Bug Bounty Research Project Report for Rec Room Inc. Services

By: Derrick Boyer and Hunter Weaver

Dwight Hobbes

CSCI 415

13 April 2025

Executive Summary

This report outlines a targeted security assessment of Rec Room Inc.'s web applications. While the original scope included a broad range of platforms—console, VR, mobile, and desktop—the focus was narrowed to three specific web targets: rec.net, recroom.com, and devportal.rec.net. These were selected for their exposure to user input and potential for common web-based vulnerabilities.

Testing focused on four primary attack types: Cross Site Scripting (XSS), SQL Injection (SQLi), Path Traversal, and Insecure Direct Object Reference (IDOR). Manual input testing and request interception via Burp Suite were used across all targets.

For both XSS and SQLi, no exploitable vulnerabilities were found. Inputs across all three platforms were tested with a variety of payloads, both through the frontend and via intercepted requests. In the case of XSS, rec.net and devportal.rec.net effectively handled all attempts, while recroom.com reflected input in a way that appeared potentially vulnerable but ultimately filtered or rejected all payloads. Similarly, SQL injection attempts—including union-based and login bypass payloads—failed to execute, suggesting the likely use of backend filtering or prepared statements to handle queries safely.

Path traversal testing revealed that most endpoints correctly filtered or sanitized malicious inputs. While devportal.rec.net did respond to encoded traversal attempts, no sensitive files were exposed, and access controls (403 errors) were properly enforced. For IDOR, the forum pages on forum.rec.net allowed navigation between public categories using predictable URL patterns, but access to private or sensitive content was correctly restricted. These results suggest appropriate access validation is in place.

Overall, the applications appear to be securely built and resistant to common attack types. While no active vulnerabilities were identified, it is recommended that recroom.com's search input be reviewed and hardened to better match the stricter handling seen on the other platforms. Full details of the testing and findings follow in the sections below.

Scope

The scope for this project was large, but reduced to a few targets. The bounty offer specified a scope that included their Xbox, PS4, PS5, Steam, Web Browser, and virtual reality applications as well as websites for their community, studio, and developer portal. The focus for this project was on the three web applications they included: Rec.net - as well as any subdirectories included from the website (*.rec.net/*), RecRoom.com for their studio application, and DevPortal.Rec.net for their developer portal where developers who have signed up have access to their APIs. These three targets were attacked with four separate methods, which include cross site scripting, SQL injection, path traversal, and IDOR. More in depth analysis of these attacks can be found under "Methodologies" later in the report.

Target Summary

Rec Room is a cross-platform, multiplayer social virtual reality and gaming platform. Users use it to create, share, and play games in an interactive social environment. It was developed by Rec Room Inc., and is available on multiple virtual reality headsets, Xbox, Playstation 4 and 5 consoles, as well as PC, iOS, and Android devices. Their web services were the main point of attack for this report, as discussed previously in the "Scope" section. Their web services are mostly written using C# for scripting and Azure to host their services, such as AWS for their backend databases and such. The application seems to be built in a way to prevent or deter common or basic attack types. The findings from this report are discussed, in detail, in the following "Methodologies" section.

Methodologies

Cross Site Scripting (XSS):

Cross Site Scripting is a web-based attack that allows hackers to inject malicious scripts into websites. Doing so can grant the attacker access to sensitive information such as passwords, usernames, credit card info, and any other account info stored by the website, as well as allowing them to potentially compromise a user's interactions with the website. This typically involves finding an input of some kind, testing to see if it filters out HTML or some other markup or scripting language that the payload is written in, and then injecting that payload directly, or by intercepting the request using a tool like Burp Suite. There are three categories of cross site scripting. There is reflected cross site scripting which is when an attacker's malicious script is injected into a request sent to the server or backend of the website. The result of this attack is then reflected in the response. This is typically used for gaining access to confidential info. There is also stored cross site scripting which is when a payload is injected and stored on the server. For example, data stored in the backend, like chat messages in a comment section on a webpage. The result of this is served back to users who do not know the payload is active on the server. This is typically used to send malicious scripts to users, whose browsers have no way of identifying the script as harmful, and execute it. The last kind is DOM-based cross site scripting which involves running scripts on the client's side of the website for any particular reason.

