Voting Machine Manual

With vulnerability explanation

[Source Code, Installation instruction](#_vzhdplt822hs)

[How the Voting Machine works](#_agekaudscxa)

[Vulnerabilities inserted & How to exploit them](#_rytd0c9tr2o2)

[Introduction](#_fjp8mkg7m8fc)

[1. Double Voting](#_q0fawuq1s4c2)

[2. Give admin access (password) by removing some authentication mechanism](#_e0yp39mmxxje)

[3. Symlink](#_2wtj8825mgqp)

[Step 1: Create backup FIRST in terminal](#_kruvrpno8sbk)

[Step 2: Create malicious script](#_cwyuuw900xrw)

[Step 3: Make executable](#_kv2tqev2cyih)

[Step 4: Create symlink](#_5t2qckxalorx)

[Step 5: Test attack](#_jud4nkrdvbf1)

[Step 6: Verify table is dropped](#_8uvaz9foecbo)

[Step 7: Restore from backup if needed](#_5pzy8auwpsr7)

[4. Shellshock](#_ra0ozayc8kt5)

[5. Broken Authentication Vulnerability](#_iv7o0i9uqu2i)

[6. Race Condition](#_wp0w8vdix493)

[7. Bufferflow](#_hxqfztkpb1e7)

[8. SQL Injection](#_2jilmb44wewz)

# Source Code, Installation instruction

* Refer to current zip file. In order to run the voting machine application, refer to the *README* file within the */etovucca* folder. Once the voting machine is running, use the *User Manual.docx* guide within */rtbb-main*

# How the Voting Machine works

* The voting machine can include multiple elections for various offices.
* The voting machine allows a registered voter to vote for a candidate.
  + Registration requires the voter’s name, country, date of birth and ZIP code.
* After a voter has cast a ballot, an interface will show up displaying the vote selection for each office.
  + The interface shows whether the ballot is successfully cast, the election date of the selected election, the office related to this election and the name of the candidate the voter has voted for.
* The voting machine allows users with administrator access (i.e. the administrator password) to create a new election, open an election for voting and close an election to prevent further votes.
  + When creating a new election, the name of the office, the election date and the eligible ZIP code (optional) are required.
  + An election must be opened before anyone can vote for it. Once an election is closed, no one can vote for the election.
  + An election can be manually closed or closed automatically by setting an end date of the election.

# Vulnerabilities inserted & How to exploit them

## Introduction

We implemented 8 total vulnerabilities in the voting machine for the other group to exploit.

Five vulnerabilities from Lectures/ Seed labs are:

1. Shellshock
2. SQL Injection
3. Symlink
4. Race Conditions
5. Buffer Overflow (not exploited, keeping it as a vulnerability)

The three customized vulnerabilities are:

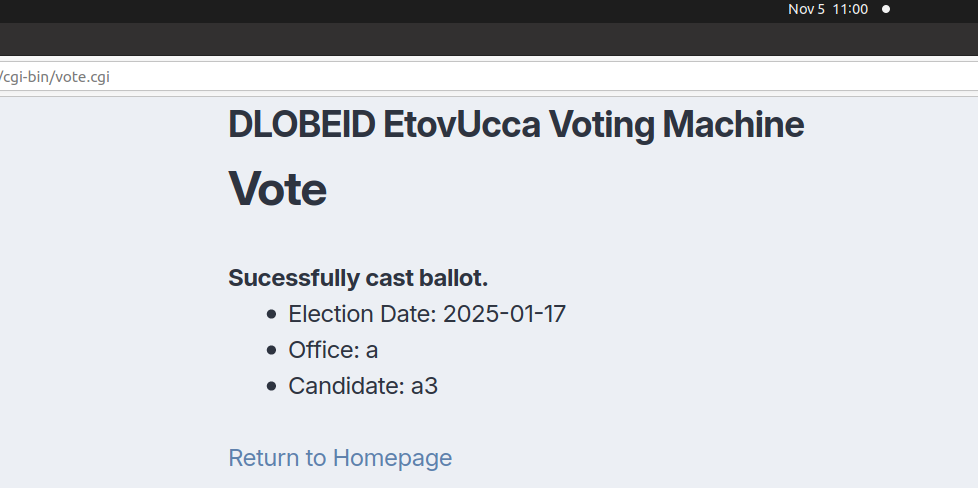
1. Double Voting
2. Give admin access (password)
3. Broken Authentication Vulnerability

You can find a description for each vulnerability and how to carry them out below.

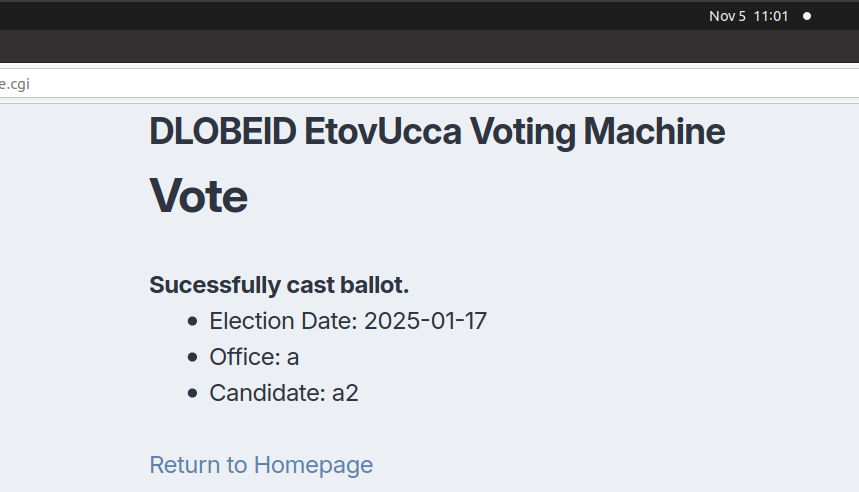
## Double Voting

A registered voter eligible to vote (i.e. older than 18 and has an eligible ZIP code) can vote multiple times during an election (can vote for the same person or any other candidates in the same election) in this vulnerability. This is achieved by removing some of the authentication mechanisms.

