Chaining Cache Poisoning To Stored XSS

Rohan Aggarwal Follow
Jul 28 · 3 min read

One of the benefits of being a developer is that you can guess how stuff is working at the server end. You can try to debug how the developer might have coded a certain functionality or how he might have configured his application, especially in the case where the application is built in some **framework** or **CMS**.

For example, many developers using Rails framework use Scaffold to quickly generate major pieces of application like model, views and controller in a single operation. Besides this, Scaffold also creates JSON API endpoint for each route automatically which is most of the time overlooked by the devs or they forgot to remove this JSON endpoint while pushing to production.

So a dev who removed sensitive information from a normal route, might forget to remove the same from JSON endpoint of that route. Checking such misconfiguration might leak some sensitive data in a rails application. Therefore, having knowledge about the application technology really gives you differential findings than others.

Recently, I came across a Drupal application in a bug bounty program on Hackerone. Since I've used drupal before, I started looking for some common misconfigurations related to drupal. And within 10 minutes, I found one.

If you've developed on drupal before, you might know that it has it's own internal caching system which is **enabled by default**. Easiest way to find whether caching is enabled is to look for **x-drupal-cache** header in the response. So you give unique key(*endpoint* + *parameters*) in request and in response you get **x-drupal-cache: MISS** header but if you request again with that same key and you get **x-drupal-cache: HIT** header in the response, caching is enabled. To learn more about how caching works and its exploitation, read this by albinowax.

After I came to knew that caching is enabled, one of the ways I can exploit is by poisoning the cache with XSS payload. But to do that I need to find an HTTP header that gets reflected in the response. Since drupal just like any other PHP framework sometimes supports X-Original-URL and X-Rewrite-URL headers because of Zend, I tried injecting these headers in request but sadly the application wasn't accepting. What to do now? If nothing works, **try brute-forcing**.

So I used a burp suite extension called Param Miner which will brute force common headers. After few seconds I got a hit. It found a header named **style** which was getting reflected in response. I quickly checked whether it's vulnerable to XSS and it was.

After that, I knew I can easily chain cache poisoning to stored XSS. I created a unique key & added style header with XSS payload and fired the request.

The response with XSS payload is cached for the above unique request. Now whenever someone visits www.redacted.com/?q=admin&liec4897=1, our poisoned response will be served by drupal resulting in Stored XSS.

That’s it. A simple drupal misconfiguration leads to Stored XSS.

TIMELINE

Jun 14, 2019 — Report Submitted

Jun 15, 2019 — Triaged

Jul 12, 2019 — Resolved

Jul 13, 2019 — Rewarded

If you found this post useful in any way, make it useful for others as well by sharing. More coming.

And you can also hit me up on twitter if you have any questions.

Bug Bounty   Cache Poisoning   Xss

419 claps

WRITTEN BY

Rohan Aggarwal

Follow

A noob finding noob bugs to secure the internet from other noobs | Found vulns in Yahoo, Twitter, Matomo,etc | Love doing CTF & HTB | https://nahoragg.github.io

See responses (1)

More From Medium

Also tagged Xs

Stealing JWTs in localStorage via XSS

David Roccasalva in Privasec RED

Sep 10 · 5 min read ★

322

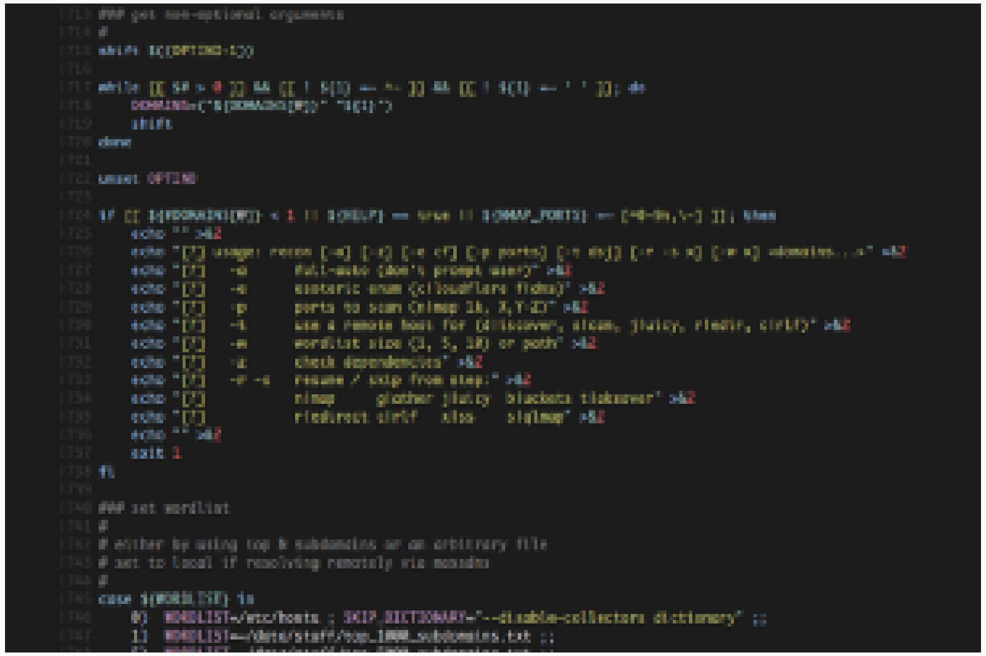
Related reads

Chinese Hackers Back Beijing's Authoritarian Pals

Foreign Policy in Foreign Policy

Jul 30, 2018 · 7 min read ★

87



```
### get non-optional arguments
#
shift $((OPTIND-1))

while [[ $# > 0 ]] && [[ ! ${1} =~ ^- ]] && [[ ! ${1} == ' ' ]]; do
    DOMAINS+=("${DOMAINS[@]}" "${1}")
    shift
done

unset OPTIND

if [[ ${#DOMAINS[@]} < 1 || ${HELP} == true || ${NMAP_PORTS} =~ [^0-9n,\-] ]]; then
    echo "" >&2
    echo "[?] usage: recon [-a] [-c] [-e cf] [-p ports] [-r [illegible]] [-r -s x] [-w x] <domains...>" >&2
    echo "[?]   -a      full-auto (don't prompt user)" >&2
    echo "[?]   -e      esoteric enum (cloudflare fishy)" >&2
    echo "[?]   -p      ports to scan (nmap 1k, X,Y-Z)" >&2
    echo "[?]   -r      use a remote host for ([illegible])" >&2
    echo "[?]   -w      wordlist size (1, 5, 10) or path" >&2
    echo "[?]   -c      check dependencies" >&2
    echo "[?]   -r -s   resume / skip from step:" >&2
    echo "[?]           nmap      [illegible]" >&2
    echo "[?]           [illegible]" >&2
    echo "" >&2
    exit 1
fi

### set wordlist
#
# either by using top N subdomains or an arbitrary file
# set to local if resolving remotely via massdns
#
case ${WORDLIST} in
    0) WORDLIST=/etc/hosts ; SKIP_DICTIONARY="--disable-collectors dictionary" ;;
    1) WORDLIST=~/data/staff/top_1000_subdomains.txt ;;
```

Related reads

Reconnaissance: a eulogy in three acts

europa
Feb 11, 2018 · 8 min read

1.6K

ideas take center stage - with no ads in sight. Watch