harbourline.example

Web App Scan — Crawl & Attack | 14 October 2026, 06:40 UTC

F

Overall grading 0/100

Input on the shipment tracking page reaches the database unescaped.

Grade driven by: Input is placed directly into a database query

1	2	1	3	1
CRITICAL	HIGH	MEDIUM	LOW	INFO

About this check

What we checked

We do the whole of the crawling scan first — follow the links, record every address, parameter and form, and read what comes back. Then we go back to what we found and TEST it, by submitting values chosen to make a badly built application misbehave. We try to make a database answer a question it was never asked, to get a command run on the server, to get our own markup rendered back to another visitor, to read a file outside the application's own folder, and to make the application fetch an address of our choosing. We try published default passwords on the sign-in forms we are able to submit to — at most three per account name, so we cannot lock out a real administrator. A form that carries an anti-forgery token is left alone and reported as NOT TESTED rather than as clean, and on a modern framework that is most login forms. Where you have given us two sign-ins AND told us which of them is the higher-privileged one, we ask for the same pages as each and compare the answers. Two accounts you have not ranked, or two at the same level, are not compared at all — we will not invent an order between accounts you consider equal — and the report says so rather than reporting nothing found. Two things follow from all that and are said plainly rather than buried. THIS SCAN CHANGES DATA on the application it is pointed at: where a form stores what it is given, we store an inert marker in it, and the report lists the places a marking rule submitted to. One gap in that list we would rather state than have you discover — a file left by the upload test is not yet enumerated there. And we deliberately submit NOTHING to forms that look destructive — a delete, a cancel, a reset. Those stay on the map and the report counts them; they are simply never given a value of ours. Point this at a target you are ready to have tested.

Why it matters

Reading an application tells you what it exposes. Only submitting to it tells you whether it mishandles what it is given — and that is where the breaches come from. A form that passes what you type straight into a database query, a page that renders another visitor's markup, an address that hands over another customer's record because nothing checked whose it was: none of these is visible from the outside, and none of them appears in a scan that only looks. They are also the findings that matter most, because each one is a working way in rather than a weakness somebody might combine with something else.

How to read the grade

The letter is driven by the worst finding in the run, and the report names which one drove it. Everything the ATTACKING pass claims, it demonstrates — a value that came back rendered, a database error that appears only in the tampered request, an address that answered for an account that should not have reached it — so an attacking finding is not a suspicion we are passing on to you. Be aware that this run also carries the whole of the crawling scan's reading, and a few of those are inferred rather than demonstrated, so the letter can be driven by one of them; every finding says which kind it is. An A means everything this scan was able to test answered the way it should. It does not mean the application is safe: what we could not test is reported as not tested rather than counted as a pass, which is why a run that stopped at its

budget, a sign-in we could not complete, or a target with only one account says so in as many words. Checks that need our out-of-band listener report that they were not checked when it is not configured, rather than reporting clean.

Findings (8)

#	Finding	Risk
1	SQL Injection — Input is placed directly into a database query	CRITICAL
2	Reflected XSS — Input is reflected with the characters needed to inject markup	HIGH
3	Directory Listing — Anyone can browse the files in one of your folders	HIGH
4	Content Security Policy Allows Inline Script — Your content rules allow scripts written directly into the page	MEDIUM
5	Cookie Without HttpOnly Flag — The cookie PHPSESSID can be read by scripts in the page	LOW
6	Missing Security Header — Missing security header: X-Content-Type-Options	LOW
7	Missing Security Header — Missing security header: Referrer-Policy	LOW
8	Software Version Disclosure — This application announces the exact software and versions it runs	INFO

CRITICAL

1. SQL Injection — Input is placed directly into a database query

CWE-89

A value this page accepts is put straight into a database query instead of being passed to it separately. We sent a single harmless character — a quote — and MySQL or MariaDB answered with a syntax error; the same request without that one character came back as an ordinary page, which is what makes the error ours rather than the page's own. Anyone who can reach this page can therefore change what the query asks for: read records they are not entitled to, alter them, or delete them. We stopped at proving the parameter reaches the database and did not run any query of our own.

Confirmed - tested directly; the server behaved this way.