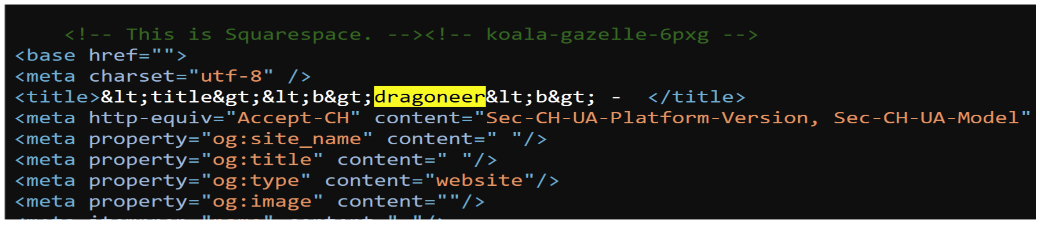
When searching for cross site scripting vulnerabilities there were three main targets of focus. The targets included Rec Room's main website for users, rec.net, as well as any pages navigated to within that website (*.rec.net/*), their main website for information, recroom.com, and their developer portal, devportal.rec.net, which is used by developers who need API access to Rec Room's services.

To start, rec.net was the first target. The initial search began by testing any and all inputs on the website. This included query boxes to search for rooms for players, as well as the same for events, creators, and inventions. The initial payload used was the unique word “dragoneer.” By using a word as unique as such, it was assumed it would not appear anyway in the website’s source code unless it was reflecting the query terms within. This assumption was correct in all cases tested. However, when reviewing the source code for the website, the word was never reflected. This meaning, it never showed up in the source code. That meant that the queries were either being filtered for markup/scripts, or not being used in the code for the client, other than to send the request. This appeared to be the case for each subdirectory travelled to. Each input seemed to be immune to injecting some sort of script. Burp Suite was also used to intercept and inject a simple payload which did not run. This means that there appeared to be no cross site scripting vulnerabilities on the rec.net website. The third option, the developer portal, appeared to be similar and no vulnerabilities could be found.

The second website appeared to be somewhat more interesting. When searching in the news section on recroom.com, the query input did show up in the source code. Upon querying for “dragoneer” the word showed up in two places: In the HTML and in the Javascript included in the GET request for the website. Because of this, the team moved on to more advanced payloads. The first attempt was to use the HTML code:

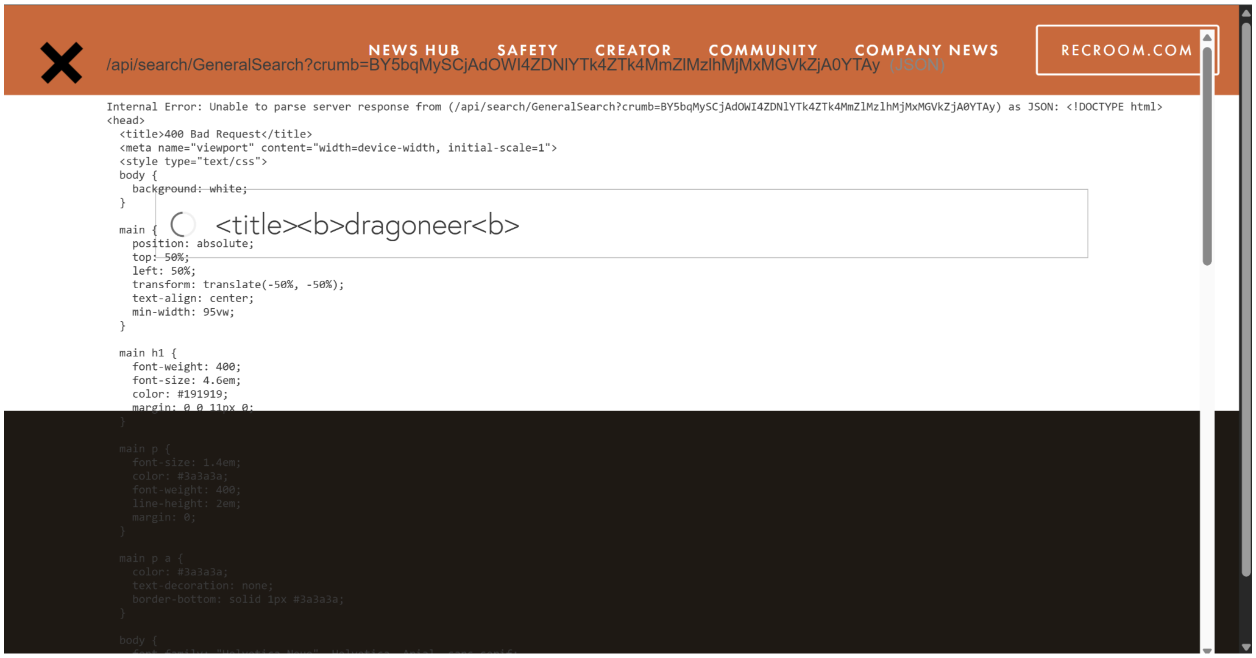
<title><b>dragoneer<b>.

