

# Traverse the Path

Attacks on extraction implementation

# Who am I

Jan Harrie

Security Engineer @HashiCorp by day 🌞  
and soon-to-be-dad by night 😊

Hobbies:

- Cooking
- Outdoor activities
- Dog



# Motivation

I started working on a secure extraction library in Golang to secure our supply chain by offering a best practices implementation for such a well studied problem class.

To get product teams motivated to adopt the library, I started reviewing our code base and the results were ... unexpected

# Motivation

Funny enough, Joern Schneeweisz posted after my submission the following on LinkedIn:



Which nails the topic of this talk ;)

# Recap: Well known archive attacks

- Exhaustion attacks, e.g., [42.zip](#)
- Path traversal attacks, e.g., archive entries with leading `../`
- Symlinks to sensitive files, e.g., links to `passwd`  $\Rightarrow$  `/etc/passwd` that are read after extraction
- Zip-Slip attacks, with smart packed archives, example will follow

Sounds not too tough huh?

# Exhaustion Attack Prevention

Implement limits for:

- Input size
- Output size
- Maximum files #
- Extraction-Recursion

Establish extraction timeouts



# Path Traversal Attack Prevention

Ensure that the joined path starts with the destination  
**AND** a path separator!

Archive structure

```
file  
sub/  
sub/../../../../escaped
```

Combined with path /tmp/dst

```
/tmp/dst/file  
/tmp/dst/sub/  
/tmp/escaped
```

That's really a thing? What?

# Bug: (Limited) Path Traversal Attacks

Adjusted code example from [hashicorp/go-slug](https://github.com/hashicorp/go-slug)

```
func NewUnpackInfo(dst string, header *tar.Header) (UnpackInfo, error) {
    path := header.Name
    if path[0] == '/' {
        path = path[1:]
    }
    path = filepath.Join(dst, path)
    target := filepath.Clean(path)
    if !strings.HasPrefix(target, dst) {
        return UnpackInfo{}, errors.New("invalid filename, traversal with \"..\" outside of current directory")
    }
}
```

Let's start looking for bugs



# Bug: (Limited) Path Traversal Attacks

Adjusted code example from [hashicorp/go-slug](https://github.com/hashicorp/go-slug)

```
func NewUnpackInfo(dst string, header *tar.Header) (UnpackInfo, error) {  
    path := header.Name  
    if path[0] == '/' {  
        path = path[1:]  
    }  
    path = filepath.Join(dst, path)  
    target := filepath.Clean(path)  
    if !strings.HasPrefix(target, dst) {  
        return UnpackInfo{}, errors.New("invalid filename, traversal with \"..\" outside of current directory")  
    }  
}
```

Remove the prefix

# Bug: (Limited) Path Traversal Attacks

Adjusted code example from [hashicorp/go-slug](https://github.com/hashicorp/go-slug)

```
func NewUnpackInfo(dst string, header *tar.Header) (UnpackInfo, error) {  
    path := header.Name  
    if path[0] == '/' {  
        path = path[1:]  
    }  
    path = filepath.Join(dst, path)  
    target := filepath.Clean(path)  
    if !strings.HasPrefix(target, dst) {  
        return UnpackInfo{}, errors.New("invalid filename, traversal with \"..\" outside of current directory")  
    }  
}
```

Combine the path's

# Bug: (Limited) Path Traversal Attacks

Adjusted code example from [hashicorp/go-slug](https://github.com/hashicorp/go-slug)

```
func NewUnpackInfo(dst string, header *tar.Header) (UnpackInfo, error) {
    path := header.Name
    if path[0] == '/' {
        path = path[1:]
    }
    path = filepath.Join(dst, path)
    target := filepath.Clean(path)
    if !strings.HasPrefix(target, dst) {
        return UnpackInfo{}, errors.New("invalid filename, traversal with \"..\" outside of current directory")
    }
}
```

Check the prefix

# Bug: (Limited) Path Traversal Attacks

Adjusted code example from [hashicorp/go-slug](https://github.com/hashicorp/go-slug)

```
func NewUnpackInfo(dst string, header *tar.Header) (UnpackInfo, error) {
    path := header.Name
    if path[0] == '/' {
        path = path[1:]
    }
    path = filepath.Join(dst, path)
    target := filepath.Clean(path)
    if !strings.HasPrefix(target, dst) {
        return UnpackInfo{}, errors.New("invalid filename, traversal with \"..\" outside of current directory")
    }
}
```

Can you spot the bug?

