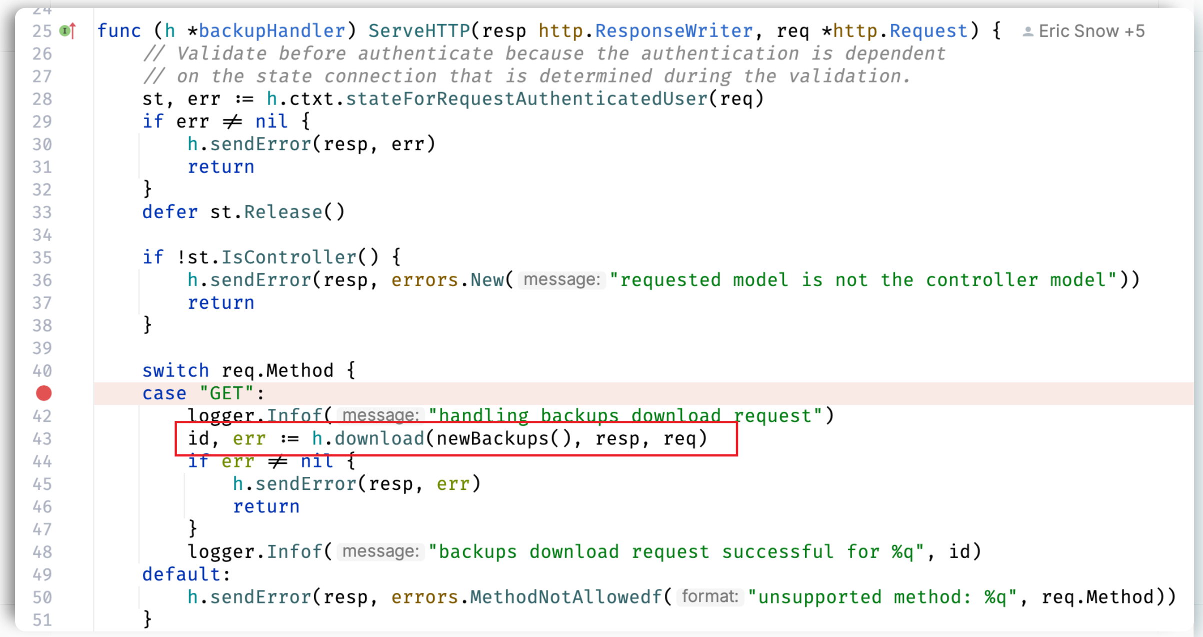
✔ Arbitrary file reading vulnerability

When I learned codeql, I found that there was a potential loophole in juju. When the authentication passed, I downloaded the backup file. I could control the id value to any file location, and then download the file through download().

The version affected by the vulnerability is the latest version 3.0 and below(juju⩽latest version)

✔ Causes of loopholes

apiserver/backup.go



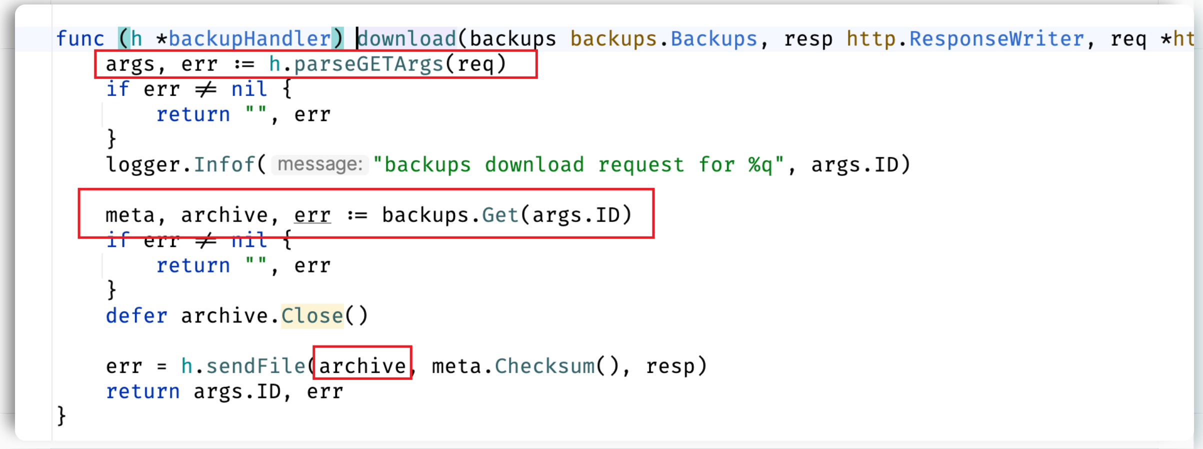
```go
func (h *backupHandler) ServeHTTP(resp http.ResponseWriter, req *http.Request) {
    // Validate before authenticate because the authentication is dependent
    // on the state connection that is determined during the validation.
    st, err := h.ctxt.stateForRequestAuthenticatedUser(req)
    if err ≠ nil {
        h.sendError(resp, err)
        return
    }
    defer st.Release()

    if !st.IsController() {
        h.sendError(resp, errors.New( message: "requested model is not the controller model"))
        return
    }

    switch req.Method {
    case "GET":
        logger.Infof( message: "handling backups download request")
        id, err := h.download(newBackups(), resp, req)
        if err ≠ nil {
            h.sendError(resp, err)
            return
        }
        logger.Infof( message: "backups download request successful for %q", id)
    default:
        h.sendError(resp, errors.MethodNotAllowedf( format: "unsupported method: %q", req.Method))
    }
```

When the authentication is passed, I can control the data in the request package.

