Time-efficient assessment of open-source projects for Red Teamers

Pass the Salt 2019

Thomas Chauchefoin (@swapgs)
Julien Szlamowicz (@SzLam_)
Agenda

- Introduction
- Methodology
- Findings
- Disclosure
- Conclusion
Introduction
Synacktiv is a French company focusing on offensive security: manual assessment, source code review, reverse engineering...

Three teams
- Pentest
- Reverse engineering
- Development

We are remote-friendly
Reach us at apply@synacktiv.com or at the social event
WE NEED A SYSADMIN
Context

- Red team assessment: only a fashionable term for “real-world” pentest?
- Big scopes!
  - Limited effort per exposed asset
  - We need to reach the internal network as fast as we can
- Facing the Blue Team
- OSS is not less secure than proprietary software but:
  - Easier to get and deploy in a lab
  - Quicker to assess than an obfuscated / closed product
Case study

- This talk aims at presenting our (sort of) methodology and findings in GLPI
- Hopefully didactic enough to be interesting to people not working in infosec
- Discovered issues were patched several months ago
  - Make sure you’re at least on 9.4.1.1
  - Don’t expose it publicly
- Identified the first day of a 2-weeks Red Team engagement
  - Gave us a good insight on the target’s internal network
"GLPI ITSM is a software for business powered by open-source technologies. Take control over your IT infrastructure: assets inventory, tickets, MDM" (glpi-project.org)

- Mostly supported by Teclib’, editor of Armadito and Uhuru, under GPLv2
- Plugins help adding various features
  - Inventory
  - MDM
    - Software deployment
    - Configuration
GLPI

- Telemetry shows it’s commonly used in France and Brazil
  - 28K pingbacks last year
  - 9K from French IP addresses
- You can add yourself on the website to show you like the project
  - C.N.A.M.T.S, 130K computers and 90K users (2007)
  - Police Nationale, 100K computers (2012)
  - Various government departments
- Seems like an interesting target in our context: let’s break it :-)

SYNACKTIV
Considerations

- During regular pentests, you can be loud and intrusive
  - Exhaustive rather than opportunistic
- During Red Team engagements, the goals change
  - Get a foot in the door ASAP
  - Remain undetected
  - Deep compromise
  - A single entry point is enough
- Time constraint
Methodology
Considerations

What is a good Red Team vulnerability?

- Forget everything about client-side attacks in the first place (except for phishing campaigns)
- No destructive actions
- Low forensic/detection footprints
- No feature breaking or raised exceptions (Sentry is quite popular nowadays)
- Reproducible in our lab first
Replicating the environment

- When assessing OSS, you are never really in blackbox
- Try to replicate an accurate environment
  - HTTP server
  - CGI’s version
  - Product version
- It will be very helpful to
  - Avoid early detection
  - Abuse specific configurations, vulnerabilities or behaviour
- Any information leak is valuable
Assessing the attack surface

- We are only interested in unauthenticated code paths
- PHP applications not using frameworks will often have several scripts directly reachable

Prevented by

- Ensuring a given constant is defined
- User has a session with a given value, etc

In real life, these checks are always forgotten at least once
Assessing the attack surface

Top-Down

User input

Casts/checks

Sanitization

Vulnerable code?

Bottom-Up

User input

Casts/checks

Sanitization

Vulnerable code

$b bla = \$_GET["bla"]

int($bla)
preg_match("/\/<\",$bla)

addslashes($bla)
mysql_real_escape_string($bla)

eval($bla)
system($bla)
dbquery($query,$bla)
Assessing the attack surface

- In practice, we tend to use a hybrid approach when reading source code
  - Find vulnerabilities quickly
  - No need to be exhaustive
- The lab allows performing dynamic analysis and using our blackbox skillset
Assessing the attack surface

- Our colleague @Tiyeuse developed a tool to find reachable files “doing things”
  - Not only declaring classes and functions
  - Not exiting after checking for a constant declared in another file
  - Possibility to add custom patterns to exclude authentication checks

- GLPI had several pre-authenticated vulnerabilities in such files
  - Less code to read
  - Less things to understand
  - Happier auditor :-)

Other tools and tricks

- We don’t have semantic tooling
  - PHP-Parser can still help create a “smart grep”
- RIPS scanner is awesome
  - But a bit expensive for everyday use
- Dumping every DB query to a log file
  - Harder to miss SQL errors (injections)
  - Easier to debug PoCs
- Instrument low-level PHP functions to search for specific behaviours
  - Unbalanced quotes?
- Profilers: fracker, xhprof
Assessing the attack surface