# Bug: (Limited) Path Traversal Attacks

Adjusted code example from [hashicorp/go-slug](https://github.com/hashicorp/go-slug)

```
func NewUnpackInfo(dst string, header *tar.Header) (UnpackInfo, error) {
    path := header.Name
    if path[0] == '/' {
        path = path[1:]
    }
    path = filepath.Join(dst, path)
    target := filepath.Clean(path)
    if !strings.HasPrefix(target, dst) {
        return UnpackInfo{}, errors.New("invalid filename, traversal with \"..\" outside of current directory")
    }
}
```

Missing separator at the end of dst

Nothing big, but still not as intended

```
dst := "/tmp/dst"
path := "../dst2/escaped"
target := filepath.Join(dst, path) // ==> /tmp/dst2/escaped
```

# Attack Detection: Zip-Slip Attack

Don't traverse symlinks during extraction and check every element!

Archive entries

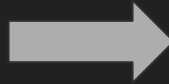
```
sub/  
sub/root -> ../  
sub/root/outside -> ../  
sub/root/outside/escaped
```

# Attack Detection: Zip-Slip Attack

Don't traverse symlinks during extraction and check every element!

Archive entries

```
sub/  
sub/root -> ../  
sub/root/outside -> ../  
sub/root/outside/escaped
```



Extracted files

```
sub/  
sub/root  
outside  
../escaped
```

Sounds not too tough huh?

# Existing implementation

```
currentPath := dst // Start at the root of the unpacked tarball.
```

```
components := strings.Split(header.Name, "/")
```

Split the path

)' line of code." data-bbox="398 275 535 315"/>

```
for i := 0; i < len(components)-1; i++ {
    currentPath = filepath.Join(currentPath, components[i])
    fi, err := os.Lstat(currentPath)
    if os.IsNotExist(err) {
        // Parent directory structure is incomplete. Technically this
        // means from here upward cannot be a symlink, so we cancel the
        // remaining path tests.
        break
    }
    if err != nil {
        return UnpackInfo{}, fmt.Errorf("failed to evaluate path %q: %w", header.Name, err)
    }
    if fi.Mode()&fs.ModeSymlink != 0 {
        return UnpackInfo{}, fmt.Errorf("cannot extract %q through symlink", header.Name)
    }
}
```



# Existing implementation

```
currentPath := dst // Start at the root of the unpacked tarball.  
components := strings.Split(header.Name, "/")
```

```
for i := 0; i < len(components)-1; i++ {  
    currentPath = filepath.Join(currentPath, components[i])  
    fi, err := os.Lstat(currentPath)  
    if os.IsNotExist(err) {  
        // Parent directory structure is incomplete. Technically this  
        // means from here upward cannot be a symlink, so we cancel the  
        // remaining path tests.  
        break  
    }  
    if err != nil {  
        return UnpackInfo{}, fmt.Errorf("failed to evaluate path %q: %w", header.Name, err)  
    }  
    if fi.Mode()&fs.ModeSymlink != 0 {  
        return UnpackInfo{}, fmt.Errorf("cannot extract %q through symlink", header.Name)  
    }  
}
```

Iterate over the  
elements




# Existing implementation

```
currentPath := dst // Start at the root of the unpacked tarball.
components := strings.Split(header.Name, "/")

for i := 0; i < len(components)-1; i++ {
    currentPath = filepath.Join(currentPath, components[i])
    fi, err := os.Lstat(currentPath)
    if os.IsNotExist(err) {
        // Parent directory structure is incomplete. Technically this
        // means from here upward cannot be a symlink, so we cancel the
        // remaining path tests.
        break
    }
    if err != nil {
        return UnpackInfo{}, fmt.Errorf("failed to evaluate path %q: %w", header.Name, err)
    }
    if fi.Mode()&fs.ModeSymlink != 0 {
        return UnpackInfo{}, fmt.Errorf("cannot extract %q through symlink", header.Name)
    }
}
```

Get the type of  
each element

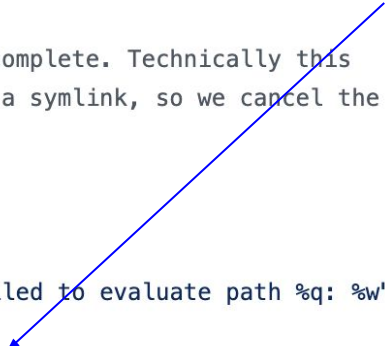


# Existing implementation

```
currentPath := dst // Start at the root of the unpacked tarball.
components := strings.Split(header.Name, "/")

for i := 0; i < len(components)-1; i++ {
    currentPath = filepath.Join(currentPath, components[i])
    fi, err := os.Lstat(currentPath)
    if os.IsNotExist(err) {
        // Parent directory structure is incomplete. Technically this
        // means from here upward cannot be a symlink, so we cancel the
        // remaining path tests.
        break
    }
    if err != nil {
        return UnpackInfo{}, fmt.Errorf("failed to evaluate path %q: %w", header.Name, err)
    }
    if fi.Mode()&fs.ModeSymlink != 0 {
        return UnpackInfo{}, fmt.Errorf("cannot extract %q through symlink", header.Name)
    }
}
```

If an element in  
the path is a  
symlink ⇒ fail!



