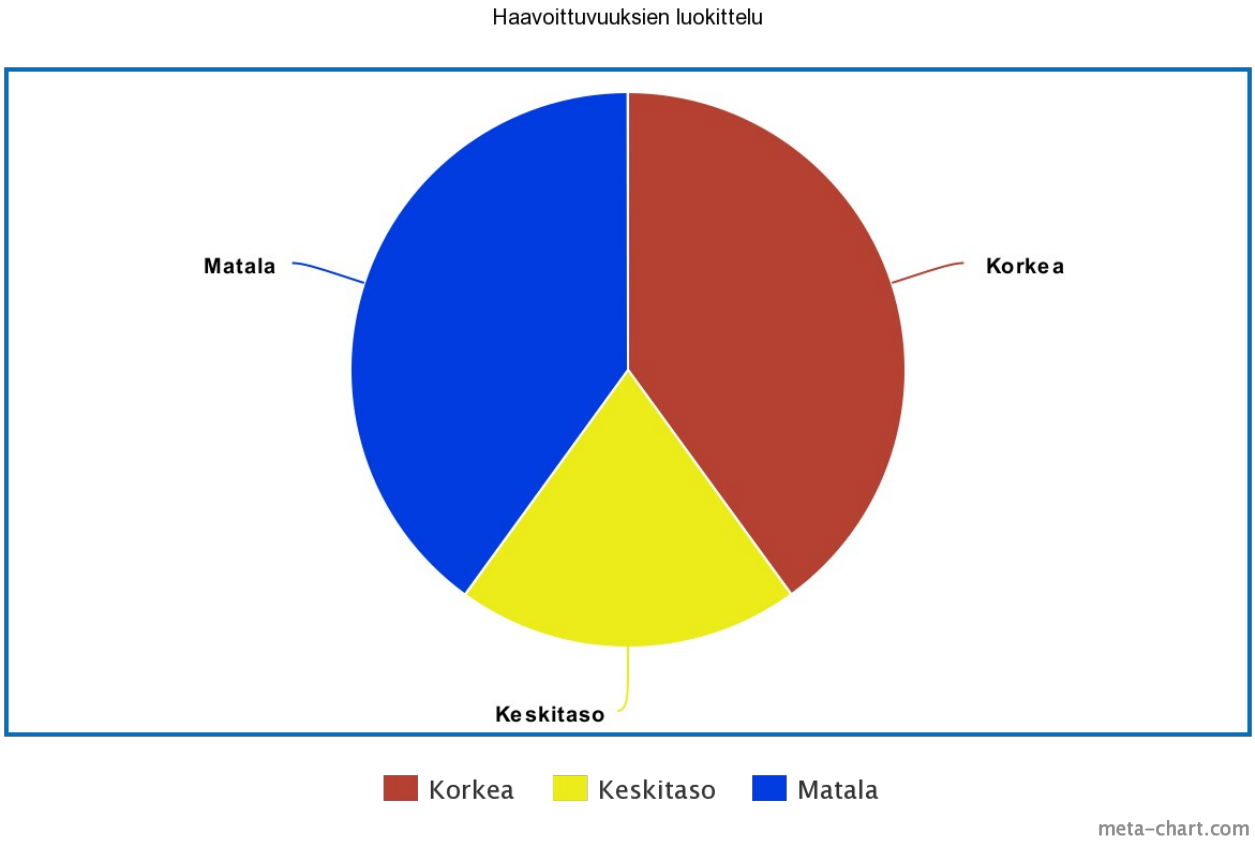
Oletuksena yhteenveto on suomeksi ja yksityiskohtaiset raportit englanniksi esimerkiksi kansainvälisiä kehittäjiä varten. Raportti voidaan haluttaessa laatia kokonaan suomeksi tai englanniksi.

Yhteenveto

Sivustolla on havaittu useita haavoittuvuuksia, joista kaksi on kriittistä.

Havaittujen haavoittuvuuksien lukumäärä ja luokittelu

Vakavuusaste	**Havaittu, raportoitu**	**Haavoittuvuuksien lukumäärä**
KORKEA	1 063	2
KESKITASO	2/1 *)	1
MATALA	318	2
INFORMATIIVINEN	36	ei analysoitu



Vakavuusaste tarkoittaa haavoittuvuuden luokittelua:

- Korkea: kriittiseksi arvioitu haavoittuvuus, jonka hyväksikäyttäminen voi aiheuttaa vakavia ongelmia
- Keskitaso:

Havaittu yhteensä tarkoittaa montako kertaa tämän tason tietoturvaongelmia on havaittu sivustolla. Tämä on suoraan analyysityökalun raportoima määrä.

Haavoittuvuuksien lukumäärä on asiantuntijamme analyysi siitä, montako erillistä haavoituvuutta sivustolla on. Esimerkiksi, on mahdollista, että sama haavoittuvuus esiintyy jokaisella yksittäisellä sivuston sivulla. Yksi ja sama haavoittuvuus voidaan siis havaita useita kertoja.

*) Merkintä 2/1 tarkoittaa, että työkalu on raportoinut 2 haavoittuvuutta, näistä toinen on asiantuntija-analyysissamme havaittu aiheettomaksi.