- Create a wrapper around $_GET and $_POST:

```php
class ObjectAccess implements ArrayAccess {
    // ...
    public function offsetExists($key) {
        echo $this->name." -> isset: ".$key."\n";
        return isset($this->items[$key]);
    }

    public function offsetGet($key) {
        echo $this->name." -> get: ".$key."\n";
        return $this->items[$key];
    }
    // ...
}
```

- No need to browse all the includes to find accepted parameters
Approach

- After isolating access control functions, a quick run of *debroussailleuse* gave us the list of reachable files.
  - Still ~400 files left (excluding *vendors/*)
- In theory, files in */scripts/* are protected by a `.htaccess`
- Our target uses *nginx*
  - It’s in the official documentation
  - *AllowOverride* is set to *None* since *Apache 2.3.9*
Findings
Information leak

- Accessing `ajax/telemetry.php` discloses
  - GLPI version
  - GLPI modules
  - PHP version
  - PHP modules
  - Operating system
  - HTTP server
- Enough to start creating a lab
DEMO
SQL injection in `compute_dictionary.php`?

- Digging in `scripts/` yields interesting results
  - `scripts/compute_dictionary.php`

```php
if (isset($_GET['dictionary'])) {
    $rulecollection = RuleCollection::getClassByType($_GET['dictionary']);
    if ($rulecollection) {
        if ($_GET['dictionary'] == 'RuleDictionarySoftware' [...]) {
            $rulecollection->replayRulesOnExistingDB( [...] , $_GET['manufacturer']);
        }
    }
}
function replayRulesOnExistingDB([...], $params = []) {
    [...]
    if (count($items) == 0) {
        //Select all the different software
        $sql = "SELECT DISTINCT `glpi_softwares`.`name`,
    [...]
    if (isset($params['manufacturer']) && $params['manufacturer']) {
        $sql .= " AND `glpi_softwares`.`manufacturers_id` = "" . $params['manufacturer'] . "";
    }
    if ($offset) {
        $sql .= " LIMIT " . intval($offset) . ",999999999";
    }
}
SQL injection in `compute_dictionary.php`?

- But it doesn’t work! :-S
SQL injection in *compute_dictionary.php*?

- The reason lies in *inc/includes.php*

```php
// Security system
if (isset($_POST)) {
    [...]
    $_POST = Toolbox::sanitize($_POST);
}
if (isset($_GET)) {
    $_GET = Toolbox::sanitize($_GET);
}
if (isset($_REQUEST)) {
    $_REQUEST = Toolbox::sanitize($_REQUEST);
}
```
SQL injection in `compute_dictionary.php`?

- **Toolbox::sanitize()** is implemented this way

```php
static public function sanitize($array) {
    $array = array_map('Toolbox::addslashes_deep', $array);
    $array = array_map('Toolbox::clean_cross_side_scripting_deep', $array);
    return $array;
}
```

- **addslashes_deep()**
  - Recursive `mysql_real_escape_string()`

- **clean_cross_side_scripting_deep()**
  - Replaces `< >` by their HTML entities

- **sanitize()** will fail in several cases (it's regex time)
SQL injection in `unlock_tasks.php`

- A hit was found in `scripts/unlock_tasks.php`
- **CVE-2019-10232**

```php
if (isset($_GET['cycle'])) {
    $cycle = $_GET['cycle'];
}

[...]
$crontask = new Crontask();
$cycle = "SELECT `id`, `name` FROM `glpi_crontasks` WHERE `state` = ".Crontask::STATE_RUNNING."' AND unix_timestamp(`lastrun`) + $cycle * `frequency` < unix_timestamp(now())";

[...]
foreach ($DB->request($query) as $task) {
    if (!empty($only_tasks) && !in_array($task['name'], $only_tasks)) {
        echo $task['name'] . ": is still running but not in the whitelist\n";
        continue;
    }
```
SQL injection in *unlock_tasks.php*

- However...
  - The injection doesn’t allow creating users
  - Passwords are hashed with *bcrypt*
    - *PHP_PASSWORD_BRCRYPT_COST = 10*
  - Our 8 1080 Ti GPUs will hardly be enough
- **Need to find another way to get in—let’s inspect the table glpi_users**
  - *name*
  - *password*
  - *last_login*
  - *password_forget_token*
  - *personal_token*
  - *api_token*
SQL injection in *unlock_tasks.php*

- The *Remember me* feature is enabled by default and uses the *personal_token* value
  
  ```
  ["2","$2y$10f10tNcc[...]wmVSULi"]
  [user_id, hash(personal_token)]
  ```

- Several hash algorithms supported
- Leaking a token is enough to log in
- We could also use the API key or reset users’ password
- Any data allowing to authenticate is a secret, they should be stored in the database the same way
DEMO
Abusing the *Remember me* feature

- While looking *Remember Me* feature, its implementation seemed weird
  
  ```php
  if ($CFG_GLPI["login_remember_time"]) {
      $data = json_decode($_COOKIE[$cookie_name], true);
      if (count($data) === 2) {
          list($cookie_id, $cookie_token) = $data;
      }
  }
  ```

- Thanks to `json_decode()`, we can play with types of
  - `$cookie_id`
  - `$cookie_token`
Abusing the *Remember me* feature