# Bug: Zip-Slip Attack

Can YOU spot the bug?

```
currentPath := dst // Start at the root of the unpacked tarball.
components := strings.Split(header.Name, "/")

for i := 0; i < len(components)-1; i++ {
    currentPath = filepath.Join(currentPath, components[i])
    fi, err := os.Lstat(currentPath)
    if os.IsNotExist(err) {
        // Parent directory structure is incomplete. Technically this
        // means from here upward cannot be a symlink, so we cancel the
        // remaining path tests.
        break
    }
    if err != nil {
        return UnpackInfo{}, fmt.Errorf("failed to evaluate path %q: %w", header.Name, err)
    }
    if fi.Mode()&fs.ModeSymlink != 0 {
        return UnpackInfo{}, fmt.Errorf("cannot extract %q through symlink", header.Name)
    }
}
```


```
header.Name := "sub/does-not-exist/../../root/outside/escaped
```

# Bug: Zip-Slip Attack

```
currentPath := dst // Start at the root of the unpacked tarball.
components := strings.Split(header.Name, "/")

for i := 0; i < len(components)-1; i++ {
    currentPath = filepath.Join(currentPath, components[i])
    fi, err := os.Lstat(currentPath)
    if os.IsNotExist(err) {
        // Parent directory structure is incomplete. Technically this
        // means from here upward cannot be a symlink, so we cancel the
        // remaining path tests.
        break
    }
    if err != nil {
        return UnpackInfo{}, fmt.Errorf("failed to evaluate path %q: %w", header.Name, err)
    }
    if fi.Mode()&fs.ModeSymlink != 0 {
        return UnpackInfo{}, fmt.Errorf("cannot extract %q through symlink", header.Name)
    }
}
```

Code optimization  
invented a security issue



Bug got assigned [CVE-2025-0377](#) and addressed in [hashicorp/go-slug#76](#)

# Invariant Zip-Slip Attack: Link Write Attack

Did you know that one archive can contain multiple entries with the same name? It depend on the implementation how such edge-cases are handled.

Archive entries

```
link -> file  
link
```

# Invariant Zip-Slip Attack: Link Write Attack

Did you know that one archive can contain multiple entries with the same name? It depend on the implementation how such edge-cases are handled.

Archive entries

```
link -> file  
link
```

The tar binary extracts every entry and overwrites existing one

```
/tmp% tar -xvf example.tar  
x link  
x link
```

# Invariant Zip-Slip Attack: Link Write Attack

Did you know that one archive can contain multiple entries with the same name? It depend on the implementation how such edge-cases are handled.

Archive entries

```
link -> file  
link
```

The tar binary extracts every entry and overwrites existing one

```
/tmp% tar -xvf example.tar  
x link  
x link  
/tmp% cat link  
hi :wave:
```



# Invariant Zip-Slip Attack: Link Write Attack

Did you know that one archive can contain multiple entries with the same name? It depend on the implementation how such edge-cases are handled.

Archive entries

```
link -> file  
link
```

The tar binary extracts every entry and overwrites existing one

```
/tmp% tar -xvf example.tar  
x link  
x link  
/tmp% cat link  
hi :wave:  
/tmp% cat file  
cat: file: No such file or directory
```

But what is the default behavior in programming languages?

# Excursion: Godocs

## func OpenFile

```
func OpenFile(name string, flag int, perm FileMode) (*File, error)
```

OpenFile is the generalized open call; most users will use `Open` or `Create` instead. It opens the named file with specified flag (`O_RDONLY` etc.). If the file does not exist, and the `O_CREATE` flag is passed, it is created with mode `perm` (before `umask`). If successful, methods on the returned `File` can be used for I/O. If there is an error, it will be of type `*PathError`.

But what about symlinks to files that do not exist?

# Excursion: Golang

Turns out: The `Open` syscall is used in golang under the hood [\[ref\]](#) which traverses symlinks.

```
func open(path string, flag int, perm uint32) (int, poll.SysFile, error) {  
    fd, err := syscall.Open(path, flag, perm)  
    return fd, poll.SysFile{}, err  
}
```

Let's use that and have some fun

(I observed the same behaviour in Python. Further details can be found [here](#))