Yksityiskohtainen analyysi

Issue: **SQL injection**
Severity: **High**
Confidence: **Firm**
Host: **https://www.example.com**
Path: **/aaa-bbbb-ccc-dd//**

Issue: **SQL injection**
Severity: **High**
Confidence: **Firm**
Host: **https://www.eample.com**
Path: **/aaa-bbb-ccc-ddd/**

Issue detail

The **domain** JSON parameter within the **twk_uuid_59bfbef5c28eca75e4620b5d** cookie appears to be vulnerable to SQL injection attacks. The payloads **' and '6740'='6740** and **' and '7387'='7389** were each submitted in the domain JSON parameter within the twk_uuid_59bfbef5c28eca75e4620b5d cookie. These two requests resulted in different responses, indicating that the input is being incorporated into a SQL query in an unsafe way.

Expert opinion (available in premium service package only)

- This appears to be **false positive,** actual difference in reponses is only in timestamp and one byte difference in content-length.

Issue: **Cross-site scripting (DOM-based)**
Severity: **High**
Confidence: **Tentative**
Host: **https://www.example.com**
Path: **/aaa/bbb/**

Issue detail

The application may be vulnerable to DOM-based cross-site scripting. Data is read from **document.location** and passed to **jQuery()**.

Static analysis

Data is read from **document.location** and passed to **jQuery()** via the following statements:

- var fullurl = document.location.toString();
- var anchor_arr = fullurl.split(/#(?!\!)/);
- anchor = anchor_arr[0];
- var anchor_offset = jQuery('#' + anchor).offset();

- Web Security Academy: Cross-site scripting
- Web Security Academy: DOM-based cross-site scripting

Issue remediation

We recommend checking jQuery version and updating to newest

Issue: **HTTP request smuggling**
Severity: **Medium**
Confidence: **Firm**
Host: **https://www.example.com**
Path: **/tag/xxxxxxxx/**

ot/

Issue detail

The web server responded with a **Connection: close** header when sent a request with both **Transfer-Encoding** and **Content-Length** headers, but **Connection: keep-alive** when only the **Transfer-Encoding** header was sent. This suggests that the front-end server may be disabling back-end connection reuse when it receives ambiguous requests, in order to mitigate request smuggling attacks. Please note that this mitigation does not prevent request tunnelling.

Issue remediation

You can resolve all variants of this vulnerability by configuring the front-end server to exclusively use HTTP/2 when communicating with back-end systems. Alternatively, you could ensure all servers in the chain run the same web server software with the same configuration. Disabling back-end connection reuse is likely to reduce the impact of this vulnerability, but does not mitigate all possible exploits.

Specific instances of this vulnerability can be resolved by reconfiguring the front-end server to normalize ambiguous requests before routing them onward. Alternatively, you could configure the back-end server to reject the message and close the connection when it encounters an ambiguous request.

Issue: **Client-side HTTP parameter pollution (reflected)**
Severity: **Low**
Confidence: **Firm**
Host: **https://www.example.com**
Path: **/cccc-dddd-eeeee/**

Issue detail

The name of an arbitrarily supplied URL parameter is copied into the response within the query string of a URL.

The payload **pgu&esp=1** was submitted in the name of an arbitrarily supplied URL parameter. This input was echoed as **pgu&esp=1** within the "href" attribute of an "a" tag.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary query string parameters into URLs in the application's response.

Issue remediation

Ensure that user input is URL-encoded before it is embedded in a URL.

Issue: **Open redirection (DOM-based)**
Severity: **Low**
Confidence: **Tentative**
Host: **https://www.example.com**
Path: **/blogi/page/3/**

Issue detail

The application may be vulnerable to DOM-based open redirection. Data is read from **document.referrer** and passed to **fetch.body**.

Issue remediation

The most effective way to avoid DOM-based open redirection vulnerabilities is not to dynamically set redirection targets using data that originated from any untrusted source. If the desired functionality of the application means that this behavior is unavoidable, then defenses must be implemented within the client-side code to prevent malicious data from introducing an arbitrary URL as a redirection target. In general, this is best achieved by using a whitelist of URLs that are permitted redirection targets, and strictly validating the target against this list before performing the redirection.

Issue: **Open redirection (DOM-based)**
Severity: **Low**
Confidence: **Tentative**
Host: **https://www.example.com**
Path: **/xxxxx/aaaa-bbb.pdf**

Issue detail

The application may be vulnerable to DOM-based open redirection. Data is read from **location.href** and passed to **fetch.body**.

Issue remediation

The most effective way to avoid DOM-based open redirection vulnerabilities is not to dynamically set redirection targets using data that originated from any untrusted source. If the desired functionality of the application means that this behavior is unavoidable, then defenses must be implemented within the client-side code to prevent malicious data from introducing an arbitrary URL as a redirection target. In general, this is best achieved by using a whitelist of URLs that are permitted redirection targets, and strictly validating the target against this list before performing the redirection.

Issue: **Open redirection (DOM-based)**
Severity: **Low**
Confidence: **Tentative**
Host: **https://www.example.com**
Path: **/tag/xxxxxx/**

Issue detail

The application may be vulnerable to DOM-based open redirection. Data is read from **document.cookie** and passed to **fetch.body**.

Issue remediation

The most effective way to avoid DOM-based open redirection vulnerabilities is not to dynamically set redirection targets using data that originated from any untrusted source. If the desired functionality of the application means that this behavior is unavoidable, then defenses must be implemented within the client-side code to prevent malicious data from introducing an arbitrary URL as a redirection target. In general, this is best achieved by using a whitelist of URLs that are permitted redirection targets, and strictly validating the target against this list before performing the redirection.