```php
$ php -a
php> var_dump(json_decode('["1", 1, null, {}, true, false]'));
array(5) {
    [0]=> string(1) "1"
    [1]=> int(1)
    [2]=> NULL
    [3]=> object(stdClass)#1 (0) {}
    [4]=> bool(true)
    [5]=> bool(false)
}
```
Abusing the *Remember me* feature

Then, our values are used this way

```php
$user = new User();
$user->getFromDB($cookie_id);
$token = $user->getAuthToken();
if ($token !== false && Auth::checkPassword($token, $cookie_token)) {
    $this->user->fields['name'] = $user->fields['name'];
    return true;
} else {
    $this->addToError(__('Invalid cookie data'));
}
```

- `$user->getAuthToken()` creates a new `personal_token` if it doesn’t exist
Abusing the *Remember me* feature

The *personal_token* is then compared with the hash provided in the cookie.

```php
static function checkPassword($pass, $hash) {
    $tmp = password_get_info($hash);
    if (isset($tmp['algo']) && $tmp['algo']) {
        $ok = password_verify($pass, $hash);
    } else if (strlen($hash)==32) {
        $ok = md5($pass) == $hash;
    } else if (strlen($hash)==40) {
        $ok = sha1($pass) == $hash;
    } else {
        $salt = substr($hash, 0, 8);
        $ok = ($salt .sha1($salt.$pass) == $hash);
    }
    return $ok;
}
```
Abusing the *Remember me* feature

The `personal_token` is then compared with the hash provided in the cookie

```php
static function checkPassword($pass, $hash) {
    $tmp = password_get_info($hash);
    if (isset($tmp['algo']) && $tmp['algo']) {
        $ok = password_verify($pass, $hash);
    } else if (strlen($hash)==32) {
        $ok = md5($pass) == $hash;
    } else if (strlen($hash)==40) {
        $ok = sha1($pass) == $hash;
    } else {
        $salt = substr($hash, 0, 8);
        $ok = ($salt . sha1($salt . $pass) == $hash);
    }
    return $ok;
}
```
Abusing the *Remember me* feature

The hashed value to compare is controlled by the attacker (CVE-2019-10233)

```php
static function checkPassword($pass, $hash) {
    $tmp = password_get_info($hash);
    if (isset($tmp['algo']) && $tmp['algo']) {
        $ok = password_verify($pass, $hash);
    } else if (strlen($hash) == 32) {
        $ok = md5($pass) == $hash;
    } else if (strlen($hash) == 40) {
        $ok = sha1($pass) == $hash;
    } else {
        $salt = substr($hash, 0, 8);
        $ok = ($salt . sha1($salt . $pass) == $hash);
    }
    return $ok;
}
```
Abusing the Remember me feature

If the provided hash doesn’t match any well-known algorithms, we need to talk about PHP comparisons

```php
static function checkPassword($pass, $hash) {
    $tmp = password_get_info($hash);
    if (isset($tmp['algo']) && $tmp['algo']) {
        $ok = password_verify($pass, $hash);
    } else if (strlen($hash)==32) {
        $ok = md5($pass) == $hash;
    } else if (strlen($hash)==40) {
        $ok = sha1($pass) == $hash;
    } else {
        $salt = substr($hash, 0, 8);
        $ok = ($salt.sha1($salt.$pass) == $hash);
    }
    return $ok;
}
```
Abusing the *Remember me* feature

Quick reminder about PHP loose comparisons...

```
"0e12345" == 0  # TRUE
"0e12345" == "0e54321"  # TRUE
"1foobarbaz" == 1  # TRUE
"1e12345" == 1  # FALSE
...
```
Abusing the *Remember me* feature

Thus we can make the code compare

```
int.sha1(int.personal_token) == int
```

- We are likely able to find an int producing a suitable SHA-1 output within a few tries
Abusing the *Remember me* feature

- @bitcoinctf brought to our attention that it is also possible to do this...