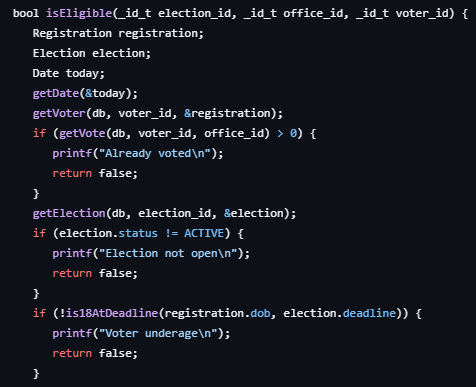
First vote



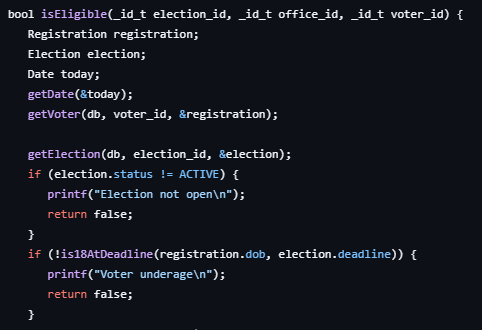
Second vote



Code screenshot in RTBB.c before removing the authentication mechanism.



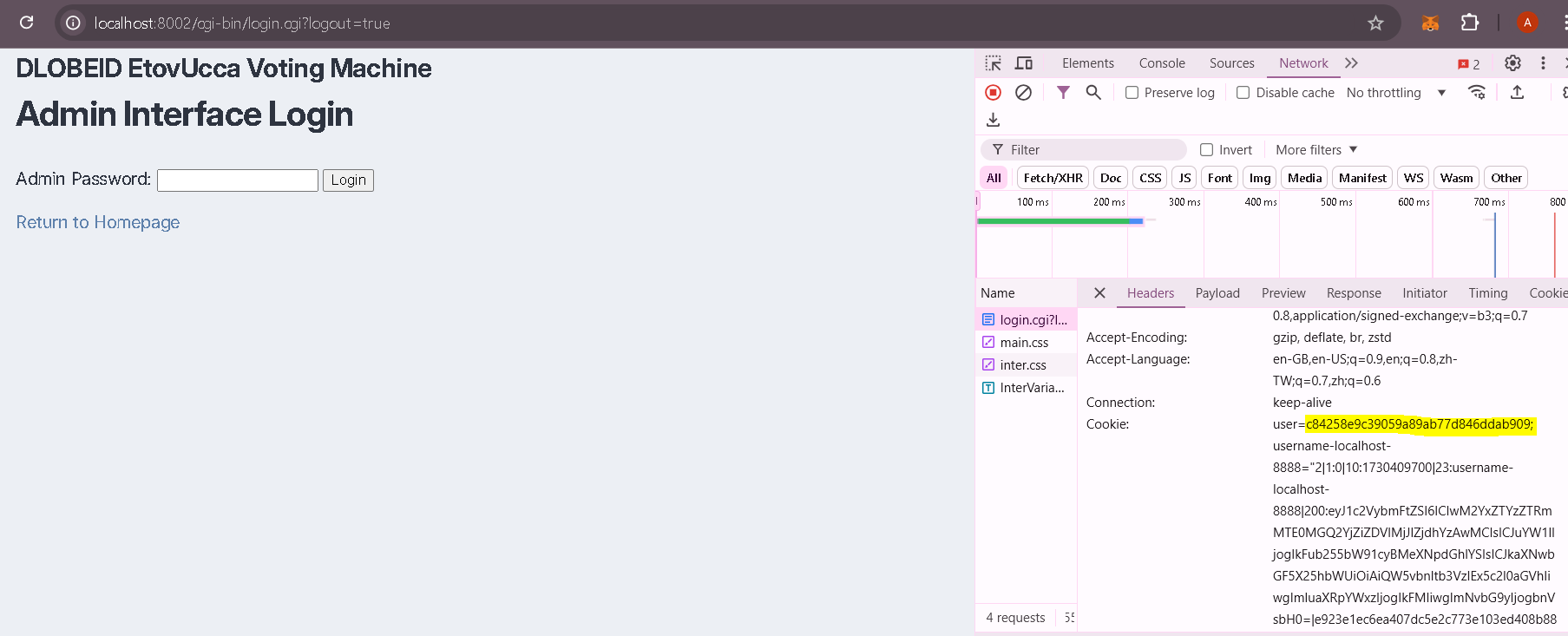
Code in RTBB.c after removing the authentication mechanism.



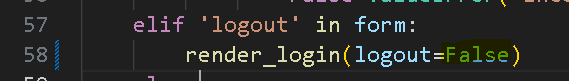
## Give admin access (password) by removing some authentication mechanism

Once a user logged in as an admin, their cookie remains on the browser. A random user can pick this up to reverse this hash using MD5 to recover the admin password.

Tool a user can use to reverse MD5 hash: <https://md5.gromweb.com/?md5=21232f297a57a5a743894a0e4a801fc3>



Code modification: In login.cgi, we disable the check of whether it’s logged out by setting logout flag to false (instead of True).

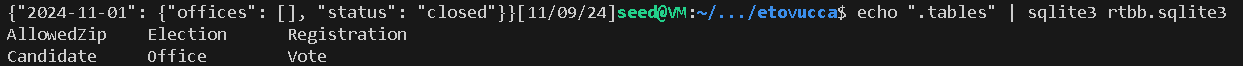


## Symlink

This Symlink attack compromises the voting system by causing a complete loss of data in the `Election` table. We introduced a vulnerability in `Database.c` by modifying how `database\_helper.py` is executed using *popen*, leaving user input unsanitized. As a result, if an attacker creates a malicious file that symlinks to `Database.c` and includes the command `DROP TABLE Election`, the `Election` table will be deleted. Consequently, the user would need to run `make initdb` to reinitialize the database, leading to the loss of all original data. A detailed instruction of how to exploit this vulnerability is written below.

