Remote code execution in Homebrew by compromising the official Cask repository

Remote code execution in Homebrew by compromising the official Cask repository - RyotaK, blog.ryotak.net.

- Published: 2021-04-21
- Original: <https://blog.ryotak.me/post/homebrew-security-incident-en/>
- Current location: <https://blog.ryotak.net/post/homebrew-security-incident-en/>
- Preserved from: https://blog.ryotak.net/post/homebrew-security-incident-en/ (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Remote code execution in Homebrew by compromising the official Cask repository

* *2021-04-21* * 2092 **HomebrewVulnerabilityRubySupply Chain

: https://blog.ryotak.net/post/homebrew-security-incident/ (
)

(Official blog post about this incident is available here: https://brew.sh/2021/04/21/security-incident-disclosure/)

Preface

Homebrew project is running a “Vulnerability Disclosure Program” on HackerOne, which allows hackers to perform the vulnerability assessment. This article describes a vulnerability assessment that is performed with permission from the Homebrew project’s staff and is not intended to recommend you to perform an unauthorized vulnerability assessment. If you found any vulnerabilities in Homebrew, please report it to Homebrew project’s vulnerability disclosure program.

TL;DR

In the `Homebrew/homebrew-cask` repository, it was possible to merge the malicious pull request by confusing the library that is used in the automated pull request review script developed by the Homebrew project. By abusing it, an attacker could execute arbitrary Ruby codes on users' machine who uses `brew`.

Reason to investigate

One afternoon, I had a slight time before my next appointment1, so I decided to look for an interesting program on HackerOne. As I wanted to find a vulnerability in the software/services I was using, I looked around on my PC, and the `brew` command caught my eyes. Then, I remembered that I saw a program named Homebrew on HackerOne, so I decided to find the vulnerability in it.

Program	Launch date	Reports resolved	Bounties minimum	Bounties average	
Homebrew	04 / 2017	17	-	-	☆

Selection of targets

To select the target, I looked at the policy page of the vulnerability disclosure program. And I noticed that `Homebrew/homebrew-*` repository is in scope. As I'm not good at reading complicated Ruby codes, I decided to find a vulnerability in `Homebrew/homebrew-*`.

Scope

The following sites and applications are within scope for this program:

- The `brew` package manager (Homebrew/brew)
- The Homebrew/homebrew-* official taps

Initial investigation

I think the following two vulnerabilities are common in GitHub repositories:

- Leakage of API tokens that has permission against the repository
- Vulnerabilities in the CI script that is used by the repository

So, I started to check these 2 vulnerability types on repositories that are in scope. To check the first vulnerability, I cloned all repositories created by the member of Homebrew and scanned a token-like string. However, as GitHub has a feature to scan for leaked tokens, this type of vulnerability is not common these days. And as expected, I couldn't find any valid tokens.2

Then, I started to read codes to check the second one.

Investigation of CI scripts

Homebrew project uses GitHub Actions to run the CI scripts. 3 So I looked into the `.github/workflows/` directory of each repository.

