Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Antradar Gyroscope 07.2018

Cure53, Dr.-Ing. M. Heiderich, MSc. N. Krein, BSc. D. Weißer

Index

- Introduction
- Scope
- Identified Vulnerabilities
  - ANT-01-004 Web: Reflected XSS via codegen feature (High)
  - ANT-01-005 Web: Stored XSS via report function leads to ACL Bypass (Critical)
  - ANT-01-006 Web: SQL Injection/XSS via dynamic template functions (Critical)
  - ANT-01-007 Web: Reflected XSS in showhelp Function (High)
  - ANT-01-009 Web: Missing certificate verification could lead to SQLi/RCE (High)
  - ANT-01-012 Web: Missing CSRF check on speaker pictures update (Low)
- Miscellaneous Issues
  - ANT-01-001 Web: SQL Queries are incorrectly escaped (Medium)
  - ANT-01-002 Web: Passwords insecurely checked and stored (Low)
  - ANT-01-003 Web: hash_equals implemented incorrectly (Low)
  - ANT-01-008 Backend: MySQL root user employed exclusively (Medium)
  - ANT-01-010 Web: Recurrent use of weakly-typed comparisons (Medium)
  - ANT-01-011 Web: Potential SQL Injection via userid cookie (Medium)
  - ANT-01-013 Web: Hardcoded secrets and passwords (Low)
  - ANT-01-014 Web: IP spoofing via arbitrary proxy headers (Low)
  - ANT-01-015 Web: Current CSP settings does not prevent XSS (Medium)
- Conclusions

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

“The Gyroscope Framework is Antradar's flagship platform on which our web-based management systems are built. It uses a specific and yet flexible user interaction model to reveal and explore the inner-relations of a database.”

From http://www.antradar.com/docs-gyroscope-doc

This report documents the results of a penetration test and source code audit (later referred to as the “assessment”) against the Antradar Gyroscope PHP web application framework. The assessment was carried out by Cure53 in July 2018 and yielded fifteen security-relevant discoveries, including two ranked with the highest, *“Critical”*-level of severity.

In terms of resources and methodology, three Cure53 testers were tasked with executing this assessment and provided with a budget of five days. The testers received access to reference implementations, source code and technical documentation to be able to maximize coverage. This effectively means that the project relied on a so-called white-box methodology. Thanks to the open premise of a white-box approach, the Cure53 team and the framework developer remained in frequent contact throughout the assessment. Email exchanges were used as the primary means of communication and Cure53 sent regular updates about the test status and progress to the in-house team. Foreshadowing some of the conclusions, it should be clearly stated that the assessment revealed a plethora of security issues. Coming to a total of fifteen issues, the discoveries were categorized as six vulnerabilities and nine general security weaknesses. As important as the high overall number of findings is the fact that the issues received rather high severities due to their potential security implications. Namely, as already mentioned above, two of the vulnerabilities were rated as *“Critical”.* Three more problems were deemed to carry “*High”*-level risks. All issues were live-reported during the assessment so that the framework maintainer could begin to implement first fixes while the Cure53’s tests were still ongoing.

In the following sections, the report will first describe the scope and the test setup that Cure53 was provided with for the test. On this note, it must be added that the preparation phase was done impressively well by the in-house team and the setup was excellent overall. The subsequent sections of the report are dedicated to documenting all issues in detail, with mitigation offered as applicable. Finally, the report concludes with broader notes on the results of this assessment of the Antradar Gyroscope project. In other words, the last section is dedicated to discussing a set of observations and impressions that Cure53 testers have gained with reference to the general security posture of the Antradar Gyroscope PHP web application framework and its various components featured in this July 2018 assessment’s scope.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Key scope:**
  - Antradar Gyroscope web application framework written in PHP.
  - Antradar Gyroscope reference implementations for Cure53 to work with.
- **Server credentials & Sources**
  - Credentials were shared with Cure53
  - Sources were shared with Cure53
- **Plain reference application:**
  - https://vuln.antradar.com/plain/
  - https://vuln.antradar.com/plain/iphone.php
- **Realistic reference application:**
  - https://vuln.antradar.com/gsadmin/
  - https://vuln.antradar.com/gsadmin/iphone.php

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *ANT-01-001*) for the purpose of facilitating any future follow-up correspondence.

ANT-01-004 Web: Reflected XSS via *codegen* feature *(High)*

Gyroscope offers a feature called *codegen* which is not being linked in the backend directly. In this context, it is possible to use a form to dynamically generate code by changing certain settings through a form interface. It does not look like the generated code is actually stored back into the file system but it is rather echoed back to the user. The interesting aspect is that *codegen* can be used by low-privileged users as well if they directly call it via the service router.