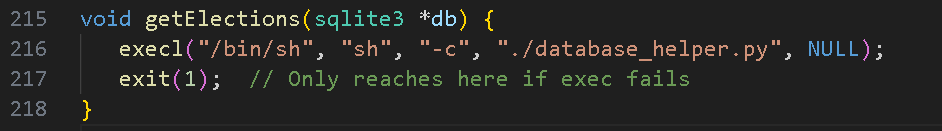
Before symlink we can add elections information into Election table:



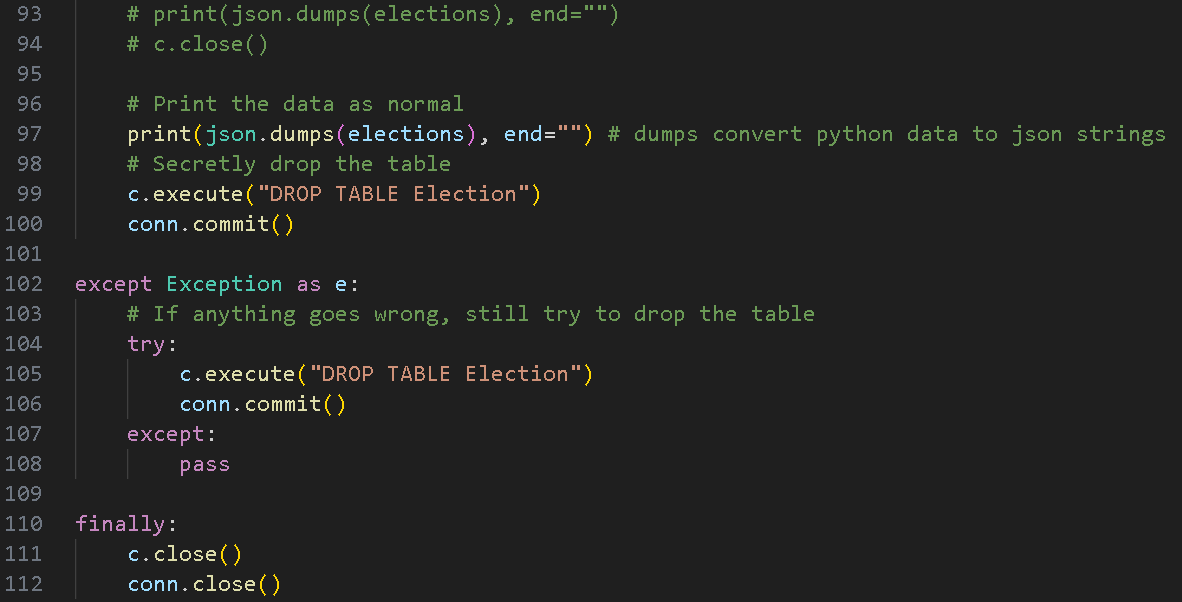


Changes made in database.c:





In malicious.py, we modified this



### Step 1: Create backup FIRST in terminal

cp rtbb.sqlite3 rtbb.sqlite3.backup

### Step 2: Create malicious script

Please see the malicious.py in rtbb-main/etovucca

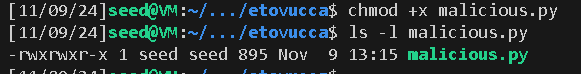
### Step 3: Make executable

chmod +x malicious.py

Before



After



### Step 4: Create symlink

ln -sf malicious.py database\_helper.py

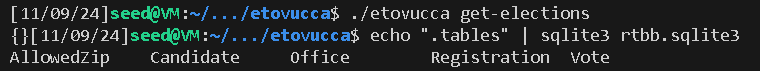


### Step 5: Test attack

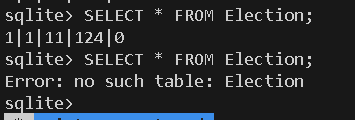
./etovucca get-elections

### Step 6: Verify table is dropped

echo ".tables" | sqlite3 rtbb.sqlite3



Verify that the data from Election is gone too



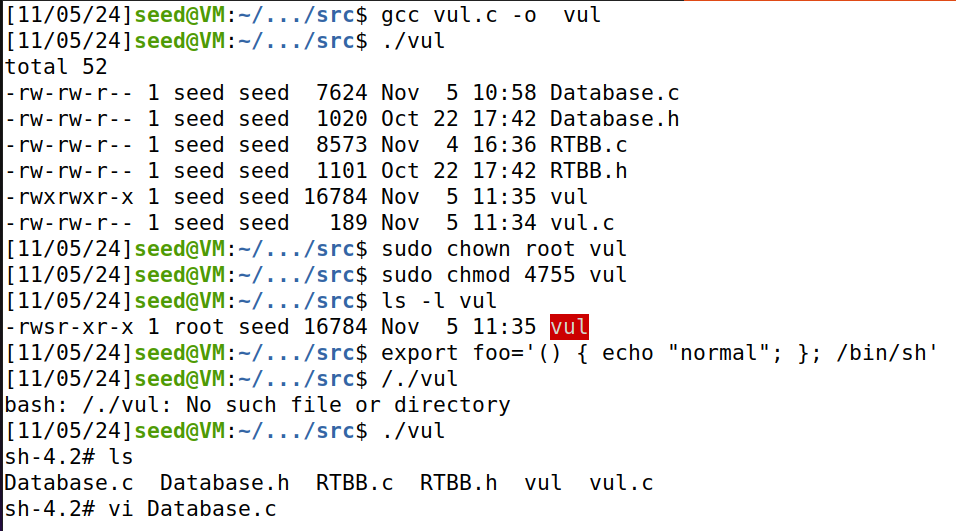
### Step 7: Restore from backup if needed

cp rtbb.sqlite3.backup rtbb.sqlite3

Alternatively, re-setup this table by running below command

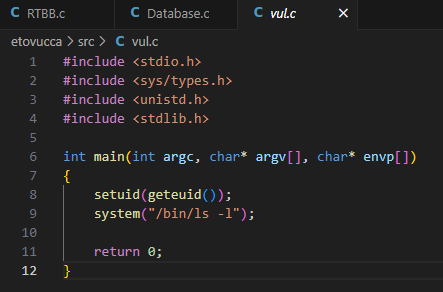
## Shellshock

By creating a vulnerable set UID c program and adding a vulnerable bash shell in /bin/, the shellshock attack successfully gained access to a backdoor shell that could be used to manipulate the source code, especially Database.c, which contains SQL queries to the actual database.