After reviewing some repositories, I was very interested in `review.yml` and `automerge.yml` of `Homebrew/homebrew-cask`. It looks like `review.yml` checks the contents of the user-submitted pull request, and if that pull request is simple enough (e.g. Bumps version), it'll approve these pull requests. After that, `automerge.yml` automatically merges approved pull requests.

```
- name: Review Pull Request
  id: review
  uses: Homebrew/actions/review-cask-pr@master
  if: always()
- name: Post Pull Request Review
  uses: Homebrew/actions/post-review@master
  with:
    token: ${{ (github.event.pull_request.user.login == 'BrewTestBot' && secrets.GITHUB_TOKEN) || secrets.HOMEBREW_GITHUB_API_TOKEN }}
    event: ${{ steps.review.outputs.event }}
    body: |
      ${{ steps.review.outputs.message }}
  if: always() && steps.review.outputs.event
```

```
automerge:
  if: >
    startsWith(github.repository, 'Homebrew/') &&
    (
      (
        github.event.review.state == 'approved' &&
        github.event.pull_request.base.repo.full_name == github.event.pull_request.head.repo.full_name
      ) || github.event_name != 'pull_request_review'
    )
  runs-on: ubuntu-latest
  steps:
    - uses: reitermarkus/automerge@46b8b95bac3325964476acc56f9160a98d1e7291
      with:
        token: ${{ secrets.HOMEBREW_GITHUB_API_TOKEN }}
        merge-method: squash
        squash-title: true
        do-not-merge-labels: 'do not merge,awaiting maintainer feedback,awaiting user reply'
        pull-request: ${{ github.event.inputs.pull-request }}
        dry-run: ${{ github.ref == 'refs/heads/debug-automerge' }}
```

Investigation of review.yml

The ruby script used by `review.yml` 4 fetches pull request contents as a diff file and parses it with `git_diff` Gem. And then, it'll approve the pull request only if all conditions below are met:

- Modifying only 1 file
- Not moving/creating/deleting file
- Target filepath matches `\ACasks/[^/]+\.rb\Z`
- Line count of deletions/additions are same

- All deletions/additions matches `/\A[+-]\s*version "([^"]+)"\Z/` or `\A[+-]\s*sha256 "[0-9a-f]{64}"\Z`
- No changes to format of versions (e.g. `1.2.3` => `2.3.4`)

... etc5

I scrutinized the conditions above, but I couldn't find any flaws in them. So I concluded it's not possible to inject arbitrary codes in these conditions. After that, I was checking other scripts for a while, but for some reason, I couldn't forget about this script. So I decided to dig into this script and started looking at the library that parses the diff file.

Investigation of git_diff

While I was looking into `git_diff` repository, I found an issue that reports wrong parsing of changed lines count. After seeing this issue, I started wondering if I could somehow confuse `git_diff` and disguise the pull request to meet the above conditions.

It seemed that `git_diff` did the following to parse the diff file:

- Split the contents of the file with line breaks
- For each line, check if `^diff --git(?: a/(\S+))?(?: b/(\S+))?` matches, and if so, replace the file information currently being processed with the one that matches the regular expression.
- If step 2 didn't match, check if it matches one of the following regular expressions and if it matches, replace the file path information of the source/destination according to the contents.

```
if a_path_info = %r{^[-]{3} /dev/null(.*)$}.match(string)
  @a_path = "/dev/null"
elsif a_path_info = %r{^[-]{3} "?a/(.*)$}.match(string)
  @a_path = a_path_info[1]
elsif b_path_info = %r{^[+]{3} /dev/null(.*)$}.match(string)
  @b_path = "/dev/null"
elsif b_path_info = %r{^[+]{3} "?b/(.*)$}.match(string)
  @b_path = b_path_info[1]
elsif blob_info = /^index ([0-9A-Fa-f]+)\.\.([0-9A-Fa-f]+) ?(.+)?$/.match(string)
  @a_blob, @b_blob, @b_mode = *blob_info.captures
elsif /^new file mode [0-9]{6}$/.match(string)
  @a_path = "/dev/null"
elsif /^deleted file mode [0-9]{6}$/.match(string)
  @b_path = "/dev/null"
elsif mode_info = /^(old|new) mode ([0-9]{6})$/.match(string)
  if mode_info.captures[0] == "old"
    @a_mode = mode_info.captures[1]
  else
    @b_mode = mode_info.captures[1]
  end
elsif copy_rename_info = /^(copy|rename) (from|to) (.*)$/.match(string)
  if copy_rename_info.captures[1] == "from"
    @a_path = copy_rename_info.captures[2]
  else
    @b_path = copy_rename_info.captures[2]
  end
elsif binary_info = %r{^Binary files (?:/dev/null|"?a/(.*)) and (?:/dev/null|"?b/(.*)) differ$}.match(string)
  @binary = true
  @a_path ||= binary_info[1] || "/dev/null"
  @b_path ||= binary_info[2] || "/dev/null"
elsif /^similarity/.match(string)
  # trash
  true
end
```

- If step 3 didn't match, treat it as a change to the file content, consider it as an addition if it starts with `+`, and deletion if it starts with `-`, otherwise consider it as the original file content without modifications.
- Repeat the steps above and finish once all the lines are processed.