Affected URL:
https://vuln.antradar.com/gsadmin/myservices.php?cmd=codegen_makeform&seed=album

Although the form is not directly functional because the required JavaScript files are missing, it can still be used to manually craft the request to *codegen_makecode*. The problem now lies within the fact that all of the submitted values of the form are directly reflected once *codegen* generates the code, thus causing a simple reflected Cross-Site

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scripting (XSS) vulnerability. To illustrate this, a Proof-of-Concept (PoC) consisting of an HTML file was created.

PoC.html:

```
<html>
  <body>
    <form action="https://vuln.antradar.com/gsadmin/myservices.php?
cmd=codegen_makecode&seed=album" method="POST">
      <input type="hidden" name="mastertable" value="albums" />
      <input type="hidden" name="primarykey" value="albumid" />
      <input type="hidden" name="primaryfield" value="album" />
      <input type="hidden" name="primaryrecords" value="albums" />
      <input type="hidden" name="c&#95;primaryrecords" value="Album" />
      <input type="hidden" name="primarydispfield" value="albumname" />
      <input type="hidden" name="record" value="pic" />
      <input type="hidden" name="records" value="pics" />
      <input type="hidden" name="lookuptable" value="pics" />
      <input type="hidden" name="lookupkey" value="picid" />
      <input type="hidden" name="c&#95;record" value="Picture" />
      <input type="hidden" name="c&#95;records" value="Pictures" />
      <input type="hidden" name="uploaddir"
value="&#46;&#46;&#47;images&#47;picss&#47;" />
      <input type="hidden" name="sizes"
value="thumb&#124;300&#124;300&#10;banner&#124;800&#124;600" />
      <input type="hidden" name="fileext"
value="&#46;pngxyz&apos;&lt;&#47;textarea&gt;&lt;script&gt;alert&#40;document.do
main&#41;&lt;&#47;script&gt;" />
      <input type="hidden" name="tinypngapi" value="" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

As soon as a logged-in victim browses to the attacker's website that hosts this file, *codegen* is automatically triggered and XSS occurs. From this point, all sorts of further XSS-based attacks against the victim can be performed. For example, it would be possible to craft a new login page where the victim would enter their credentials and unwittingly send them to the attacker's webserver.

It is recommended to use output validation mechanisms to reflect the supplied values in the *codegen* feature. Depending on the context, one should either use *htmlspecialchars* or *htmlspecialchars* with *ENT_QUOTES* as additional parameters. For normal output escaping outside of HTML tags, *htmlspecialchars* is sufficient to encode all malicious

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

tags. For situations where values are reflected inside the HTML tags, it is important to use *ENT_QUOTES* to guarantee that injecting event handlers into the context cannot be accomplished.

Exploitability note: *This issue is exploitable on multi-tenant setups and is vulnerable on each installation where browsers like Firefox or IE are used. Chrome partially mitigates this issue with its XSS filter.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-005 Web: Stored XSS via *report* function leads to ACL Bypass (*Critical*)

For unspecified reasons it was decided that all teams that make use of the Gyroscope platform (in this case Cure53 and the Smurfs) share the same *report* settings that can be configured inside the web-interface. Since all *reports* are allowed to have a “custom function” that can contain malicious JavaScript code, it is possible that this feature can be abused to run arbitrary JavaScript code in the context of an entirely different team / security firm. To demonstrate, the Smurfs can issue chosen instructions to exploit Cure53 and this is discussed next.

Steps to reproduce:

- A Smurf logs in and browses to the *report* setting.
- The Smurf enters and saves the following JavaScript code into the *Activity Log* of the custom function setting: *javascript:alert(1);*
- As soon as Cure53 logs into Gyroscope and examines the logs for their team by clicking on *Activity Log,* the XSS flaw is triggered.

Instead of the displayed *alert(1),* the Smurfs actually gain the option to automatically instruct Cure53 to perform all kinds of malicious actions in the context of their currently logged-in user. The vulnerable elements were spotted in the section of the source code provided next.

Affected File:
gyroscope/site/plain/icl/listreports.inc.php

Affected Code:

```
<div class="listitem">
      <a onclick="<?echo $reportfunc;?>reloadtab('rpt<?echo $reportkey;?>','<?
echo $dbreportname;?>','rpt<?echo $reportkey;?>',(self.rptreload_<?echo
$reportkey;?>?rptreload_<?echo $reportkey;?>:null));addtab('rpt<?echo
$reportkey;?>','<?echo $dbreportname;?>','rpt<?echo $reportkey;?>',
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
(self.rptinit_<?echo $reportkey;?>?rptinit_<?echo $reportkey;?>:null));"><?echo
htmlspecialchars($reportname);?></a>
</div>
```

Instead of letting the user choose an arbitrary *report* function, it should only be possible to select from a whitelisted set of supported custom functions. However, if this functionality is a business requirement, it should at least be made sure that different teams have their own *report* settings clearly separated. This could be implemented like other protections of editable resource, e.g. by either performing a *gsguard()* check or by manually verifying the *gsid* of the currently authenticated user.

Exploitability note: *This issue is exploitable on multi-tenant setups and is vulnerable on each installation of Gyroscope where users are able to edit report types.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-006 Web: SQL Injection/XSS via dynamic *template* functions (*Critical*)

The *templates* for events and the like make it possible to dynamically render variables inside the generated PDF brochures. According to the *readme,* events can also be rendered as HTML files. Nevertheless, this feature does not appear to be present in the current state of the application. For example, the author can include statements like *{{speaker speakerid=13 Doug Goofy}}* to dynamically include speaker details inside the generated PDF. Conversely, inside the source code it was spotted that the parsing of some variable parameters fails to make sure that they are escaped for SQL queries. This can be seen in the code furnished next.

Affected File:
gyroscope/site/gsadmin/tmpl.php

Affected Code:

```
function tmpl_bloglist($params){
	$deptid=$params['id'];
	global $db;
	global $curlang;
	global $dict;

	include_once 'makeslug.php';

	$query="select * from blog where groupdeptid=$deptid and
published_$curlang=1 order by blogdate desc";
[...]
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
function tmpl_blog($params){
        $deptid=$params['id'];
        global $db;
        global $curlang;
        global $dict;

        include_once 'makeslug.php';

        $perpage=5;
        if (is_numeric($params['perpage'])) $perpage=$params['perpage'];

        $query="select blog.*,unix_timestamp(blogdate) as bdate from blog where
groupdeptid=$deptid and published_$curlang=1 ";
[...]
```

The user has the option to include a statement like *{{blog id=foo}}* to dynamically render blog details into the PDF. However, as depicted above, the *deptid* is concatenated into an SQL query without making sure it is actually numeric. This leads to an exploitable SQL Injection, depending on whether blog details are enabled inside Gyroscope. Thus, by submitting the value of *{{blog id=foo SQL INJECTION POC}},* the following PDF is generated and signifies a proof of the SQL Injection.

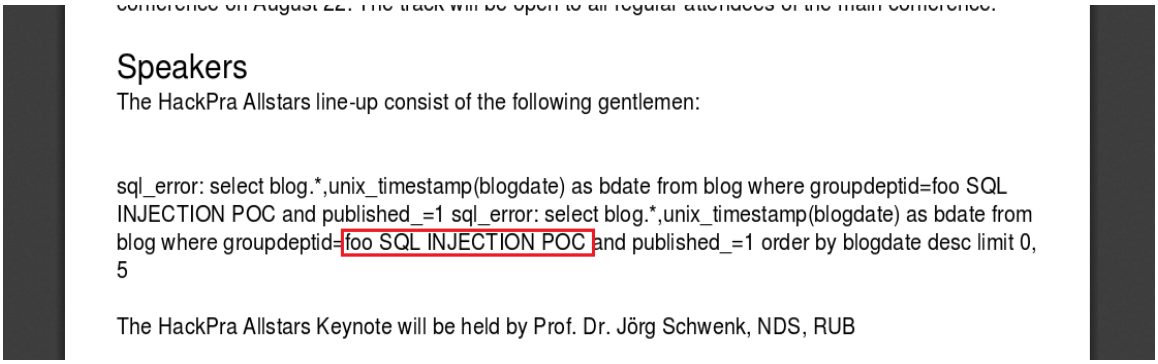

Fig.: SQL Injection via a rendered PDF

In an exploitable scenario, this can be abused to read all sorts of data from the database, for example encrypted passwords. Moreover, combined with the issue described in ANT-01-008, it allows a user to write and read local files by means of using *load_file* and *INTO OUTFILE* statements.

It is recommended to make sure that the supplied *params* inside the *tmpl_* functions are numeric and escaped. An even better solution would be to exclusively rely on prepared statements as described in ANT-01-001.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

If one is to trust the provided *readme,* it should also be possible to render the event in a HTML file. Since on multiple occasions *tmpl.php* fails to perform output-validation of certain *params* values, this would then result in exploitable XSS conditions as well. Another example would concern a vulnerable sink furnished next.

Affected Code:

```
function tmpl_youtube($params){
	$key=$params['key'];
	if ($key=='') return;