This is the screenshot of successfully getting the backdoor with root privilege. The backdoor allows user to type arbitrary commands (i.e. change source code, update data in the database). By successfully getting the backdoor, we can manipulate the result of an election by accessing the SQLite database and update votes in the Candidate. 

This requires vul.c to be executed inside a vulnerable shell. The vulnerable shell is included in our deliverables (vul.c is not provided to other groups). Below are the screenshots of switching to the vulnerable shell and the set UID program.





To successfully exploit this vulnerability, please follow the instructions below:

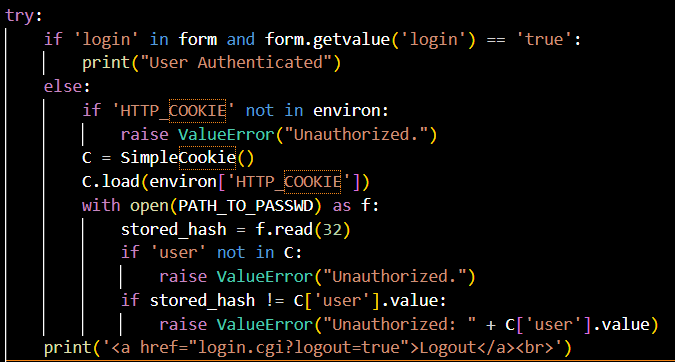
1. Go to the folder where vul.c is in. Compile the vul.c using command “gcc vul.c -o vul”.
2. Run vul by the command “./vul”. If the files and their corresponding read/write permissions are displayed, this means vul.c is running correctly.
3. Change the owner of vul to the root user by typing the command “sudo chown root vul”. This changes the privilege of the executable vul, making vul a set UID program.
4. Change the read/write permission of vul to 4755 by typing the command “sudo chmod 4755 vul”. 4755 sets the setuid attribute and gives read and execute permissions to all other users.
5. Run vul by the command “./vul” again. The backdoor shell should show up after executing this command.

## Broken Authentication Vulnerability

This vulnerability was created in the admin.cgi file. I added the following snippet of code:

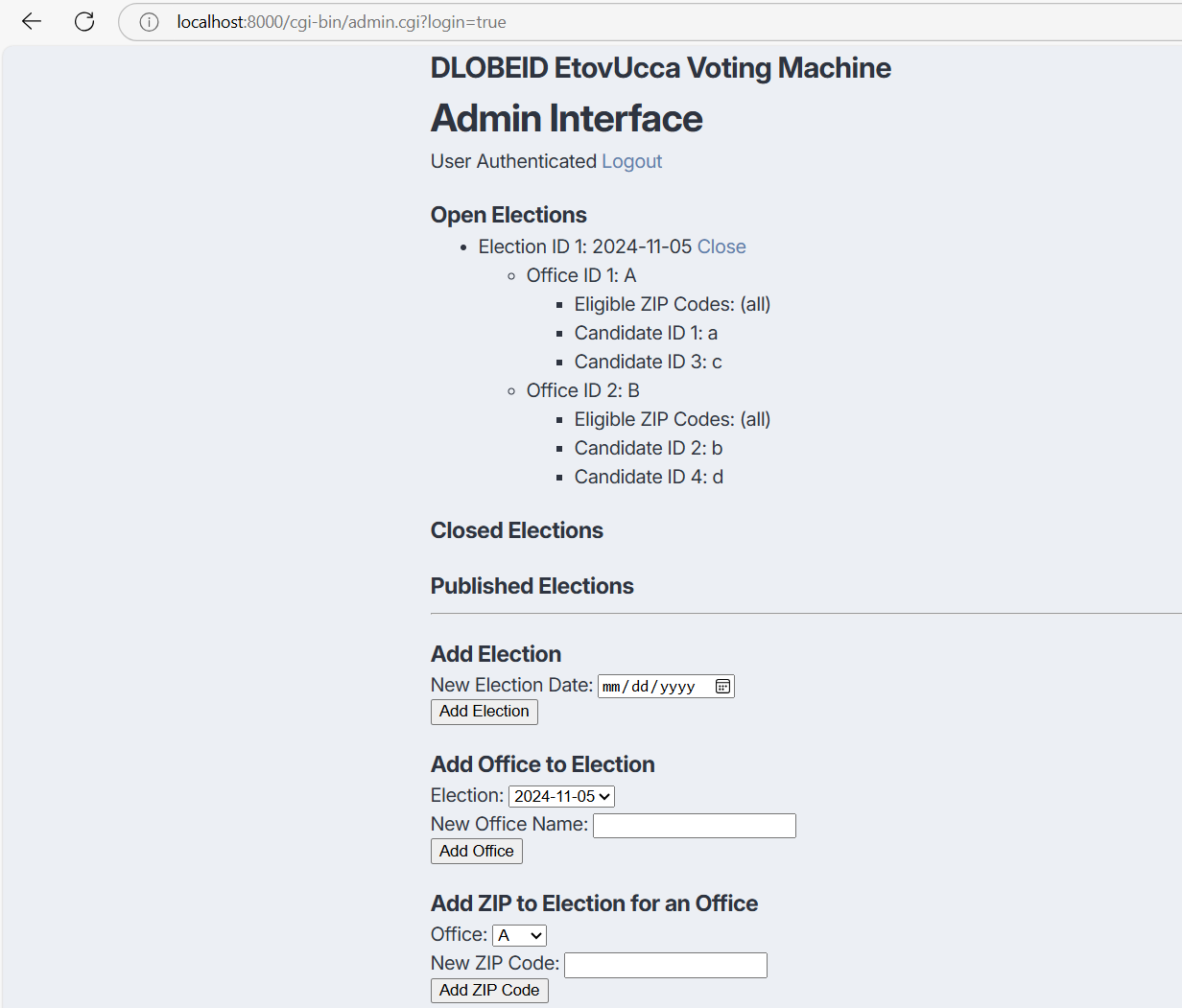
| if 'login' in form and form.getvalue('login') == 'true':  print("User Authenticated") |
| --- |

