"If you could grep the contents of every file on GitHub, what would you do?"

Using BigQuery to find (and fix!) bugs in GitHub projects.

Who even am I?

GitHub Public Dataset!

2.2 trillion files (master@HEAD, in most cases)

Commit information (diffs, hashes, author, committer,)

File *contents* (2.2TB; excludes > 1MB files)

https://cloud.google.com/bigquery/public-data/github

1. Files (repo, file id, path, metadata)
2. Commits (ref, hashes, diff, parent, author, committer)
3. Contents (file id, file contents for all files < 1MiB)
4. Languages (repo, lang. name, bytes)
5. Licenses (repo, license)

Why BigQuery?

```
~  bq query 'SELECT
  dataset_id,
  table_id,
  row_count,
  size_bytes
FROM
  `bigquery-public-data.github_repos.__TABLES__`
ORDER BY
  row_count DESC,
  size_bytes DESC'

+--------------+-----------------+------------+---------------+
|  dataset_id  |    table_id     | row_count  |  size_bytes   |
+--------------+-----------------+------------+---------------+
| github_repos | files           | 2287437607 |  340898815933 |
| github_repos | contents        |  255149733 | 2376496065261 |
| github_repos | commits         |  222868780 |  789922381206 |
| github_repos | sample_files    |   72879442 |   10722663446 |
| github_repos | languages       |    3353696 |     205272194 |
| github_repos | licenses        |    3353696 |     109080210 |
| github_repos | sample_contents |    2905870 |   25812254485 |
| github_repos | sample_commits  |     672309 |    2676919179 |
| github_repos | sample_repos    |     400000 |      13130351 |
+--------------+-----------------+------------+---------------+
```

results := BigQuery GitHub public dataset + codeWeWantToFind + regularExpressions

Why not GitHub's search API?

Repositories
Code 1K
Commits
Issues 15
Marketplace
Topics
Wikis 3
Users

Languages

Go 1,346

Advanced search Cheat sheet

Showing 1,345 available code results

Sort: Best match

venndenu – crypto.go — Go

Showing the top three matches Last indexed on Jul 7

```
22          return hex.EncodeToString(securecookie.GenerateRandomKey(32))
23  }
24
25  func GeneratePaymentId() string {
26          return hex.EncodeToString(securecookie.GenerateRandomKey(16))
27  }
28
29  func GenerateAssetId() string {
30          return hex.EncodeToString(securecookie.GenerateRandomKey(16))
```

PGo-Projects/tasky – gen_keys.go — Go

Showing the top four matches Last indexed 8 days ago

```
10  func main() {
11          key := securecookie.GenerateRandomKey(64)
12          if key == nil {
13                  panic("Key not generated sucessfully")
14          }
15          fmt.Println(base64.StdEncoding.EncodeToString(key))
16          key = securecookie.GenerateRandomKey(64)
```

tryy3/webbforum – random.go — Go

Showing the top five matches Last indexed on Jul 13

```
6           "github.com/gorilla/securecookie"
```

Accidentally, Ephemeral Keys

```go
func main() {
    // The result of our key generation is never stored.
    var store = sessions.NewCookieStore(securecookie.GenerateRandomKey(32))

    // rest of program
    r := mux.NewRouter()
    r.HandleFunc("/", someHandler)
    // etc.
}
```

- Key generation is hard; securely storing keys is harder.
- Original documentation favored convenience
- Users were generating "accidentally ephemeral" keys.

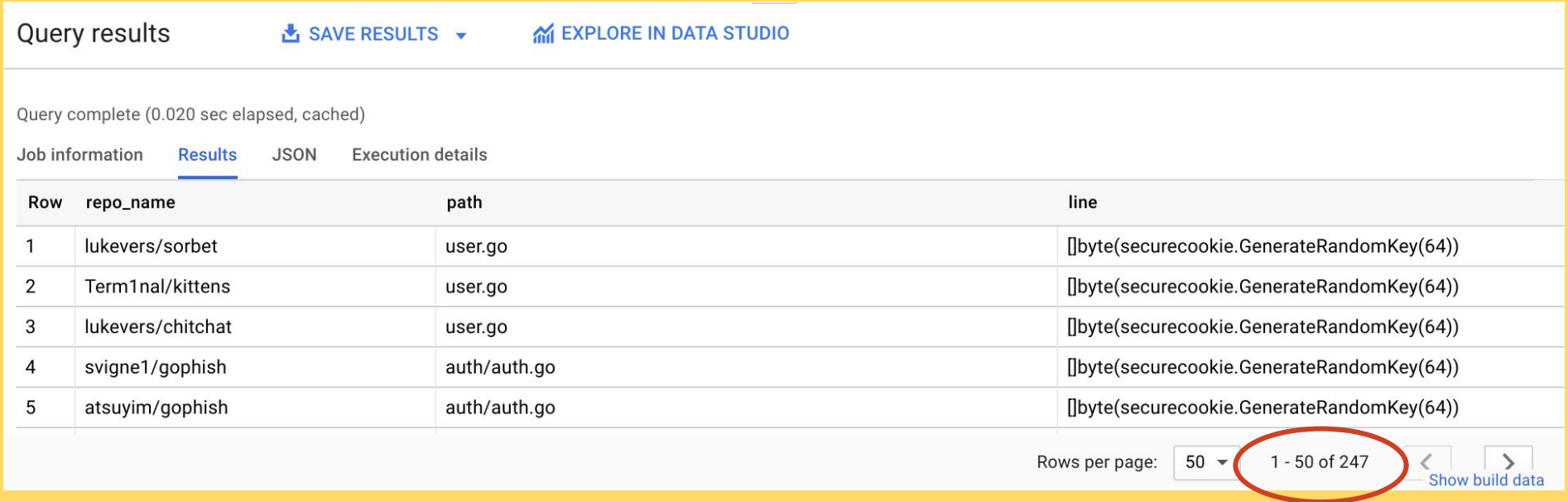
Let's use everybody's favorite tool: regex!