These processes seem to be okay at first glance, but it was possible to change the source/destination file path information multiple times in step 3.

The diff file generated by GitHub will be the following format:

> diff --git a/source file path b/destination file path
> index parent commit hash..current commit hash filemode
> --- a/source file path
> +++ b/destination file path
> @@ line information @@
> Details of changes (e.g.: `+asdf`,`-zxcv`)

Additional lines will be represented by prepending "+" to the line.

This means if the added line matches `++ "?b/(.*)`, it'll be treated as a file path information rather than the change against file contents. And by checking the required conditions above, I noticed that the required condition for the file path being changed is only `\ACasks/[^/]+\.rb\Z`. As mentioned above, the file path information can be changed multiple times, so the above conditions can be bypassed by making the following changes, and the pull request will be treated as a harmless pull request with 0 line changes. 6

```
++ "b/#{Arbitrary codes here}"
++ b/Casks/cask.rb
```

Preparing for the demonstration

As bug bounty platforms such as HackerOne require PoC/demonstration in the report, I decided to demonstrate this vulnerability.

Since it's not a good idea to modify the casks that are being used without permission, I tried to find a test cask in the `homebrew-cask` repository. However, I couldn't find it. So I contacted the Homebrew staff who operates the vulnerability disclosure program on HackerOne.

After that, I received `I can't add a test cask just for this but you could try to make a harmless modification to an existing cask perhaps?` from the staff. Therefore, I chose a random cask and decided to make harmless changes.

Demonstrating the vulnerability

Since I saw a pull request that inadvertently posted an API Token on GitHub, I decided to make changes to `iterm2.rb` that this pull request was trying to update.

Before adding the modification, I noticed that `++ b/Casks/iterm2.rb` would throw an error if these variables are not defined. So I forked `Homebrew/homebrew-cask` and added the following 2 lines to `Casks/iterm2.rb`.7

```
++ "b/#{puts 'Going to report it - RyotaK (https://hackeorne.com/ryotak)';b = 1;Casks = 1;iterm2 = {};iterm2.define_singl
++ b/Casks/iterm2.rb
```

By defining `b`, `Casks`, `iterm2`, `iterm2.rb` in the first line, the second line won't throw an error. Therefore, it can be executed as a valid Ruby script. Also, by adding these changes, GitHub will return the following diff:

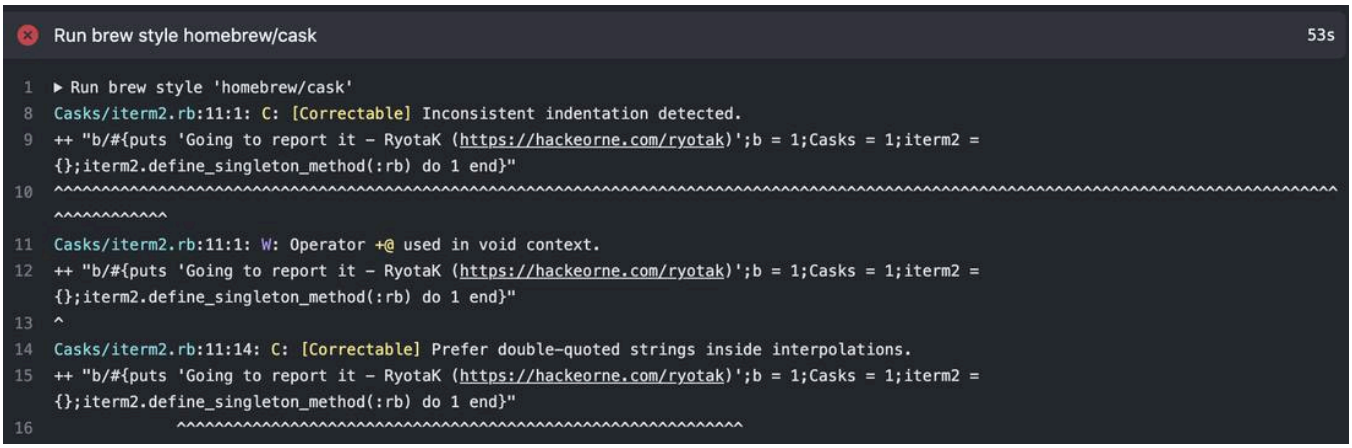
```
diff --git a/Casks/iterm2.rb b/Casks/iterm2.rb
index 3c376126bb1cf9..ba6f4299c1824e 100644
--- a/Casks/iterm2.rb
+++ b/Casks/iterm2.rb
@@ -8,6 +8,8 @@
     sha256 "e7403dcc5b08956a1483b5defea3b75fb81c3de4345da6000e3ad4a6188b47df"
   end

+++ "b/#{puts 'Going to report it - RyotaK (https://hackeorne.com/ryotak)';b = 1;Casks = 1;iterm2 = {};iterm2.define_sing
+++ b/Casks/iterm2.rb
   url "https://iterm2.com/downloads/stable/iTerm2-#{version.dots_to_underscores}.zip"
   name "iTerm2"
   desc "Terminal emulator as alternative to Apple's Terminal app"
```

As mentioned above, `git_diff` treats lines that match `+++ "?b/(.*)` as file path information rather than added lines, so this diff will be treated as a pull request that making a change of 0 lines.

After making this change, I made a pull request8 and started demonstrating the vulnerability.

Problem occurred

Even after waiting for a while, the pull requests were not merged, and when I checked the CI execution log, I noticed `Required status checks for pull request 104191 are not successful.` in the output. By looking into failed checks, I confirmed that a workflow that uses `brew style` was failed, which means Rubocop rejected the changes.



Rubocop allows source codes to disable its feature by adding `# rubocop:disable all` at the end of the line. The first line could be fixed by adding the comment, but the second line couldn't. As the second line must return `Casks/iterm2.rb` in the capturing group of `+++ "?b/(.*)`, it can't be fixed by just adding the comment.

After some tries, it turned out that it was possible to ignore Rubocop without changing the last line by making the following changes:

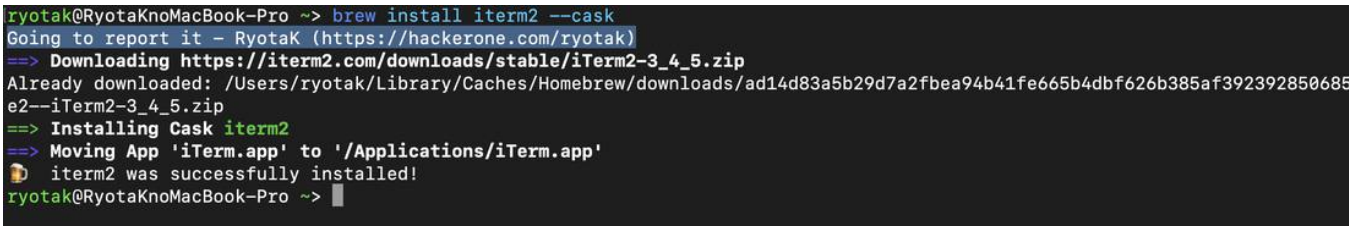
```
++ "b/#{puts 'Going to report it - RyotaK (https://hackerone.com/ryotak)';b = 1;Casks = 1;iterm2 = {};iterm2.define_singl
++ "b/" if # rubocop:disable all
++ b/Casks/iterm2.rb
```

By adding `if` in the second line and letting the next line evaluate as an `if` expression, it was possible to fix the `Operator / used in void context.` warning.