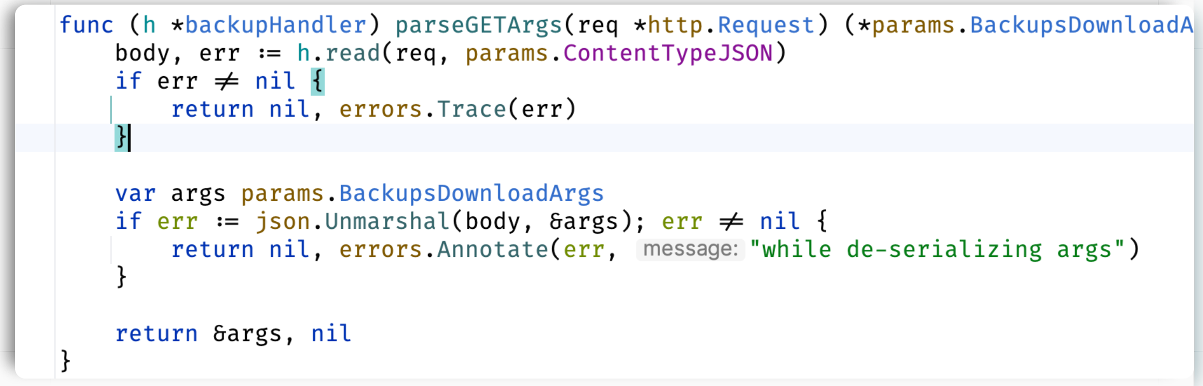
```
{
"id":"/etc/passwd"
}
```



```
func (h *backupHandler) download(backups backups.Backups, resp http.ResponseWriter, req *ht
    args, err := h.parseGETArgs(req)
    if err ≠ nil {
        return "", err
    }
    logger.Infof( message: "backups download request for %q", args.ID)

    meta, archive, err := backups.Get(args.ID)
    if err ≠ nil {
        return "", err
    }
    defer archive.Close()

    err = h.sendFile(archive, meta.Checksum(), resp)
    return args.ID, err
}
```



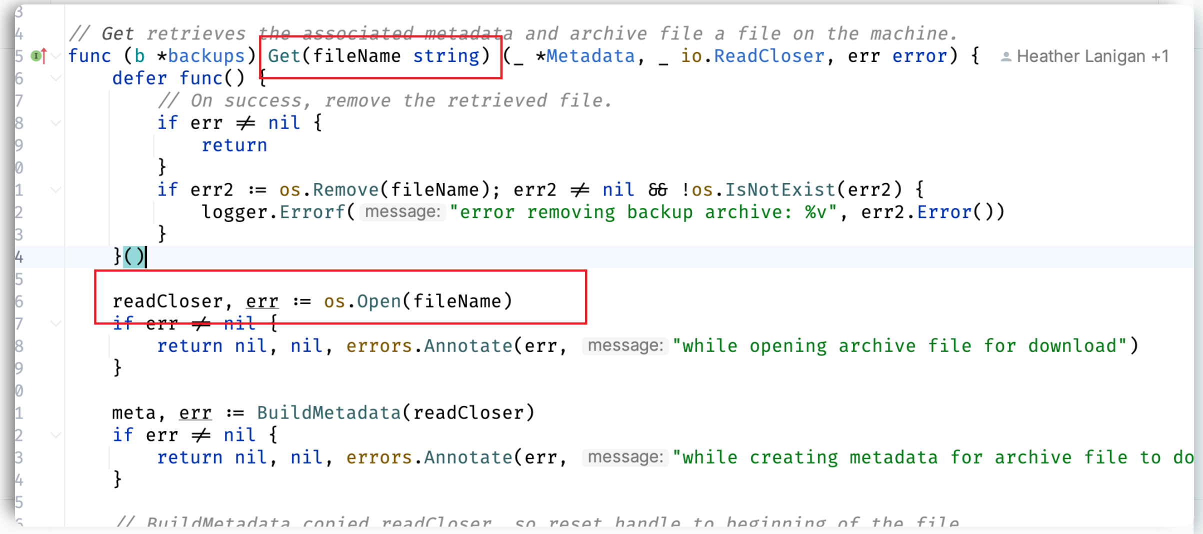
```
func (h *backupHandler) parseGETArgs(req *http.Request) (*params.BackupsDownloadA
    body, err := h.read(req, params.ContentTypeJSON)
    if err ≠ nil {
        return nil, errors.Trace(err)
    }

    var args params.BackupsDownloadArgs
    if err := json.Unmarshal(body, &args); err ≠ nil {
        return nil, errors.Annotate(err,  message: "while de-serializing args")
    }

    return &args, nil
}
```

parseGETArgs() The passed-in parameter values are parsed directly, and the id is not judged.

Causes the contents of the file to be obtained directly using the os.open () function



```
// Get retrieves the associated metadata and archive file a file on the machine.
func (b *backups) Get(fileName string) (_ *Metadata, _ io.ReadCloser, err error) {   Heather Lanigan +1
    defer func() {
        // On success, remove the retrieved file.
        if err ≠ nil {
            return
        }
        if err2 := os.Remove(fileName); err2 ≠ nil && !os.IsNotExist(err2) {
            logger.Errorf( message: "error removing backup archive: %v", err2.Error())
        }
    }()

    readCloser, err := os.Open(fileName)
    if err ≠ nil {
        return nil, nil, errors.Annotate(err,  message: "while opening archive file for download")
    }

    meta, err := BuildMetadata(readCloser)
    if err ≠ nil {
        return nil, nil, errors.Annotate(err,  message: "while creating metadata for archive file to do
    }

    // BuildMetadata copied readCloser, so reset handle to beginning of the file.
```

This creates a security hazard.

Backup downloads should be limited to a specific directory, rather than being modified at will.