Authorization Bypass: A Cautionary Tale CVE (2020-11679, 2020-11680, 2020-11681)

Author: Aaron Bishop

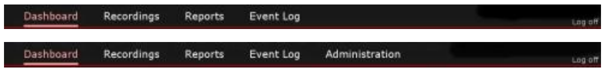
monitors illuminating the Guy Fawkes mask that conceals their face. In reality, however, most applications have the vast majority of their functionality (and potential attack surface) behind the login page where the "known" users roam. What if the feared "unknown attacker" turns out to be a trusted, authenticated, "known" user? Maybe the "unknown attacker" registered for the application or compromised a valid account and now has access. Is the sensitive functionality properly restricted from these "known" users as well?

For example, can low-privileged users browse directly to functionality that should be reserved for higher-privileged roles? Hiding elements with `CSS` or `HTML` comments or not including links in the navbar may be convenient for normal users but it does not provide security or proper access control which opens the door for abuse and attacks. Similarly, restricting `GET` requests can prevent users from viewing restricted information, but what about other HTTP methods? Are low-privileged users able to `POST` a form or `PUT` JSON thus allowing them to create objects or modify the state of the application?

So our story begins... during a penetration test, a valid, low-privileged user was able to perform actions that should have been restricted - including promoting their own account to admin.

Authorization Bypass in a Penetration Test

I had access to two accounts: a low-privileged *Reviewer* and a high-privileged *Administrator*. The *Reviewer* had restricted access and could not browse to the functionality in the **Administration** tab that was visible from the *Administrator* role:

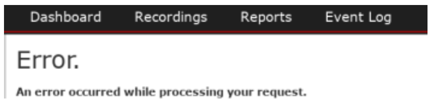


Browsing directly to the page is the simplest way to check if access controls exist but isn't the only option. For instance, I wondered how the application would respond if I called *Administrator* functionality that used another method, like the following *POST* request to modify my user roles:

```
POST /Administration/Users/Edit/:ID HTTP/1.1
Host: $RHOST
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: $REVIEWER_COOKIES
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 349

UserId=:ID&Email=bypass%40test.com&FirstName=bypass&LastName=bypass&LDAPUser=false
&Roles%5B0%5D.RoleId=1&Roles%5B0%5D.IsSelected=true&Roles%5B0%5D.IsSelected=false
&Roles%5B1%5D.RoleId=3&Roles%5B1%5D.IsSelected=true&Roles%5B1%5D.IsSelected=false
&Roles%5B2%5D.RoleId=5&Roles%5B2%5D.IsSelected=true&Roles%5B2%5D.IsSelected=false
&Locked=false
```

Well ... the application responded with the following error:



Dashboard Recordings Reports Event Log

Error.

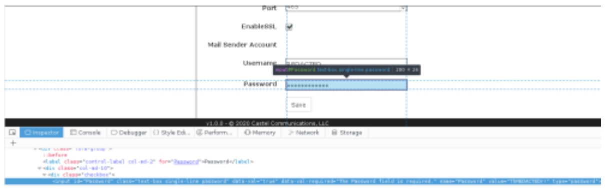
An error occurred while processing your request.

But the error was not a denial of access or due to insufficient privileges; it was a general error that had not appeared in the application before.

In the past I've come across applications that set permissions and access levels when a user authenticates, so I logged out, then logged back in and the Administrative functionality was now accessible.

As my low-privileged user I could directly call functionality provided by **/Administration/Users** to add the *Administrator* role to my account or I could create a new user with the *Administrator* role. (This issue was reported, resolved, and assigned CVE-2020-11679.)

I then used my escalated privileges to view pages that were previously restricted, pages such as **/Administration/SMTP**. I reviewed the source for **/Administration/SMTP** and noticed the credentials for the server were in the page:



```
::before
<label class="control-label col-md-2" for="Password">Password</label>
<div class="col-md-10">
<div class="checkbox">
```

Returning authentication credentials in the source of the application is a dangerous practice; if the page is cached or a malicious user is able to bypass restrictions (like I did), the credentials would be disclosed and the account compromised. In this instance, the SMTP account could be abused to spoof messages, spam others, etc.. (This issue was reported, resolved, and assigned CVE-2020-11681)

I removed the *Administrator* role from my account so it was, once again, low privilege and evaluated additional calls. I found that I could directly call the functionality of the following **/Administration** pages:

```
POST /Administration/Alerts/Create          POST /Administration/Alerts/Delete
POST /Administration/Archiving/             POST /Administration/Archiving/Create
POST /Administration/Archiving/Delete       POST /Administration/FileStores/Create
POST /Administration/FileStores/Delete      POST /Administration/LDAP
POST /Administration/Roles/Create           POST /Administration/Roles/Edit/:RoleId
POST /Administration/Roles/Delete           POST /Administration/SMTP
POST /Administration/Users/Create           POST /Administration/Users/Edit/:UserId

POST /Administration/Users/ResetPassword
POST /Administration/FileStores/Edit/:FileStoreId
POST /Administration/Archiving/Edit/:ArchiveRuleId
POST /Administration/Alerts/Edit/:EventAlertId
```

I used the calls to modify the information in other accounts (like the account email), then forced password resets, and finally compromised the accounts.

Low-privileged users could directly call functionality intended only for users with the *Administrator* role; the ability to bypass authorization was reported, resolved, and assigned CVE-2020-11680.

Conclusion

My user was "trusted", but my actions should not have been. I was able to directly call functionality that should have been restricted; I promoted my own role to *Administrator* and retrieved credentials for another host. If an "unknown attacker" in a Guy Fawkes mask compromised a user account they could have, from there, moved on to admin and the extent of damage caused could have been catastrophic.

Review OWASP Proactive Controls, Be proactive. Protect your application, even if it is from your own "known" and "trusted" users.

PIN Security Assessment
PCI Policies
PCI Training
HIPAA Compliance
HIPAA Policies
HIPAA Training
HIPAA BA Program
HITRUST

Consulting
Reseller
Forensic/Incident Response
Data and Network Security
Data Security Academy
Webpage Integrity Monitoring (WIM)
Shopping Cart Inspect
Shopping Cart Monitor

Abuse

CONTACT

Contact Us
Contact Sales
Contact Support
Report a Vulnerability