Since all checks on the pull request were successfully run, `BrewTestBot` merged my pull request.

Problem occurred... again

As the pull request merged successfully, I executed `brew install iterm2 --cask` and confirmed that `Going to report it - RyotaK (https://hackerone.com/ryotak)` were printed. Then send an image as a PoC in the report.



After that, while waiting for a reply to the report I sent, I received the following reply on Twitter.

> @ryotkak - do you take credit for my #homebrew behaviour? :) pic.twitter.com/CczRDTemu9
>
> — mrkosmici (@mrkosmici) April 18, 2021

I couldn't understand it for few seconds, but somehow `brew cleanup` prints `Going to report it - RyotaK (https://hackerone.com/ryotak)` too.

When I tried it in my machine in a hurry, I could confirm that `Going to report it - RyotaK (https://hackerone.com/ryotak)` was displayed even when `brew cleanup` was executed.

I didn't notice it because I was in a hurry at this time, but as a result of investigating later, I found that a modified cask was executed if someone executed `brew search` etc. in addition to `brew cleanup`. It was designed to evaluate all casks when some commands were executed, so even if the target cask was not installed, the modified code will be executed.

As the only changes I made were to print additional logs, and maintainers reverted the changes immediately, this didn't have much impact. However, I was very surprised because I didn't expect this to happen.

Conclusion

In this article, I described the vulnerability that was existed in the Homebrew's official tap. If this vulnerability was abused by a malicious actor, it could be used to compromise the machines that run `brew` before it gets reverted. So I strongly feel that a security audit against the centralized ecosystem is required. I want to perform security audits against PyPI/npm registry... etc, but as they don't allow the vulnerability assessment explicitly, I can't do this.

If you have any comments/questions about this article, please send me a message on Twitter(@ryotkak).

Timeline

| Date (JST) | Event | | | April 17, 2021 | Found the vulnerability | | | April 17, 2021 | Sent an email to the maintainer | | | April 18, 2021 | Received a response from the maintainer | | | April 18, 2021 5 pm | Started the demonstration | | | April 18, 2021 5 pm | Sent a report | | | April 18, 2021 6 pm | Successfully merged the pull request | | | April 18, 2021 7 pm | Pull request was reverted | | | April 18, 2021 8 pm | Primary fix completed | | | April 19, 2021 | Secondary fix completed | | | April 21, 2021 | Incident has been disclosed | |

-

Specifically, it's about the same as the Dead by Daylight match wait time before improvement.

-

Also, Homebrew had an incident related to the leakage of GitHub API Token in 2018, so it seems that the awareness of the members was high enough.

-

I'm sorry that the last three articles are all related to GitHub Actions. (But GitHub Actions is a very good attack surface.)

-

https://github.com/Homebrew/actions/blob/bac0cf0eef64950c5fa7b60134da80f5f52d87ab/.github/workflows/review-cask-pr.yml

-

I omitted the conditions that are not important to the vulnerabilities I found, but there were many other conditions. For details, please check https://github.com/Homebrew/actions/blob/bac0cf0eef64950c5fa7b60134da80f5f52d87ab/.github/workflows/review-cask-pr.yml

-

It is intentional that the second line is not a string literal, because of another bug in `git_diff`, it doesn't ignore double quotes when closing a string literal.

-

At this point, I thought that each cask file will be executed only if it was specified in `brew install`. Given the situation in which the maintainer was in contact, I thought that if it got reverted immediately, no user would execute the modified code.

-

https://github.com/Homebrew/homebrew-cask/pull/104191