For technical readers

A value from the request is concatenated into a SQL statement, so the database parses the attacker's punctuation as syntax rather than reading it as data — which reaches every row that connection can read, and on several engines the file system and command execution as well. The repair is parameterised statements everywhere, including the places that tempt people to build strings: an ORDER BY column or a table name cannot be bound as a parameter and needs an allowlist of permitted values instead. Escaping routines and request filtering reduce the surface without closing it.

What we sent, and what came back

The request with one quote added to the tracking reference

```
GET /track?ref=HL-20931' HTTP/1.1
Host: harbourline.example
User-Agent: Mozilla/5.0 (ClickScan)
```

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/html

<h1>Error</h1><p>You have an error in your SQL syntax; check the manual that corresponds to your MySQL
server version for the right syntax to use near ''HL-20931''' at line 1</p>
```

The same request without the quote, for comparison

```
GET /track?ref=HL-20931 HTTP/1.1
Host: harbourline.example
User-Agent: Mozilla/5.0 (ClickScan)
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<h1>Shipment HL-20931</h1><p>Status: In transit, Rotterdam</p>
```

What to do

Stop building this query by joining text together. Use the database library's parameter binding — prepared statements, bound or named parameters — so values are sent to the database separately from the instruction and can never change its meaning. Escaping quotes is not an adequate substitute and fails on the numeric and identifier cases. Then check whether the same pattern is used elsewhere in the application, because it is rarely in one place, and turn off detailed database errors in production so the next defect does not describe itself to a stranger.

HIGH

2. Reflected XSS — Input is reflected with the characters needed to inject markup

CWE-79

A value this page accepts is sent straight back into the page with the punctuation an attacker needs left intact. We submitted four harmless characters — a less-than sign — and they came back exactly as sent, in the page's text, where they can start a new tag, rather than being converted into their harmless display form. That is what makes a crafted link dangerous: someone who sends your visitor a link with script in it has it run inside your site, as that visitor, with their session — which is enough to read what they can read or act as them. We proved the characters survive and deliberately did not send any script.

Confirmed - tested directly; the server behaved this way.

For technical readers

Input from the request is returned in the response without being encoded for the context it lands in, so a crafted value stops being data and becomes markup the browser parses — script then runs in this origin's security context, with whatever the signed-in session can reach. The repair is output encoding chosen by SINK rather than by input: HTML body, attribute value, JavaScript string and URL each need a different one, applied where the value is rendered. A Content-Security-Policy that forbids inline script limits what an injection achieves and is worth having, but it is defence in depth, not the fix.

What we sent, and what came back

Our marker, sent in the search box

```
GET /search?q=cs7f3e%3C%22%27%3E HTTP/1.1
Host: harbourline.example
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<h2>Results for cs7f3e<"'></h2>
```

What to do

Encode values on the way OUT, according to where they are being put: HTML-encode for page text, attribute-encode inside a tag, and use the framework's own escaping rather than hand-written replacement. Most template engines do this by default and are usually switched off for the field in question, so look for the "raw" or "unescaped" marker on this value first. A Content-Security-Policy limits the damage and is not a substitute for encoding.

HIGH

3. Directory Listing — Anyone can browse the files in one of your folders

CWE-548

Opening https://harbourline.example/uploads/ returns an automatic list of the files in that folder rather than a page. Whoever looks can read every filename there, including backups, exports and anything else left behind that was never meant to be found.

Inferred - read from published records or response headers, not tested directly.

For technical readers

The server renders an index of the directory instead of refusing the request, so every file in it can be enumerated without guessing a single name — which matters most for the files nothing links to, because that is where backups, exports and configuration collect. The repair is to switch auto-indexing off; removing the files matters more than removing the listing, which only made them easy to find.

What to do

Switch off automatic directory listing for this folder on your web server, and move anything in it that should not be public.

MEDIUM

4. Content Security Policy Allows Inline Script — Your content rules allow scripts written directly into the page

CWE-693

The script rule allows inline script. That is the exact thing content rules exist to stop: an attacker who can place markup on the page can place a working script with it, and the browser will run it.

Confirmed - tested directly; the server behaved this way.

What we sent, and what came back

The policy every page is served with