- Capturing cases where keys are generated as a function argument
- Doesn't aim to capture all cases.
- We extract the full line for added context.

```
SELECT
  repo_name,
  path,
  line
FROM (
  SELECT
    id,
    repo_name,
    path
  FROM
    `bigquery-public-data.github_repos.files`
  WHERE
    path LIKE "%.go") AS files
JOIN (
  SELECT
    id,
    REGEXP_REPLACE(
    REGEXP_EXTRACT(content, r".*\(securecookie\.GenerateRandomKey\(\d+\)\)"),
    r"^(?:\t+|\s+)", "") AS line
  FROM
    `bigquery-public-data.github_repos.contents`
  WHERE
    REGEXP_CONTAINS(content, r"\w+\(securecookie\.GenerateRandomKey\(\d+\)\)")
    AND binary = FALSE) AS contents
ON
  files.id = contents.id
GROUP BY
  repo_name,
  path,
  line
```

wait about 35 seconds

- 247 results
- Lots of struct members
- Handful of clones/forks
- LOTS of vendored code (we'll get to this later...)



```
82          // Create new session store
83          store = sessions.NewFilesystemStore(
84                  // Path to sessions
85                  *sessionsPath,
86
87                  // Secret key with strength set to 64
88                  []byte(securecookie.GenerateRandomKey(64)),
89          )
90  }
```

```
25              }
26              cookiestore = sessions.NewCookieStore(securecookie.GenerateRandomKey(32))
27      )
```

What else?

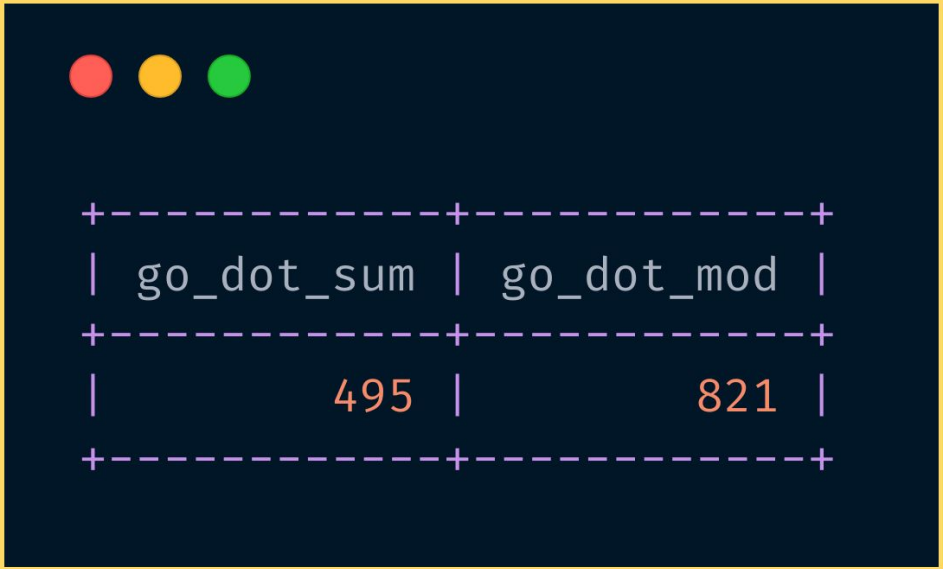
- Number of projects that have adopted Go Modules?
- Using the wrong hash functions (MD5, SHA-family) on passwords?
- Who's pinning my module at an old version? (helping consumers upgrade)

```
~  bq query 'SELECT
  MAX(EXTRACT(date
    FROM
      Timestamp)) AS date,
  ModuleCount,
  FORMAT("%.1f%%", ROUND((ModuleCount / LAG(ModuleCount) OVER (ORDER BY ModuleCount) * 100 - 100), 1)) as WeekOverWeek

FROM
  `golang.module_count`
GROUP BY
  ModuleCount
ORDER BY
  date ASC'

+------------+-------------+--------------+
|    date    | ModuleCount | WeekOverWeek |
+------------+-------------+--------------+
| 2018-08-23 |         262 | NULL         |
| 2018-08-30 |         289 | 10.3%        |
| 2018-09-06 |         417 | 44.3%        |
| 2018-09-13 |         495 | 18.7%        |
| 2018-09-20 |         562 | 13.5%        |
| 2018-09-27 |         623 | 10.9%        |
| 2018-10-04 |         665 | 6.7%         |
| 2018-10-18 |         707 | 6.3%         |
| 2018-10-21 |         821 | 16.1%        |
+------------+-------------+--------------+
```