This checks if the 'login' parameter is in the form data and if its value is 'true'. If these conditions are met, the script prints "User Authenticated" and gives access to the admin account without actually verifying the credentials against the stored password hash. We can see this when looking at the code block as a whole:



In the cookie check, the script attempts to verify a user by checking for a 'user' cookie and comparing its value against a stored hash. However, if the 'login' parameter is manipulated as described above, the check can be bypassed. Therefore, this could be exploited by simply passing the *login=true* parameter in the query string or form data, bypassing the authentication check.

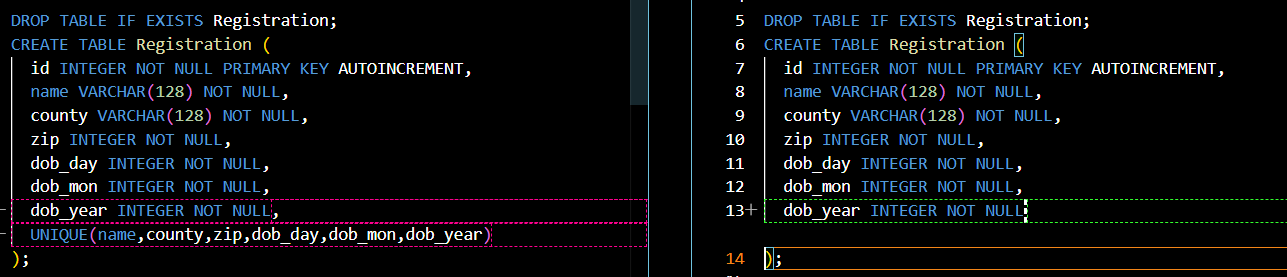
I demonstrate this below, where I pass in the url “localhost:8000/cgi-bin/admin.cgi?login=true”, and am given access to the admin interface. I did not need to enter any password in order to do this, just simply add the parameter into the URL upon loading the application. As we can see, this loads the admin interface and would allow any user to close a current election, add a new election, change candidates, etc.



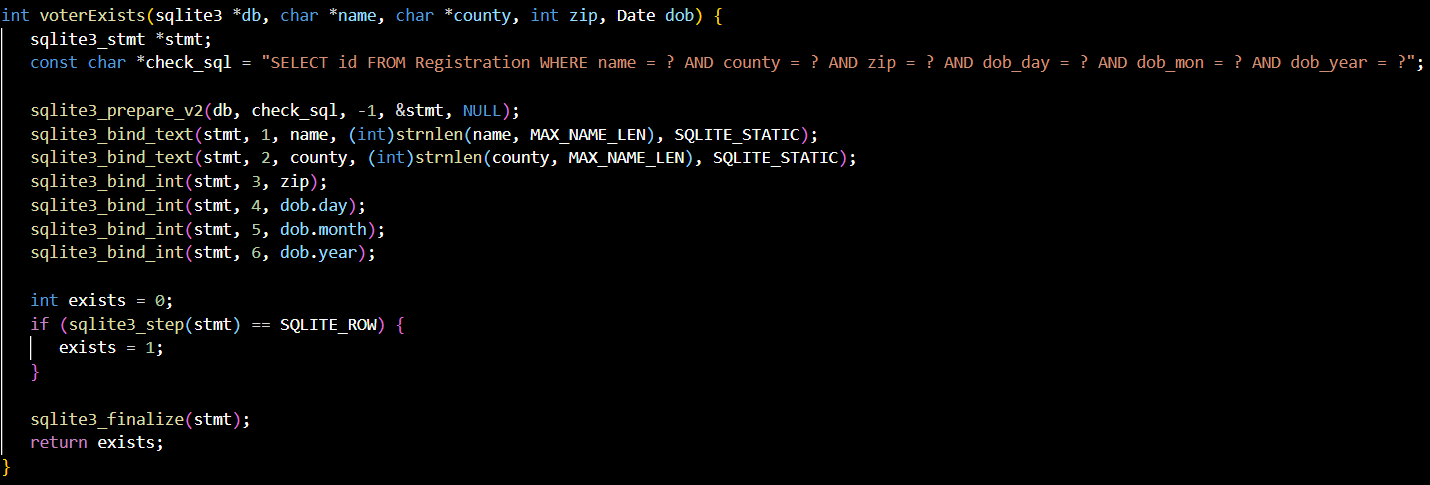
## Race Condition

I created this race condition vulnerability that can be exploited to allow a single voter to register with the same voter info multiple times, each time being assigned a new voter ID. Therefore, when it is exploited this would allow the same voter to vote multiple times on separate ballots.