?>
<div class="videoanchor">
	<iframe class="videoframe"  src="http://www.youtube.com/embed/<?echo
$key;?>" frameborder="0" allowfullscreen></iframe>
</div>
<?
}
```

To mitigate the problem, it is important to make use of *htmlspecialchars* with *ENT_QUOTES* as parameters. This goes for all other *echo*-sinks where output-validation mechanisms are missing.

Exploitability note: *This issue is exploitable on special setups and is vulnerable on installations where the templating system and blog databases are used and users have the ability to write custom items - such as reports - into datasets.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-007 Web: Reflected XSS in showhelp Function *(High)*

The Gyroscope application provides a function that shows help pages based on the provided "*topic*" parameter. To achieve this, a PHP file is included and an error is displayed in case the file does not exist. The error message reflects the contents of the parameter in an unsanitized form back to the user, thus leading to XSS. A PoC is given in the following URL.

PoC:
https://vuln.antradar.com/plain/myservices.tmpl.php?cmd=showhelp&topic=%3Csvg/onload=alert(1)%3E

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In order to trigger the vulnerability, the targeted user needs to be authenticated. An attacker could use this issue in order to issue requests on the victim's behalf and to steal sensitive data.

It is recommended to encode the parameter before printing it back to the web page.

Exploitability note: *This issue is exploitable on setups where the* showhelp *service is enabled. Browsers such as Chrome partially mitigate this issue when XSS filters are enabled.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-009 Web: Missing certificate verification could lead to SQLi/RCE *(High)*

The Gyroscope application has a feature allowing to install additional modules from an external source. Such an extension consists of a JSON file and several additional PHP scripts. This feature requires a URL to be specified from where the extensions are loaded. Because the connection is handled in an insecure way, it is possible for a Man-in-the-Middle (MitM) attacker to inject arbitrary SQL commands into the server's response. Similarly, uploading arbitrary PHP code could also be achieved. The problem occurs in the code snippet supplied next.

Affected File:
site/plain/icl/installmods.inc.php

Affected Code:

```
$url=MOD_SERVER.'?lang='.$lang.'&cmd=installmods&modids='.$modids.'&version='.
$version.'&devmode='.
$devmode.'&project='.urlencode(GYROSCOPE_PROJECT).'&vendor='.urlencode(VENDOR_NA
ME).'&vendorversion='.VENDOR_VERSION;

$curl=curl_init($url);
curl_setopt($curl,CURLOPT_RETURNTRANSFER,1);
curl_setopt($curl,CURLOPT_SSL_VERIFYPEER,0);
curl_setopt($curl,CURLOPT_SSL_VERIFYHOST,0);
```

When a module is installed, the required data is fetched from the configured *MOD_SERVER* using cURL. Crucially, remotely fetched executable code should always be relayed in a secure way (e.g. via SSL) in order to prevent MitM attackers from replacing the downloaded files with malware. Here the verification of SSL certificates is explicitly disabled, making it easy for a MitM to manipulate network traffic on the fly.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The same issue was found in several other, less critical contexts within the Gyroscope project (e.g. *site/plain/sendsms.php*). Sensitive data and executables should always be transmitted via a secure connection. Therefore, it is recommended to re-enable the certificate check.

Exploitability note: *According to the developers, this issue should not be exploitable in the current setups since the* update *mechanism in question has been deprecated.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-012 Web: Missing CSRF check on speaker pictures update *(Low)*

It was discovered that the handler responsible for updating speakers' pictures fails to employ a CSRF check. A malicious website could use this issue to force an authenticated user into sending update requests. Fortunately, the impact of this issue is only of a cosmetic nature, thus it is unlikely to be exploited.

Still, it is recommended to add the already implemented CSRF check to the upload handler for the speaker pictures item. A quick analysis of other uploaders leads Cure53 to believe that this is a more general issue that concerns multiple endpoints. For example, the *embedmediauploader()* function suffers from the same missing check.

Exploitability note: *This issue is exploitable on multi-tenant setups where image uploaders are used. Tenants can thus instruct other tenants to upload pictures to their resources.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

ANT-01-001 Web: SQL Queries are incorrectly escaped *(Medium)*

Throughout the entire Gyroscope application, SQL query parameters are always manually escaped. On most occasions this is done incorrectly and could actually lead to exploitable conditions. The code below serves as just one example.

Affected File:
gyroscope/site/gsadmin/ajx_2facheck.php

Affected Source:

```
$raw_login=$_POST['login'];
$login=str_replace("'",'',$raw_login);

$query="select * from ".TABLENAME_USERS." left join gss on
".TABLENAME_USERS.".gsid=gss.gsid where login='$login' and active=1 and
virtualuser=0";
```

The only mechanism to “prevent” SQL Injection in this context would be achieved by removing all single quotes (') from the *login* name. However, other values that can introduce an injection (such as backslashes, \) are deliberately left out. Therefore, if a user tries to login with “*username*\”, an SQL error is triggered. This is because the closing single quote gets escaped and the first single quote never gets closed. In this specific scenario, the flaw is not directly exploitable. Still, because Gyroscope requires a lot of manual code editing when deployed in a new scenario, it is not impossible for queries such as the one below to exist.

Example vulnerable query:

```
$query = "SELECT * FROM table WHERE a = '$userinput1' and b = '$userinput2'";
```

If the query’s parameters are escaped in the same manner as depicted above, an attacker can use the values included next to conduct an SQLi just by removing single quotes.

Used Parameters

```
$userinput1 = 'test\\';
$userinput2 = ' OR 1=1 SQLInjection here-- f'
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Resulting Query:

```
SELECT * FROM table WHERE a = 'test\' and b = ' OR 1=1 SQLInjection here-- f'
```

Therefore, the second user-input actually gets treated as SQL commands since the second single quote is escaped.