```php
$salt = substr(true, 0, 8);
// returns 1
$ok = ($salt->sha1($salt->$pass) == $hash);
// 1->sha1(1->$pass) == true
```

- No more need to iterate over a few integers, a single request is enough
DEMO
Going deeper

- We are admin on the solution (or any other user)
  - But the goal is still to compromise the infrastructure
  - We need to find something else on the authenticated part
- Time to compromise the underlying server
- Old vulnerabilities are patched
Fusion Inventory

While gathering technical details about the target’s infrastructure using regular features ...

GET /plugins/fusioninventory/front/send_inventory.php?itemtype=PluginFusioninventoryInventoryComputerComputer&function=sendXML&items_id=machine.xml&filename=toto HTTP/1.1

Back to the good old blackbox reflexes, a wild LFI appears

&items_id=../../../../../../../../../../../../../../../../../../etc/passwd
Fusion Inventory

- It works and this is pretty cool but we found nothing valuable on the server, let’s take a look at the code of the plugin

```php
$itemtype = $_GET['itemtype'];
$function = $_GET['function'];
$items_id = $_GET['items_id'];
header('Cache-control: private, must-revalidate'); // IE BUG + SSL
header('Content-disposition: attachment; filename='.$_GET['filename']);
header('Content-type: text/plain');
call_user_func(['PluginFusioninventoryToolbox', $function], $items_id, $itemtype);
```

- Unexpected

- Does the PluginFusioninventoryToolbox class implement more interesting functions?
Fusion Inventory

- Yes it does!

```php
function executeAsFusioninventoryUser($function, array $args = []) {
    [...]
    // Execute function with impersonated SESSION
    $result = call_user_func_array($function, $args);
    [...]
    // Return function results
```

- Only 1 requirement
  - `$args` has to be an Array
Fusion Inventory

- Fair enough, PHP allows playing with parameters

<table>
<thead>
<tr>
<th></th>
<th>?foo=bar</th>
<th>?foo[]=bar</th>
<th>?foo['bar']=bla</th>
<th>?foo[]=bar&amp;foo[]=bla</th>
</tr>
</thead>
<tbody>
<tr>
<td>?foo</td>
<td>string(3)</td>
<td>array(1)</td>
<td>array(1)</td>
<td>array(2)</td>
</tr>
<tr>
<td></td>
<td>&quot;bar&quot;</td>
<td>{ [0]=&gt;</td>
<td>[&quot;&quot;bar&quot;&quot;]=&gt;</td>
<td>[0]=&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string(3)</td>
<td>string(3)</td>
<td>string(3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;bar&quot;</td>
<td>&quot;bla&quot;</td>
<td>&quot;bar&quot;</td>
</tr>
</tbody>
</table>

- `call_user_func_array` can be used in this situation

- CVE-2019-10477
Fusion Inventory

One last thing

```php
$itemtype = $_GET['itemtype'];
$function = $_GET['function'];
$items_id = $_GET['items_id'];
header('Cache-control: private, must-revalidate');   // IE BUG + SSL
header('Content-disposition: attachment; filename='.$_GET['filename']);
header('Content-type: text/plain');
call_user_func(['PluginFusioninventoryToolbox', $function],
               $items_id, $itemtype);
```

There's no mention of a session or cookie at any moment

- That's ok, you can remove it
- This code is reachable without authentication :-(
Disclosure
Disclosure

Timeline

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early February</td>
<td>Issues reported</td>
</tr>
<tr>
<td>Early March</td>
<td>Issues fixed publicly on GitHub</td>
</tr>
<tr>
<td>March 15th</td>
<td>Release of 9.4.1</td>
</tr>
<tr>
<td>April 11th</td>
<td>Release of 9.3 backports (9.3.4)</td>
</tr>
<tr>
<td>Late April</td>
<td>Advisories publication</td>
</tr>
<tr>
<td>Early July</td>
<td>Here we are</td>
</tr>
</tbody>
</table>

The disclosure process was smooth and efficient

Maintainers responded and shipped patches in a timely manner; thanks again!
Do people patch?

- Telemetry is not very reliable
  - Old/test instances aren’t removed after some time
  - All instances might not have access to the Internet
- 3 days after patches came out, 30 instances were up-to-date
- 3 months later (end of June)
  - 8046 have been upgraded
  - 26807 remain vulnerable
- Digitemis created GLPIScan to check your instances
  - https://github.com/Digitemis/GLPIScan/
Conclusion
Conclusion and next steps

- Useless in this case but we now hunt for GLPI in internal pentests
- Indirectly, companies contribute to OSS security by including such products in pentest scopes
- We need more
  - Collaborative tools to review code
  - “Smart” static scanners
  - QL
- GLPI and MDM agents are cool targets for Red Teams and they need more attention/security contribution
THANKS FOR YOUR ATTENTION!

TIME FOR QUESTIONS