- 60% adoption.
- Wonder why?
- Assumption: the purpose of go.sum is unclear, despite its usefulness.



- Caveat: modules are still experimental as of Go 1.11
- There are still seemingly a lot of bugs with more complex repositories: repos with existing v2.0+ tags and a non "/v2" import path.
- Future: inspect Go Modules for known-bad pinned versions.

- MD5: not the right way to "hash" passwords.
- SHA-1, SHA-2 only slightly "less worse" here.
- Should use a Key Derivation Function (KDF) that "stretches" input (see: scrypt)
- Mix of developers who either don't know any better, or don't care to understand.

Goal:

- Find code that calls (e.g.) md5.Sum or sha.Sum*
- Based on variable naming semantics, determine mis-use
- Help educate those maintainers (and protect their users)

```
WHERE
    REGEXP_CONTAINS(content, r'(?i:.*(?:md5|sha)(?:256|512)?\.Sum(?:256|512|512_224|512_256|512_384)?\
((?:pass|passwd|password|pw)\).*)')
    AND binary = FALSE) AS contents
```

- Not perfect, but a good indicator
- Could "fork" this idea and look at misuse of cryptographic hashes as psuedo-HMACs
- … or even too-short keys & params for the right constructs, like bcrypt.

We've found a bunch of bugs. OK, good.

But how do we help them?


~ issue-maker --from-file=ephemeral_keys_20181021.csv --template=keys.tmpl --issue-title="API mis-use with key generation"
ts=2018-10-21T12:34:56Z msg="dry-run is ENABLED. Pass --no-dry-run to create issues."
ts=2018-10-21T12:34:56Z msg="parsed 247 repos from 'ephemeral_keys_20181021.csv'"
ts=2018-10-21T12:34:58Z msg="template 'keys.tmpl' OK"
ts=2018-10-21T12:35:00Z msg="GitHub token is valid"
ts=2018-10-21T12:34:56Z msg="creating 247 issues"
ts=2018-10-21T12:34:56Z msg="completed"

Pinned versions in go.mod?

```
  SELECT
    id,
    REGEXP_REPLACE( REGEXP_EXTRACT(content, r".+gorilla\/mux"), r"^(?:\t+|\s+)", "") AS repo,
    REGEXP_REPLACE( REGEXP_EXTRACT(content, r"(?:.+gorilla\/mux).*(v.+)"), r"^(?:\t+|\s+)", "") AS
version
```

Query results SAVE RESULTS EXPLORE IN DATA STUDIO

Query complete (16.319 sec elapsed, 2.41 TB processed)

Job information | Results | JSON | Execution details

Row	repo_name	path	repo	version
1	micromdm/scep	go.mod	github.com/gorilla/mux	v1.4.0
2	euskadi31/go-server	go.mod	github.com/gorilla/mux	v1.5.0
3	moul/protoc-gen-gotemplate	go.mod	github.com/gorilla/mux	v1.5.0
4	cespare/pastedown	go.mod	"github.com/gorilla/mux	v1.6.1
5	fern4lvarez/piladb	go.mod	"github.com/gorilla/mux	v1.6.1
6	bketelsen/captainhook	go.mod	require "github.com/gorilla/mux	v1.6.1
7	pingcap/pd	go.mod	github.com/gorilla/mux	v1.6.1
8	keratin/authn-server	go.mod	github.com/gorilla/mux	v1.6.1
9	google/go-cloud	go.mod	github.com/gorilla/mux	v1.6.1
10	bcspragu/Radiotation	go.mod	github.com/gorilla/mux	v1.6.2
11	unprofession-al/bpmon	go.mod	github.com/gorilla/mux	v1.6.2
12	micromdm/micromdm	go.mod	github.com/gorilla/mux	v1.6.2
13	gernest/utron	go.mod	github.com/gorilla/mux	v1.6.2

Rows per page: 50 1 - 44 of 44

- Better understand usage of your APIs
- Help pinned (and/or vendored!) repos upgrade
- Not just public GitHub repos: consider mirroring your company's source code into tools like BigQuery.

Thanks!

Contact:

@elithrar on Twitter (taking suggestions for things to find + can help you craft queries)

https://goo.gl/2DMszH for a Gist with my queries as a Jupyter Notebook