For the most part, user-input is globally escaped via the *GETVAL* and *GETSTR* functions. *GETVAL* correctly checks whether the input is numeric while *GETSTR* incorrectly uses *addslashes* to escape user-input. While in the current setup *addslashes* cannot be exploited, it is possible to bypass escaping with multibyte sequences when certain DB character sets are used. Even if the probability of an Asian character set is rather low, it should be made sure that a designated SQL escaping function, such as *mysqli_real_escape_string*, is used.

Even so, it is discouraged to manually escape every statement this way. Especially since it is easy to make mistakes when, for example, *GETSTR* is confused with a unquoted context where *GETVAL* should have been used. Therefore, it is highly recommended to exclusively use prepared statements to automatically and correctly bind and escape all parameter values. This is sufficiently documented inside the PHP manual, which should be reviewed for the context of relying on the MySQLi interface[1].

Exploitability note: *This is a general issue which becomes exploitable when Asian character sets are used for the databases or when simple input sanitizing is forgotten.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-002 Web: Passwords insecurely checked and stored *(Low)*

Opposing what is recommended as the industry standard, Gyroscope implements additional encryption via OpenSSL *AES-CBC* for user-passwords. This is visible in the following excerpts.

Affected File:
gyroscope/site/gsadmin/login.php

Affected Source:

```
$cfk=$_POST['cfk'];
if ($cfk!=$csrfkey&&$cfk!=$csrfkey2){
       $error_message=_tr('csrf_expire');
} else {
```

[1] http://php.net/manual/de/mysqli.quickstart.prepared-statements.php

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
	$password=md5($dbsalt.$_POST['password']);
	$raw_login=$_POST['gyroscope_login_'.$dkey];
	$login=str_replace("'",'',$raw_login);

	$query="select * from ".TABLENAME_USERS." left join gss on
".TABLENAME_USERS.".gsid=gss.gsid where login='$login' and active=1 and
virtualuser=0";
	$rs=sql_query($query,$db);

	$passok=0;

	if ($myrow=sql_fetch_array($rs)){
		$enc=$myrow['password'];
		$dec=decstr($enc,$_POST['password'].$dbsalt);
		if ($password==$dec) $passok=1;
	}
```

Affected File:
gyroscope/site/gsadmin/encdec.php

Affected Source:

```
function decstr($str,$key){
	$raw=base64_decode($str);

	$method='AES-256-CBC';
	$key=enckey($key);

	$ivlen=openssl_cipher_iv_length($method);
	$iv=substr($raw,0,$ivlen);
	$enc=substr($raw,$ivlen);

	$dec=openssl_decrypt($enc,$method,$key,0,$iv);


	return $dec;
}
```

Contrary to popular belief, this does not actually increase the security of the password storage, especially because a weak hash function is used prior to encrypting. With the help of the SQL Injections, it is possible to fetch the key from the source code anyway, especially considering that *md5* has long been proven insecure for creating password hashes. The fact that a weak comparison is used to verify password also allows type-juggling attacks and signifies a potential to bypass the authentication entirely (for

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

example by bruteforcing all values to contain only *0e* hashes[2]). Instead of increasing the complexity via unnecessary encryption, which opens doors for padding attacks via *AES-CBC* as well, it is simply recommended to use to a secure hashing function such as *bcrypt*. To securely create hashes, one should switch to PHP's *password_hash* function and use *password_verify* to securely compare them.

Exploitability note: *This is a general issue that is only exploitable once attackers obtain a copy of the Gyroscope database and decide to locally bruteforce user-passwords.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-003 Web: *hash_equals* implemented incorrectly *(Low)*

In order to backport Gyroscope for PHP versions below 5.6.0, where *hash_equals* has not been defined yet, this function is re-implemented in an insecure manner. The problem can be seen in the following code.

Affected File:
gyroscope/site/gsadmin/auth.php

Affected Source:
```
if (!is_callable('hash_equals')){
	function hash_equals($a,$b){return $a==$b;}
}
```

Since the function only uses a timing-unsafe weakly-typed comparison, it is neither considered secure, nor does it promise the same features that *hash_equals* offers. Especially since == is used instead of ===, this can actually lead to type-jugging scenarios where the comparison gets bypassed completely. It is recommended to backport *hash_equals* securely, for example by using the function proposed next.

Recommended backport:
```
if (!function_exists('hash_equals')) {
	function hash_equals($str1, $str2)
	{
		if (strlen($str1) !== strlen($str2)) {
			return false;
		} else {
			$res = $str1 ^ $str2;
			$ret = 0;
			for ($i = strlen($res) - 1; $i >= 0; $i--)
```

[2] https://www.whitehatsec.com/blog/magic-hashes/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
                        $ret |= ord($res[$i]);
                return !$ret;
        }
    }
}
```