The reason for this is because of where the word was appearing in the source code. In the HTML, the word was appearing within a title. Therefore, the title bracket was needed to break out of that and try to attempt to inject the bold script. This did not appear to work because the source code filtered out the angled brackets, as is shown in the attached image.



```
    <!-- This is Squarespace. --><!-- koala-gazelle-6pxg -->
<base href="">
<meta charset="utf-8" />
<title>&lt;title&gt;&lt;b&gt;dragoneer&lt;b&gt; -  </title>
<meta http-equiv="Accept-CH" content="Sec-CH-UA-Platform-Version, Sec-CH-UA-Model"
<meta property="og:site_name" content=" "/>
<meta property="og:title" content=" "/>
<meta property="og:type" content="website"/>
<meta property="og:image" content=""/>
```

A similar attempt was made using burp suite to intercept the request. Attached below is the adjusted, intercepted GET request made by the website and the resulting page.


NEWS HUB
SAFETY
CREATOR
COMMUNITY
COMPANY NEWS
RECROOM.COM
/api/search/GeneralSearch?crumb=BY5bqMySCjAdOWI4ZDNlYTk4ZTk4MmZlMzlhMjMxMGVkZjA0YTAy (JSON)
<title><b>dragoneer<b>

```
Internal Error: Unable to parse server response from (/api/search/GeneralSearch?crumb=BY5bqMySCjAdOWI4ZDNlYTk4ZTk4MmZlMzlhMjMxMGVkZjA0YTAy) as JSON: <!DOCTYPE html>
<head>
  <title>400 Bad Request</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style type="text/css">
  body {
    background: white;
  }

  main {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    text-align: center;
    min-width: 95vw;
  }

  main h1 {
    font-weight: 400;
    font-size: 4.6em;
    color: #191919;
    margin: 0 0 11px 0;
  }

  main p {
    font-size: 1.4em;
    color: #3a3a3a;
    font-weight: 400;
    line-height: 2em;
    margin: 0;
  }

  main p a {
    color: #3a3a3a;
    text-decoration: none;
    border-bottom: solid 1px #3a3a3a;
  }

  body {
```

The result of forwarding the adjusted request resulted in a 400 “Bad Request” error, meaning some sort of check is being done on the backend to keep an attacker from injecting scripts. The same result was found from multiple other payloads, tested both on site and with Burp Suite.

Thus, from the three targets tested, there were no vulnerabilities found. The only recommendation is that the recroom.com search queries be adjusted to better work like the developer portal and rec.net as those appeared to be far more secure. Just because no cross site scripting vulnerabilities were found, does not mean they do not exist.

SQL Injection:

SQL Injection attacks are similar to cross site scripting attacks, but involve injecting some sort of malicious SQL query to gain access to an applications database. SQL is a common language used for database interactions such as querying for specific datasets, updating info, and adding or deleting data. If an application is written in a way such that it takes input from a user, and simply injects that input into a query or some other SQL statement that interacts with their backend, then an attacker can develop a payload such that it allows them to inject their own SQL command in the input. Doing so to an unprotected application will allow them to execute whatever commands they want to the app's database.