```
GET / HTTP/1.1
Host: harbourline.example
```

```
HTTP/1.1 200 OK
Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline'; frame-ancestors 'self'
```

What to do

Remove 'unsafe-inline' and give each of your own inline scripts a nonce or a hash. A browser ignores 'unsafe-inline' once a nonce or hash is present, so you can keep it for very old browsers without weakening the rule.

LOW

5. Cookie Without HttpOnly Flag — The cookie PHPSESSID can be read by scripts in the page

CWE-1004

Your site set a cookie named PHPSESSID without marking it HttpOnly, so any JavaScript running in the page can read it. If somebody ever gets a script onto one of your pages — through a comment box, an advert or a compromised third-party file — they can take that cookie and use it as the visitor.

Confirmed - tested directly; the server behaved this way.

For technical readers

The cookie is readable from JavaScript, so any script running on the page can read it and send it elsewhere — including one injected through cross-site scripting, and one shipped by a dependency that was compromised upstream. The repair is `HttpOnly` on every cookie the page's own code does not need to read, which for a session cookie is always.

What to do

Add the HttpOnly attribute when the cookie is set. Only leave it off for a cookie your own scripts genuinely have to read.

LOW

6. Missing Security Header — Missing security header: X-Content-Type-Options

CWE-693

Without this header some browsers guess ("sniff") a response's type and may run a file as script that was never meant to be. We read 151 pages and none of them carried X-Content-Type-Options.

Inferred - read from published records or response headers, not tested directly.

For technical readers

A response header the browser would have used to constrain the page is absent, so a defence that costs nothing to deploy is not in place. These headers rarely stop an attack on their own; what they do is limit what an injection or a framing attack ACHIEVES, which is why they matter most on the day something else goes wrong. The repair is to set them at the edge, so every response carries them rather than each route remembering.

What to do

Add X-Content-Type-Options: nosniff.

LOW

7. Missing Security Header — Missing security header: Referrer-Policy

CWE-693

Without a Referrer-Policy the full URL of your pages (which can carry tokens or private paths) is sent to every third-party site your pages link to or load from. We read 151 pages and none of them carried Referrer-Policy.

Inferred - read from published records or response headers, not tested directly.

For technical readers

A response header the browser would have used to constrain the page is absent, so a defence that costs nothing to deploy is not in place. These headers rarely stop an attack on their own; what they do is limit what an injection or a framing attack ACHIEVES, which is why they matter most on the day something else goes wrong. The repair is to set them at the edge, so every response carries them rather than each route remembering.

What to do

Add Referrer-Policy: strict-origin-when-cross-origin (or stricter).

INFO

8. Software Version Disclosure — This application announces the exact software and versions it runs

CWE-200

Responses name nginx 1.24.0, PHP. We read that from what the application sent us, unprompted. An exact version turns "is anything here exploitable" into a lookup — somebody matches the version against published vulnerabilities for it instead of probing to find out.

Confirmed - tested directly; the server behaved this way.

For technical readers

The response names the product and its version, which is not an exposure in itself but removes the reconnaissance step: an attacker matches the version against published advisories rather than probing to find out. The repair is to suppress the version in responses — and to read the finding as a prompt to check whether the named version is current, which is usually the more important question.

What to do

Stop sending the version. On Apache set ServerTokens Prod and ServerSignature Off; on nginx set server_tokens off; for PHP set expose_php = Off. If a proxy or CDN sits in front, strip the headers there. Then check that the versions you are running are still supported and patched: hiding the banner is not the same as being up to date.

Web application

Mapped from https://harbourline.example/

What this scan covered

We found 212 addresses and fetched 164 of them.
We left 6 outside the scope set for this target, 8 on another host and 22 that are a kind of file a crawl does not follow.
2 addresses did not answer, so we have nothing to report about them.
10 addresses were found and never fetched. Of those, 10 were queued when the crawl stopped.

Application surface

37 places a value can be inserted - parameters, form fields, headers and path segments - were found.
It links to 1 host we did not follow: fonts.cdn.example.

What it looks like it is running

PHP (language) - a cookie named PHPSESSID
nginx 1.24.0 (server) - Server: nginx/1.24.0
Each of these is inferred from what the application volunteers - a header, a cookie name, a script path - rather than established by a probe.

Report generated 17 October 2026, 19:36 UTC. Scan reference d3736037-7019-4ad8-a9c0-0b70cfb9a6dd.