UNIVERSITY
OF TRENTO

The Nonce-nce of Web Security
An Investigation of CSP Nonces Reuse

Matteo Golinelli, Francesco Bonomi, and Bruno Crispo

University of Trento, Italy
matteo.golinelli@unitn.it, francesco.bonomi@hotmail.it,
bruno.crispo@unitn.it

Background: Content Security Policy

Web security mechanism that **prevents the exploitation** of XSS vulnerabilities

Can be specified

- In the `Content-Security-Policy` response header
- In a `<meta>` HTML tag

Enables websites to **whitelist sources** for JavaScript code, images, CSS files, …

Background: CSP & Inline Scripts

By default, CSP **blocks all inline scripts**

- **Hashes** allow scripts which hash is included in the policy
  - `Content-Security-Policy: script-src 'sha256-<HASH>'`
- **Nonces** allow scripts with a nonce attribute matching the one specified
  - `Content-Security-Policy: script-src 'nonce-cmFuZG9t'`

```
<script nonce="cmFuZG9t">
    console.log("This will execute");
</script>
<script>
    console.log("This will *not* execute");
</script>
```

Background: CSP Nonces

nonce = a number **used only once**

According to the specification, CSP nonces should be

- **Unique** for each HTTP response
- Generated using a **cryptographically secure** random number generator
- At least **128 bit long**

The Issue: Reusing the same nonce is bad

The Content Security Policy prevents the exploitation of XSS vulnerabilities

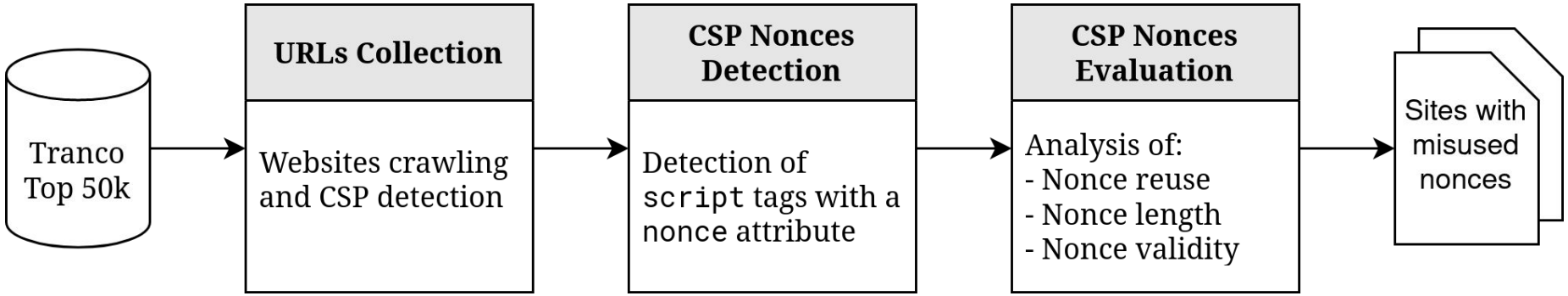
- But **mistakes and oversights** in its implementation might render it ineffective while giving website operators a false sense of security

Goal: Detect & Measure Nonces Misuse

Large-scale analysis on the Tranco Top 50k to detect

- **Nonces reuse**: used in more than one HTTP response
- **Short nonces**: shorter than 128 bit
- **Invalid nonces**: presenting invalid characters outside of the base64 encoding

Methodology



Methodology: Nonces Reuse

```
response1 <- HTTP.get(URL)
nonce1 <- response1.nonce

if nonce1 is not None:
    response2 <- HTTP.get(URL)
    nonce2 <- response2.nonce

    if nonce2 is not None and nonce2 == nonce1:
        print("Nonce reused")
    else:
        print("Nonce not reused")
```

Analysis: Reuse Causes

We attribute nonces reuse to a **web cache** or to the **server-side code** by

1. Checking if all the responses include the same nonce value
2. Using **Cache-Busting**
3. Using the **Cache Header Heuristics**

Analysis: Cache Busting

Receive a **fresh copy** of the response, instead of a cached one

- Add a random parameter to the **query string**

`https://site.com/` ➪ `https://site.com/?ran=dom`

↪ This works when a web cache includes the query string in the **cache key**

Analysis: Cache Header Heuristics

Check if the response is coming from the cache or from the origin

Lookup of the response headers to check for **cache status headers**

- Headers applied by web caches to communicate if a response is cached or not
  - ↪ `X-Cache: HIT` when cached
  - ↪ `X-Cache: MISS` when **not** cached

Analysis: Session Analysis

We check if a reused nonce is **bound to a single session** by

1. Issuing an HTTP request without providing the previously stored cookies to simulate a new visitor
2. Checking if the nonce value is different

Results: CSP Adoption

More than one in four websites that use nonces, **reuses** them in some way

Total sites using CSP	**10034**	
enforcement mode	8946	(89.2%)
report-only mode	1088	(10.8%)
Sites with CSP nonces	2271	(22.6%)
Sites reusing CSP nonces	598	(6.0%)

Results: Nonces Misuse

Total sites reusing nonces	**598**	
due to a cache	256	(42.8%)
server-side code	342	(57.2%)
in the same session	37	(6.2%)
in different sessions	561	(93.8%)

- **Due to a cache**: an attacker can only exploit DOM XSS vulnerabilities
- **Same session**: an attacker must steal a nonce to bypass the CSP
- **Different sessions**: an attacker can easily obtain a nonce

Results: Length & Invalid Nonces

Total sites using nonces	**2271**	
Sites with a short nonce (<22)	501	(22.1%)
Sites with invalid characters in the nonce	8	(0.4%)

- **Short nonces** can theoretically be brute-forced
- **Invalid nonces** are rejected by browsers, causing a self-DoS

Limitations & Future Work

We do not investigate the **randomness of the nonces**

- Analysis of the entropy

No analysis of the inline **JavaScript** included in the pages

- If they use untrusted data, the CSP is useless

Tests performed only with a **single IP address**

- Find nonces bound to IP addresses (if any)

Conclusions

Reusing the same nonce is

- ↪ in some cases, the **same as allowing all inline scripts**
- ↪ in others, a severe **relaxation of the policy**

Implementing a **proper nonce-based policy** is a complex task

- ↪ but is the only way to be **fully protected against XSS**

Extra

Extra: Distribution on the Tranco Top 50k

Distribution of websites that have a nonce-based CSP (in blue), and the subset of those which reuse a CSP nonce (in red) with respect to their ranking in the Tranco Top 50k.