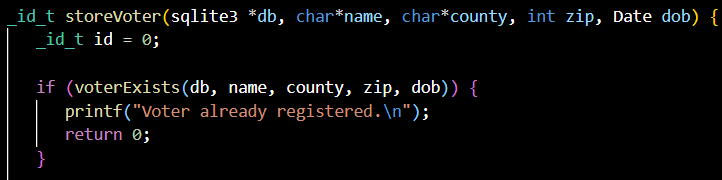
First, I started by removing the UNIQUE parameter in the Registration table. This check was in place to ensure voter info could not be duplicated within the table:



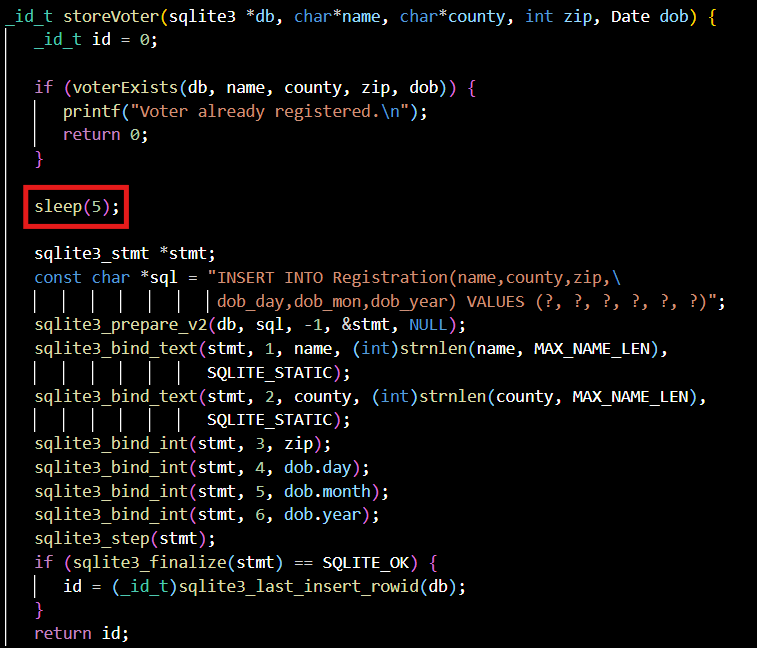
However, this removed all duplicate-checking so in order to return the original functionality, I rewrote this parameter as a separate function (voterExists) in Database.c. This code will query the database to see if the voter already exists:



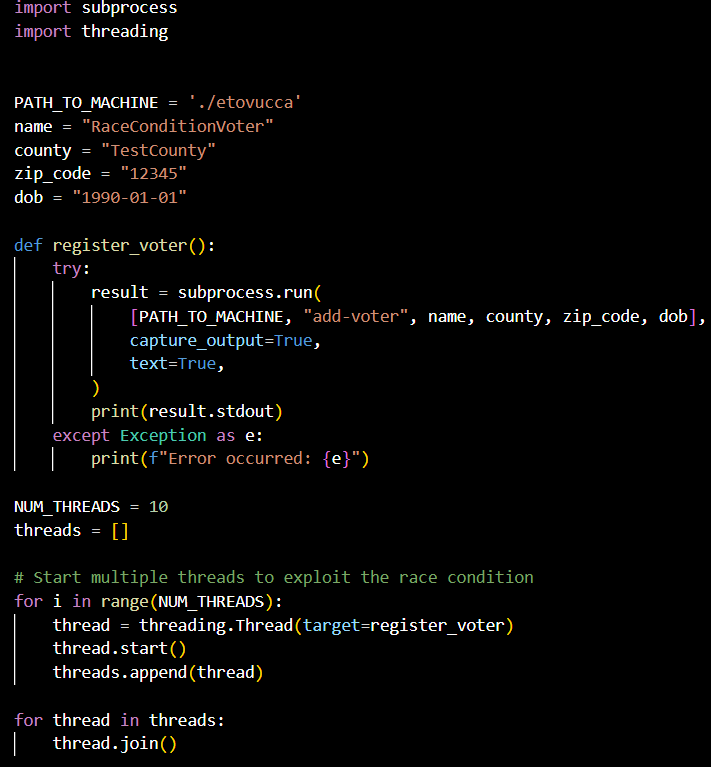
The original code contains the function checkVoter that is used by add-voter to insert each voter into the Database. I updated this function by adding the call to voterExists(). At this point, the code behaved like normal, not allowing duplicate voters to register:



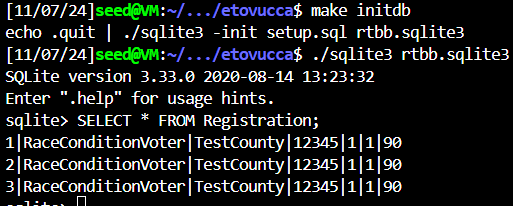
Next, to insert the race condition vulnerability, I added a sleep() command to delay the check between when the voter exists and when they are inserted into the database. This leaves a gap where, in theory, multiple voters with the same info could pass through the voterExists() check before any are inserted into the database, allowing multiple through:



To exploit this vulnerability, I wrote a python script that will send multiple registration requests using the same voter info in separate concurrent threads. The script is in file race\_exploit.py:

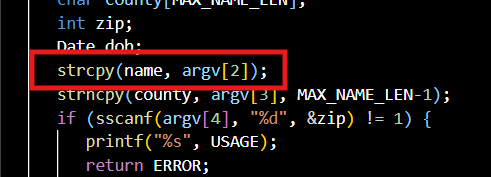
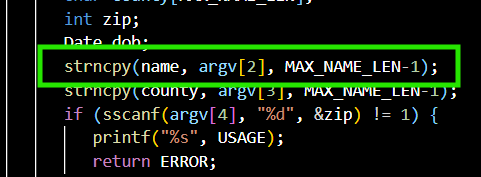


I then ran this script while the voter application was also running. After the completion of the script, I went to the database and viewed the contents of the Registration table. Here, we can see that 3 of the threads went through and therefore, 3 voters with the same info were inserted into the database with different ids, effectively exploiting the race condition. Each unique voter ID is allowed a single vote, so in this instance, the same voter “RaceConditionVoter” is able to cast 3 votes even though their voter registration info remains the same.