When looking for SQL Injection vulnerabilities, the focus was on the three website applications. There were multiple payloads and multiple entry points used. There were two types of SQL injections that were utilized. The first is a simple query injection used wherever there was a login point. At this login point simple scripts were used in place of typical login credentials in an attempt to get information back. The second kind of SQL injection used is called a union injection and involves injecting a query to unionize the data retrieved from the program's original query, with new data that the query typically won't ask for.

When attacking the first web application, Rec.net, the first place of attack was the search feature. The search feature, as previously mentioned, is used by users to find people, rooms, events, and inventions. This is the first place a union injection was used. The attempts were made using the following payload:

```
Dragon' UNION SELECT username, password FROM users-
```

The way that this payload works, is that it assumes it injects the user's input in a "where" clause in the SQL query. By doing so, it will search for people, rooms, events, or inventions with the name “Dragon,” and unionize that dataset with a query of usernames and passwords from the “users” table. The idea is that it would then return to the attacker the usernames and passwords for users on their website. This was also attempted with other keywords in the payload, rather than just with the keyword “dragon.” The attempts yielded no results. Each payload was re-attempted using Burp Suite to intercept the GET request using a proxy, and re-enter the injected SQL query in an unfiltered way. The same response resulted. The other place of attack on this web application was the login page. This is where the more simple SQL injection attack was used. The following payload was injected in the username portion of the login page:

`administrator’ --` or `admin’ --`

The idea behind these payloads is that it would send a query that would search for all users in the database whose name was “admin” or “administrator.” The dashes at the end are used to comment out the rest of the query, so it would do the search without requiring the correct password. These payloads also failed. They were also re-attempted using the proxy to inject unfiltered SQL, but yielded no better results.

The other two websites were attacked in almost identical ways. The developed portal was attacked using the simple injection payload on its login page, using both straight input and a proxy. The developer portal was attacked using the union injection payload in its product search feature, using both forms of injection. The main website Recroom.com was attacked in the same place as it was attacked for cross site scripting– the new blog search query. This website was only attacked with the union injection, using both forms of injection. Neither website yielded any results and no vulnerabilities were found.

From the multiple attacks used on all three target sites, the applications all seem to be clear of SQL injection vulnerabilities. This could be handled by multiple things. The first, and possibly most obvious reason, would be that they do not use SQL for their database queries. This, however, is highly unlikely, knowing that they use Azure to host their backend services, and Azure's databases are based on and written using SQL. The more likely reason for the lack of vulnerabilities is probably one of two things. The first being that they filter input, which doesn't seem to be the case as the entire query seems to be encoded when it is added to the GET request. The filtering could, however, be done on the backend, rather than the frontend. The second possibility is that they utilize some type of prepared statement to do their SQL queries. Prepared statements are a tool used to create queries in a way that it treats user input as parameters, rather than just concatenating the input with a prebuilt SQL query. This is a much safer way to do SQL queries and require much more advanced payloads to get past. Overall, from the research done for this report, the targeted sites seem to be vulnerability free when it comes to SQL injection attacks.

Path Traversal:

Path traversal (which is also sometimes called directory traversal) is a common way of navigating to different parts of a web application that are not supposed to be accessed by general users. Typically, this is done by just adding to the end of a URL various relative path markings. For example, if you have something like https://example.com?page=index.html then you can do:

https://example.com/../../../../../etc/passwd

This will, if the application is not properly handling path traversal attacks, allow an attacker to grab the passwd file which includes all the users on a system. While this file may not be important, it proves that a system is vulnerable and then subsequent attacks can be used in order to find sensitive information.

One of the markers for an application that may be potentially vulnerable to this kind of attack is files being included in the URL. In the above example, it can be noted that ".html" appears which hints at the fact that there is a file called index.html living somewhere on the server. From this, we can make an assumption that the server is able to respond to relative paths being provided. In Linux, which many servers for web apps run on, you can use relative paths such as "../" to move up one directory. You can also chain multiple of these together to keep moving up until you hopefully hit the root directory.