Exploitability note: *This is a general issue that only concerns installations with PHP versions lower than 5.6.0. Moreover, an exploit is rather hard to achieve in this scenario.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-008 Backend: MySQL *root* user employed exclusively *(Medium)*

In the shared test setup, the Gyroscope's MySQL interface uses the *root* user to connect to the MySQL service. Although this might not actually reflect the state of the existing production setups, it is still important to highlight the problems with this approach.

Affected File:
gyroscope/site/gsadmin/connect.php

Affected Code:

```
include_once "sql.php";

if (defined('GSSERVICE')) {
    $db=sql_get_db('p:127.0.0.1','vuln','root','mnstudio','db');
} else {
    $db=sql_get_db('127.0.0.1','vuln','root','mnstudio','db');
}
```

The most obvious issue is that compromised source code can lead to the complete database takeover, spanning also other databases that might be hosted on the same service. Combined with the fact that the application contains SQL Injection vulnerabilities, attackers can abuse all privileged *root* connections to read files (via *SELECT load_file()).* This would mean that a malicious adversary can steal all configuration options and secret keys from the filesystem. What is more, it is also possible, depending on the access rights to the web-root, to write into the files and this would enable uploads of PHP shells into the *writable* upload directories with the help of *SELECT ... INTO OUTFILE*.

It is recommended to use a least-privilege model for the database users. Gyroscope should get its own user to connect to the database and only be allowed to use *SELECT*, *UPDATE, INSERT* and *DELETE* privileges. Depending on the configuration, more privileges should be granted after carefully studying whether they are actually necessary.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Exploitability note: *This is a general issue that is not directly exploitable but rather depends on the attackers who have already found a second vulnerability like file disclosure.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-010 Web: Recurrent use of weakly-typed comparisons *(Medium)*

This ticket describes a general coding weakness that was noticed during the audit of the Gyroscope's source code. To be more precise, Gyroscope exclusively makes use of weakly-typed comparisons, namely via ==, to compare values to one another. The problem here lies within the fact that PHP is a loosely-typed language which interprets the values by casting them to different data-types in reference to the context. The PHP String Comparison Vulnerabilities[3] resource shows that quite clearly.

This issue is especially problematic since it is used throughout the application and in security-sensitive contexts, e.g. for password or hash verifications (see ANT-01-002 or ANT-01-003). Generally speaking, there is no single use of === which could be seen as a secure variant of PHP comparisons because they firstly decide whether the passed values are of the same data-type.

Simply to protect Gyroscope from issues like type-juggling and similar, Cure53 recommends to replace all occurrences of == with === across the entire Gyroscope's custom code. In practice, this should not cause any trouble with the flow of the application and it happens quite rarely that this causes bugs. If Gyroscope actually runs into trouble with this solution, then the altered parts need to be slowly rolled back while making sure that any leftover == do not remain in security-relevant contexts.

Exploitability note: *This is a general issue that is not directly exploitable but rather exhibits a weak coding practice potentially leading to exploitable issues in the future.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

[3] https://hydrasky.com/network-security/php-string-comparison-vulnerabilities/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

ANT-01-011 Web: Potential SQL Injection via *userid* cookie *(Medium)*

It was noticed that the *showaccount* function used to display account details fetches all user-information via *userinfo().* By doing so, it also includes information from the passed cookies. Since the later returned *userid* is not actually escaped, it needs to be manually verified as numeric depending on the use-case. This signifies a pitfall because it is easy to forget to cast the *userid* to a number. One such occasion was found in *showaccount.inc.php* and can be found below.