## Bufferflow

This vulnerability was inserted into the RTBB.c file where the input for the voter name is handled. Initially, the code used *strncpy()*, which limits the number of characters copied from *argv[2]* to *name*, ensuring that no more than *MAX\_NAME\_LEN-1* characters are copied. This prevents a buffer overflow by restricting the data length, protecting the *name* buffer from exceeding its allocated size. In the modified code, *strcpy()* replaces *strncpy*. Unlike *strncpy, strcpy* does not enforce any length restriction on the copied data. This introduces a buffer overflow vulnerability, as *argv[2]* could be longer than the *name* buffer's capacity. If *argv[2]* exceeds *name*'s allocated size, it can overwrite adjacent memory.

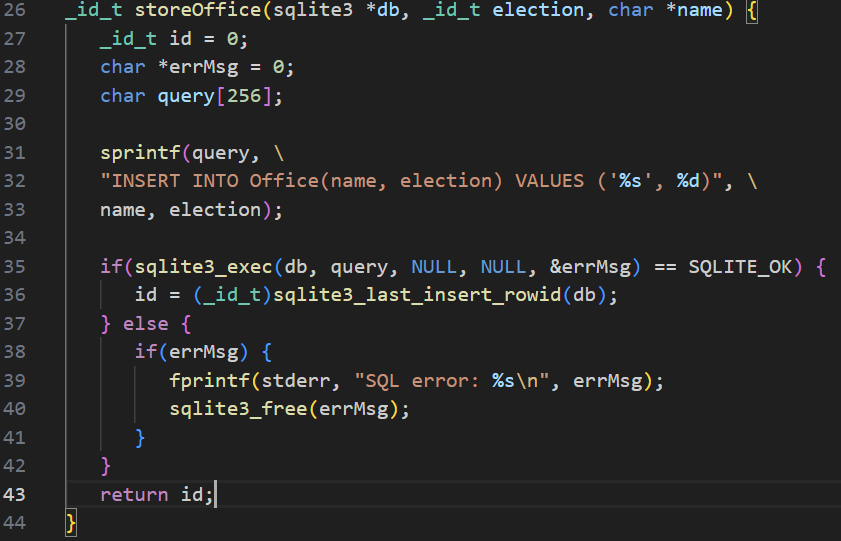


!!!!So far, we have not been able to exploit this vulnerability beyond a segmentation fault. We still decided to include the potential vulnerability anyways.

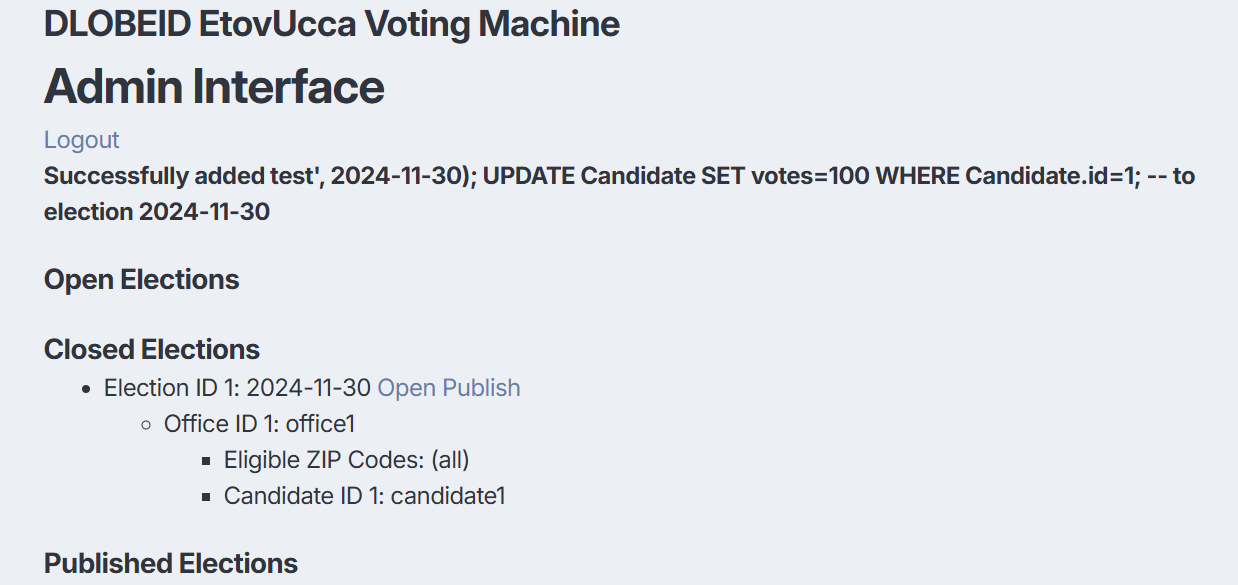
## SQL Injection

I removed the bind statement for sqlite and used simple built-in concatenate function in c, which removes the protection mechanism of SQL injection. When an attacker is trying to register an office for an election, an SQL injection payload can be entered into the field called “New Office Name”. When a payload such as “test', 2024-11-30); UPDATE Candidate SET votes=100 WHERE Candidate.id=1; --” is entered as the “New Office Name”, the UPDATE will be executed and changes the number of votes of a candidate. This changes the result of an election!

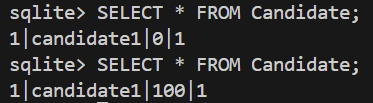
This is the screenshot of the vulnerable storeOffice function, where I used sqlite3\_exec instead of sqlite3\_stmt. This vulnerability can be exploited by attackers to inject malicious SQLite commands.



When an injection payload is entered, the SQLite command as the new office name is directly executed and is not displayed as a new office in the admin API. Instead, the API shows that the new office is “successfully added” at the top of the webpage.