A nice thing about relative paths is that they can not go past root so you can simply include as many "../" as you want. If you have too many you will simply end up at root. From there we can then move into the directory we want by just adding a path from the root to the target file. Looking at the above example this would be the "/etc/passwd/" path.

Some web apps, while attempting to use path traversal, will not allow you to include your own markings into the URLs that are used in file paths. This means that an attacker would not be able to move out of a directory by using "../". In order to circumvent this, it is sometimes possible to use escape codes to represent the "../". The escape code for a period is "%2e" and the escape code for the slash is "%2f". Putting these together will get you "%2e%2e%2f". Back to the above example, it would look like the following:

https://example.com/%2e%2e%2f%2e%2e%2f%2e%2e%2fetc/passwd

No identifiers for path traversal were located in the Recroom websites (namely https://*.rec.net/*, https://api.rec.net/, and https://devportal.rec.net/) however, attempts were still made to use directory traversal. One example of such an attempt was on https://rec.net/. The first thing attempted was simply just adding the relative paths to the end of the URL so it appeared as follows:

https://rec.net/../../../../../etc/passwd

This did not result in the user list of the server being returned so the next attempt was to use escape sequence encoding. The URL appeared similar to before but with the above-mentioned escape codes:

https://rec.net/%2e%2e%2f%2e%2e%2f%2e%2e%2fetc/passwd

This also did not return the passwd file so double encoding was used. Double encoding for "../" looks like %252e%252e%252f (because %25 = %):

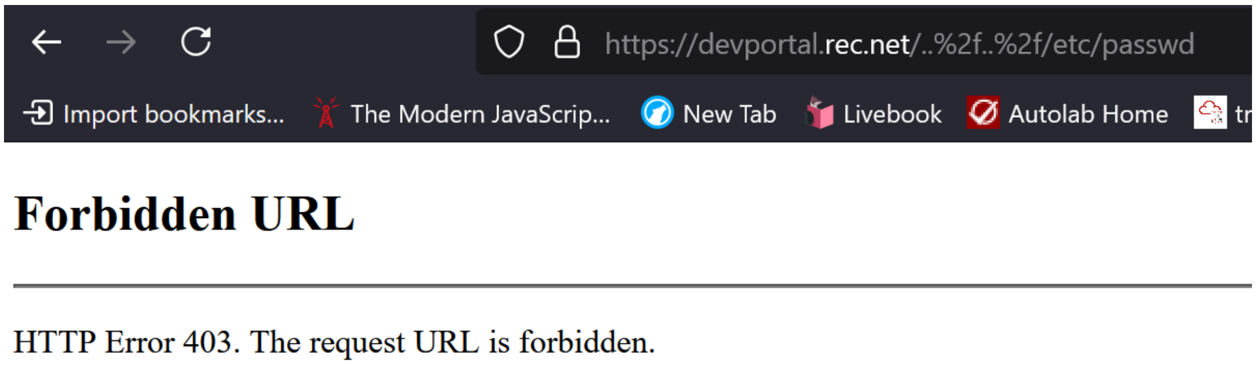
https://rec.net/%252e%252e%252f%252e%252e%252f%252e%252e%252f/etc/passwd

This also did not result in any file being returned on any of the sites (meaning this did not work for the other targets as well. Not just rec.net). While looking for path traversal vulnerabilities, Cloudflare (a company that provides web application security services) makes multiple checks to verify that the requests were from legitimate users by launching captchas before the request would be processed. The web app would also strip out "../" making the URL request echo back as just:

https://rec.net/etc/passwd

This was not the case for all attempts at path traversal, however, most would get truncated.

One interesting case that was found was on the devportal (https://devportal.rec.net/). While searching directory traversal vulnerabilities, the simple encoding for "../" did indeed work and a request was made for /etc/passwd, however, a 403 error was given in a response due to lack of authorization. An image showing as such is below.



https://devportal.rec.net/..%2f..%2f/etc/passwd

Forbidden URL

HTTP Error 403. The request URL is forbidden.