Affected File:
gyroscope/site/gsadmin/icl/showaccount.inc.php

Affected Code:
```
function showaccount(){
	global $smskey;

	$user=userinfo();

	global $db;

	$userid=$user['userid']+0;
	$query="select * from users where userid=$userid";
	$rs=sql_query($query,$db);
	$myrow=sql_fetch_assoc($rs);
	$needkeyfile=$myrow['needkeyfile'];
	$usesms=$myrow['usesms'];
	$smscell=$myrow['smscell'];

	if ($smskey=='') $usesms=0;
[...]
	<?showkeyfilepad('mykeyfile',$user['userid']);?>
```

Affected File:
gyroscope/site/gsadmin/icl/showkeyfilepad.inc.php

Affected Code:
```
function showkeyfilepad($container,$userid){
	global $db;
	global $codepage;

	$query="select * from users where userid=$userid";
	$rs=sql_query($query,$db);
	$myrow=sql_fetch_assoc($rs);
	$haskeyfile=0;
	if ($myrow['keyfilehash']!='') $haskeyfile=1;
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

As depicted above, the *$userid* variable is manually cast to an integer via *$userid=$user['userid']+0* but the later used *showkeyfilepad()* function still relies on the uncasted *$user['userid']* variable fetched from the cookies. Therefore, it is permitted for the SQL sequences to be included, thus causing an SQL Injection.

This issue is partially mitigated by the fact that a correctly hashed *auth* cookie is required in order to craft a malicious *userid* cookie. Since it is based on a hardcoded *salt* value, an attacker requires a file disclosure vulnerability to actually exploit this issue. Nevertheless, this scenario can be treated as an input-validation problem and should be fixed accordingly. For example, one approach would be to pass *$userid* to *showkeyfilepad()*.

Exploitability note: *This is a general issue that is not directly exploitable but rather depends on attackers having already found a second vulnerability such as file disclosure.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-013 Web: Hardcoded secrets and passwords *(Low)*

Following up on ANT-01-008, it needs to be reiterated that Gyroscope hardcodes a lot of secrets inside the framework's source code. This applies to *dbsalts*, the *gsreqkey* and also the MySQL credentials, among others. Quite evidently, this not a best practice approach since an attacker with file disclosure capabilities would effectively be able to leak all secrets from the filesystem.

The recommended approach in this realm is to store all secrets inside environment variables and use PHP's *getenv()* to fetch them. One possible deployment is to use Nginx' *env* directive to export the variables inside the website's *.conf-file*. The same can be accomplished via Apache's *envvars*. However, it is important that those *config* files are not readable by the current web user such as *www-data* or *nobody*.

Exploitability note: *This is a general issue that is not directly exploitable but rather depends on attackers having already found a second vulnerability such as file disclosure.*

Fix note: *This problem has not yet been addressed but might be in a future release. It was agreed that this issue is of acceptable risk.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

ANT-01-014 Web: IP spoofing via arbitrary proxy headers *(Low)*

The globally included *lb.php* defines a few routines to extract proxy headers such as *HTTP_X_FORWARDED_FOR* or *HTTP_X_REAL_IP*. These can be used to reset the value of *$_SERVER['REMOTE_ADDR']* without paying attention to the actual deployment. The sequence creates an IP spoofing issue where attackers can simply set an arbitrary value via *HTTP_X_FORWARDED_FOR*, thus fooling the application into believing the attacker connects via an entirely different IP than the one s/he actually makes use of.

Since *REMOTE_ADDR* is also used inside of the sensitive calculations to generate CSRF keys and *auth* cookies, Cure53 feels that the application should make sure that *REMOTE_ADDR* **always** contains the actual IP address of the user. Headers such as *HTTP_X_FORWARDED_FOR* should only be used if Gyroscope is installed behind a reverse-proxy.

Exploitability note: *This is a general issue that slightly weakens the generation of secrets where the user's IP address is used for the calculation and is therefore not directly exploitable.*

Fix note: *This issue has been addressed by the framework maintainers and the fix was successfully verified by Cure53 in August 2018.*

ANT-01-015 Web: Current CSP settings does not prevent XSS *(Medium)*

An analysis of the currently deployed security headers made Cure53 aware of a useless CSP setting. The CSP header return by the current state of the application can be found next.

Returned Headers:

```
[...]
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Security-Policy: child-src 'self'
```

Since the *child-src* directive only determines the resource from which nested browsing contexts are loaded, it does not tell the browser how to handle script directives. Thus, it also does not prevent XSS. After studying the source code that handles this header, it became apparent that the original intention was to set a *default-src* directive as well.

Affected File:
gyroscope/site/plain/xss.php

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
function xsscheck(){
	global $_SERVER;
	header('X-Frame-Options: SAMEORIGIN');
	header('X-XSS-Protection: 1; mode=block');
	header('X-Content-Type-Options: nosniff');
	header("Content-Security-Policy: default-src 'self'");
	header("Content-Security-Policy: child-src 'self'");
```

However, since two separate *header()* statements are used, the second one overwrites the first one. As a result, the first one is never sent back to the browser. The correct approach would be to nest both statements into one, as demonstrated next.

Proposed Patch:

```
header("Content-Security-Policy: default-src 'self'; child-src 'self'");
```

By employing the proposal above, the browser correctly receives both statements and prevents loading of scripts that reside outside of the *self*-context. In effect, it should also prevent issues such as ANT-01-007.