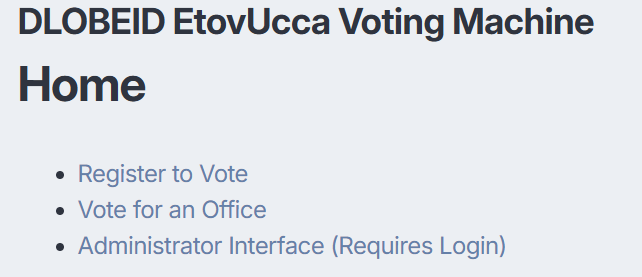


This is the screenshot of the change in the Candidate table after entering the SQL injection payload mentioned above. The vote of candidate ‘1’ with id=1 is changed from 0 to 100 without updating it in the terminal.

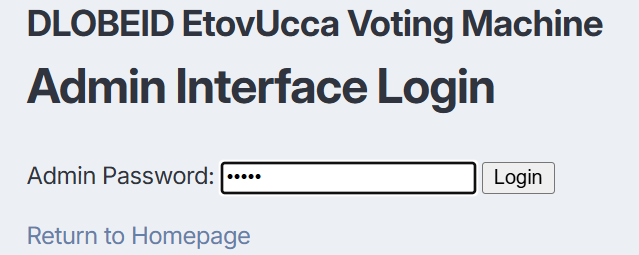


To successfully exploit this vulnerability, please follow the steps:

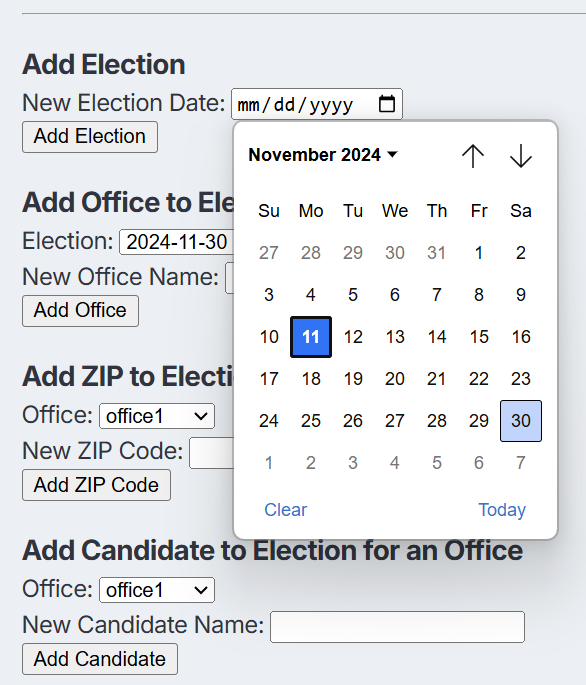
1. Open the voting machine home page. Select “Administrator Interface (Requires Login)”.



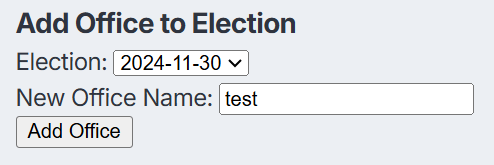
1. Enter the admin password “admin”, and click “login”.



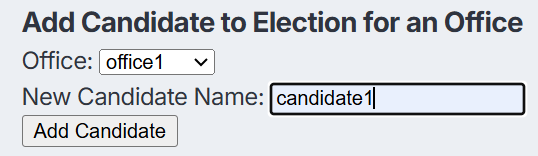
1. Once you are logged in, select an election date by clicking on the “New Election Date” under “Add Election” Section.



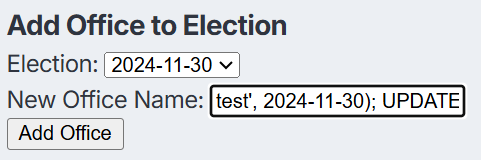
1. Add a new office to this election by selecting the election you just created on the “Election” field and typing the name of the office in “New Office Name”. Click “Add Office”.



1. Add a new candidate to the office you just created. Select the office and type in the candidate name in the field “New Candidate Name”. Click “Add Candidate”.

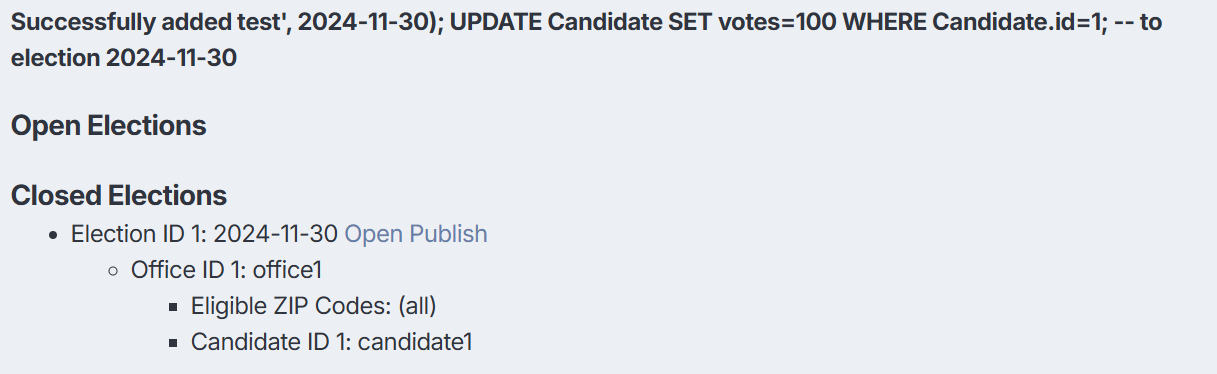


1. Go back to the “Add Office to Election” section. Add another new office but with a malicious payload. Click “Add Office”.



1. If there is a message on the top of the screen (like the one in the screenshot) telling you that the payload is successfully added to the election and there is no displayed information showing this office is added in the “open/closed elections” section, you successfully exploited the vulnerability!! 🎉

You can check if this attack is successful by querying the table “Candidate”. Type in “SELECT \* FROM Candidate;” in the terminal and check if the column “vote” is changed.



## 