It appears that recroom and the sites listed in the scope section were not vulnerable to path traversal. It would either filter out the request, state the file did not exist, or respond with a 403 in all cases tested. Recroom has done a good job at defending against path traversal attacks.

Insecure Direct Object Reference (IDOR):

Insecure Direct Object Reference (IDOR) is an attack where a hacker is able to directly modify object identifiers in a web application to access data or functionality they shouldn't have permission to access. For example, after logging into a site, the URL might look like this:

https://example.com/user/374

If the application isn't properly validating access, an attacker could simply change the URL to something like:

https://example.com/user/1

If this kind of vulnerability exists, it would allow an attacker to access someone else's account without proper authentication. In many systems, user ID "1" is often tied to the administrator account, which could give the attacker full access to the system. Exploiting this could result in a loss of confidentiality, integrity, and availability of sensitive data, including personally identifiable information (PII).

While IDOR is most commonly known for allowing unauthorized access to user accounts, it can also be used to leak restricted information or interact with parts of a website not meant for public access. Such things include internal admin tools, billing records, or support tickets.

Furthermore, while the most common tactic for identifying IDOR is through looking at URLs, another place to check is also in the POST body of web applications. Not all objects that can be directly referenced are visible on the page itself but using tools like Burp Suite can allow you to see more objects that can be directly changed.

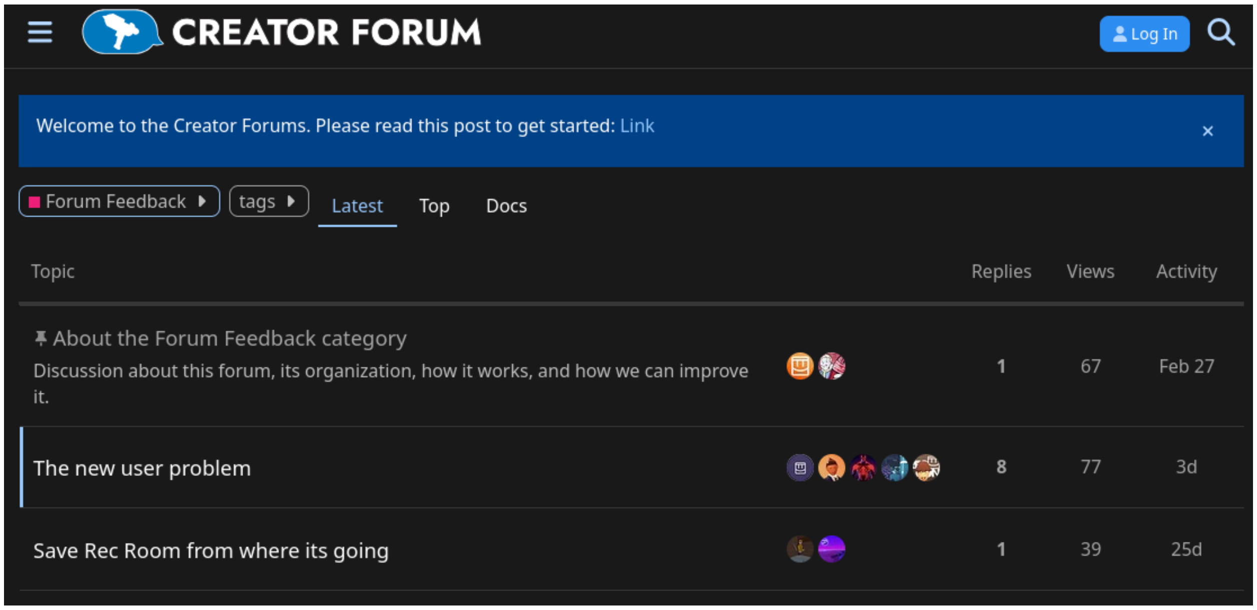
In this project, the one place where an object was able to be referenced was in the https://forum/rec.net/c/official/ page. Any user can manipulate a number in the URL to switch between different forum posts. For example, a user could type in something like:

https://forum.rec.net/c/official/4

Doing so would bring up a page with a list of other forums. However you can change the number (4 in this case) to anything. If you change it to:

https://forum.rec.net/c/official/2

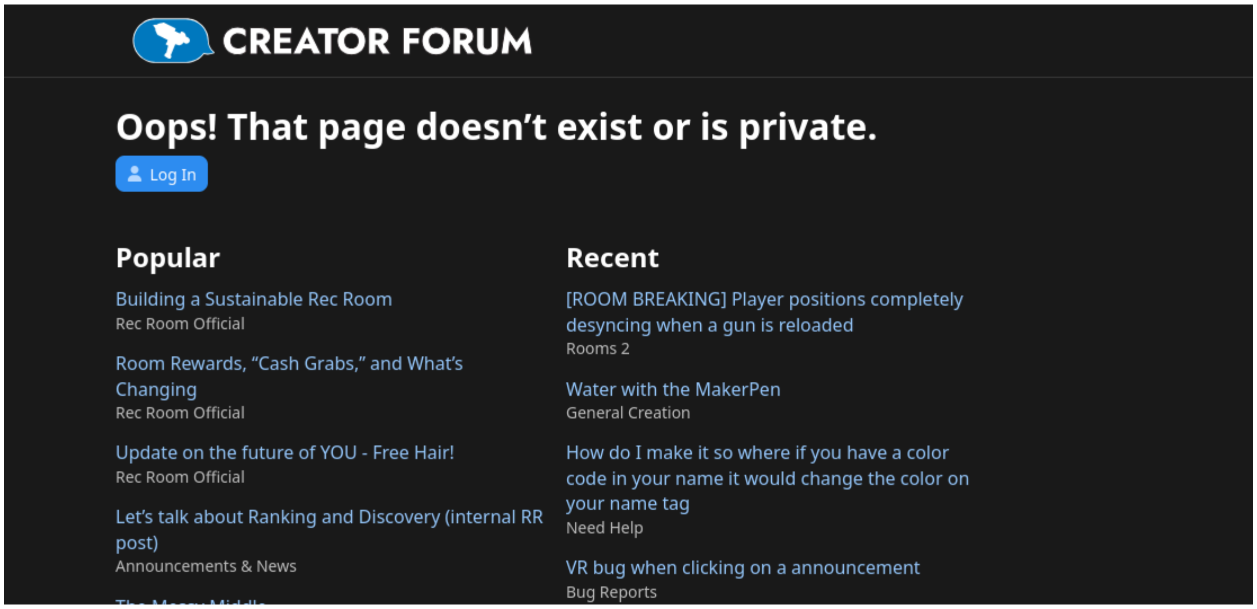
Changing the end of the URL brings up a forum discussing feedback (see the below screenshot). In an attempt to leak information that system administrators potentially left I attempted to change it to a variety of different numbers.



Welcome to the Creator Forums. Please read this post to get started: Link

Topic	Replies	Views	Activity
About the Forum Feedback category Discussion about this forum, its organization, how it works, and how we can improve it.	1	67	Feb 27
The new user problem	8	77	3d
Save Rec Room from where its going	1	39	25d

However, upon doing so, it is quickly made apparent that any forum with sensitive information is made private. Using numbers like 1, 3, or 5 bring up the below page.



Oops! That page doesn't exist or is private.

Popular

Building a Sustainable Rec Room
Rec Room Official

Room Rewards, "Cash Grabs," and What's Changing
Rec Room Official

Update on the future of YOU - Free Hair!
Rec Room Official

Let's talk about Ranking and Discovery (internal RR post)
Announcements & News

Recent

[ROOM BREAKING] Player positions completely desyncing when a gun is reloaded
Rooms 2

Water with the MakerPen
General Creation

How do I make it so where if you have a color code in your name it would change the color on your name tag
Need Help

VR bug when clicking on a announcement
Bug Reports

This shows that the Recroom team did a good job at securing forums and preventing people from gaining access to sensitive information by marking certain rooms as private. It is my belief that recroom and all of its web pages are safe from IDOR attacks.