Exploitability note: *This issue describes a general weakness which renders the current CSP setting ineffective. On multi-tenant setups where XSS vulnerabilities are already present, this issue facilitates exploitation.*

Fix note: *This problem has not yet been addressed but might be in a future release. It was agreed that this issue is of acceptable risk.*

Conclusions

After assessing the Antradar Gyroscope PHP web application framework over the course of five days in July 2018, Cure53 can conclude that the project requires a lot of work from a security standpoint. At this moment in time, major engineering work is needed to bring the project forward and allow Cure53 to consider it an actually secure framework. This can be inferred from several indicators, specifically the high number of findings, their rather concerning severities, and a significant number of general flaws in the coding practices.

To reiterate, the key challenges that the Gyroscope is facing are not insurmountable but should be seen as multifaceted. First of all, the out-of-date coding practices must be eradicated and replaced by modern approaches. Secondly and similarly, solutions must be developed to address the lacking or missing current industry standards on the

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Gyroscope's scope. Thirdly, the general attitude of having to rely on individual fixes to mitigate single issues must be exchanged for emphasizing best-practices that offer holistic and comprehensive mechanisms capable of preventing entire classes of attacks and problems.

There should be no doubt that fifteen security-relevant discoveries, with as many as five items being ascribed with "*Critical*" and "*High*" severities, cannot signify a passable outcome for this assessment. In providing some details, the out-of-date coding practices can be derived from relying on the now long-avoided hashing mechanisms (see ANT-01-002) or the fact that each application's secret is simply hardcoded inside the source code, as described in ANT-01-013. In the same vein, long-known issues with PHP language's constructs - such as its affinity to creating type-juggling problems - are not currently considered by Gyroscope. As a result, coding practices that may signal major problems in the future are propagated rather than dealt with.

In addition, Gyroscope relies on the again rather "deprecated" SQL constructions where user-input is directly mixed with hardcoded queries. Despite Cure53's recommendation to exclusively make use of prepared statements instead of manually escaping each variable, the developers claim that the current approach saves both development and debugging time. However, Cure53 cannot agree with this logic since shortages in development time cannot be seen as an argument for accepting an insecure foundation. This especially holds since one can easily create a class that handles multiple SQL derivatives and falls back to the correct prepared statement method. A subsequent, related point is that Gyroscope is functional but fails to recognize and include modern standards. This is exemplified by the absence of object-oriented code that would prevent issues with multiple, different database classes. What is more, it could effectively make Gyroscope more "extendable", thus potentially saving development time in the future.

Beyond the above, Gyroscope is riddled with multiple bugs and each requires manual fixing instead of having a general wrapper that takes care of most issues. For example, each XSS issue can be automatically solved by including sanitizers such as *DOMPurify*[4]. The fact that each and every vulnerability sink needs to be manually inspected on an individual basis creates multiple pitfalls and requires a nearly infeasible level of dedication and constant focus from the developers who would need avoid every mistake. In the current state, nearly every state changing function requires an easy-to-forget CSRF check. Similarly, every reflection requires separate output-validation. Finally, each SQL query input needs to be manually sanitized. Since approaches such as automatic unit, error or regression testing are missing, Gyroscope will have to rely on manual auditing and fixing. These aspects cannot stand, especially if a long-term development and expansion are envisioned.

[4] https://github.com/cure53/DOMPurify

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In terms of important positive feedback, Cure53 has to mention that the developers quickly reacted to each issue and proposed a patch in parallel to the testing process. It is very noticeable that the in-house team lead is open to a major change in perspective and initialized the practice of adopting new core practices. Additionally, all communication between Cure53 and the developers was fluent and efficient, with obscure aspects addressed quickly and clearly. It is noticeable that Gyroscope aims at being a secure framework and has started to put a lot of effort into achieving this goal. In that sense, this Cure53 assessment was a step in the right direction.

In summary, in the current state Gyroscope will have a hard time in protecting its users when single bugs have to be eradicated one by one and the project is plagued by an array of insecure realms. In that sense, the results of this Cure53 July 2018 assessment are negative. It must be first and foremost understood that it is close to impossible but mandatory to keep developing an application and increase its complexity while reaching security standards of the same level. Only time will tell whether the Gyroscope team actually invests into meeting all required standards and follows through on the recommendations made by the Cure53 team. Despite the currently lacking level of maturity and substandard state of the project's security posture, Cure53 believes that optimization and acceptable standards of protections and safety can in time be achieved by the Antradar Gyroscope PHP web application framework with due attention and considerable efforts.

Cure53 would like to thank Schien Dong of Antradar for his excellent project coordination, support and assistance, both before and during this assignment.