# Bug: Link-Write Attack Example

This [Nomad source code](#) looked perfect to be exploited

```
// If the header is for a symlink we create the symlink
if hdr.Typeflag == tar.TypeSymlink {
    if err = os.Symlink(hdr.Linkname, filepath.Join(dest, hdr.Name)) err != nil {
        return fmt.Errorf("error creating symlink: %w", err)
    }

    for _, path := range []string{hdr.Name, hdr.Linkname} {
        if escapes, err := escapingfs.PathEscapesAllocDir(dest, "", path); err != nil {
            return fmt.Errorf("error evaluating symlink: %w", err)
        } else if escapes {
            return fmt.Errorf("archive contains symlink that escapes alloc dir")
        }
    }

    continue
}

// If the header is a file, we write to a file
if hdr.Typeflag == tar.TypeReg {
    f, err := os.Create(filepath.Join(dest, hdr.Name))
```

# Bug: Link-Write Attack Example

But the security check was stopping me in the first place.

```
if escapes, err := escapingfs.PathEscapesAllocDir(dest, "", hdr.Name); err != nil {  
    return fmt.Errorf("error evaluating object: %w", err)  
} else if escapes {  
    return fmt.Errorf("archive contains object that escapes alloc dir")  
}
```

# Bug: Link-Write Attack Example

You can guess – based on the function body – which error I encountered to bypass the security check ;)

```
// Check path does not escape the alloc dir using symlinks.
if escapes, err := pathEscapesBaseViaSymlink(base, full); err != nil {
    if os.IsNotExist(err) {
        // Treat non-existent files as non-errors; perhaps not ideal but we
        // have existing features (log-follow) that depend on this. Still safe,
        // because we do the symlink check on every ReadAt call also.
        return false, nil
    }
    return false, err
} else if escapes {
    return true, nil
}
```

# Bug: Link-Write Attack Example

You can guess – based on the function body – which error I encountered to bypass the security check ;)



```
// Check path does not escape the alloc dir using symlinks.
if escapes, err := pathEscapesBaseViaSymlink(base, full), err != nil {
    if os.IsNotExist(err) {
        // Treat non-existent files as non-errors; perhaps not ideal but we
        // have existing features (log-follow) that depend on this. Still safe,
        // because we do the symlink check on every ReadAt call also.
        return false, nil
    }
    return false, err
} else if escapes {
    return true, nil
}
```

```
func pathEscapesBaseViaSymlink(base, full string) (bool, error) {
    resolveSym, err := filepath.EvalSymlinks(full)
    if err != nil {
        return false, err
    }
}
```

`filepath.EvalSymlink(path)`  
returns `os.ErrNotExist` if `path` does  
not exist.

# Bug: Link-Write Attack Example

The security check could be bypassed as long as the symlink in the archive points to a target that does not exist before extraction.



# Bug: Link-Write Attack Example

The security check could be bypassed as long as the symlink in the archive points to a target that does not exist before extraction.

The bug got [CVE-2024-7625](#) assigned and was remediated by aligning to the behaviour of the tar binary and deleting existing files before extraction.

```
if hdr.Typeflag == tar.TypeReg {
    f, err := os.Create(filepath.Join(dest, hdr.Name))
    fPath := filepath.Join(dest, hdr.Name)
    if _, err := os.Lstat(fPath); err == nil {
        if err := os.Remove(fPath); err != nil {
            return fmt.Errorf("error removing existing file: %w", err)
        }
    }
    f, err := os.Create(fPath)
```

Further details can be found in [HSEC-2024-17](#).

# Another day, another bug

Google released at the beginning of the year the blogpost [The Family of Safe Golang Libraries is Growing!](#).

Due to the fact that I was also working on a library – the [google/safearchive](#) library grabbed my attention and I was curious how they handle Symlinks in archives.

# google/safearchive source code

Since we are security experts, we can verify that the implementation is secure, right? ;)

```
if tr.securityMode&PreventSymlinkTraversal != 0 {
    hName := sanitizer.SanitizePath(h.Name)
    hName = strings.TrimSuffix(hName, "/")
    n := strings.Split(hName, "/")
    traversal := false
    for i := 1; i <= len(n); i++ {
        subPath := strings.Join(n[0:i], "/")
        if tr.symlinks[subPath] {
            // a symlink has already been seen on this path. We need to drop this entry.
            traversal = true
            break
        }
    }
    if traversal {
        continue
    }
    if h.Linkname != "" {
        tr.symlinks[hName] = true
    }
}
```

# google/safearchive source code

Since we are security experts, we can verify that the implementation is secure, right? ;)

```
if tr.securityMode&PreventSymlinkTraversal != 0 {  
    hName := sanitizer.SanitizePath(h.Name)  
    hName = strings.TrimSuffix(hName, "/")  
    n := strings.Split(hName, "/")  
    traversal := false  
    for i := 1; i <= len(n); i++ {  
        subPath := strings.Join(n[0:i], "/")  
        if tr.symlinks[subPath] {  
            // a symlink has already been seen on this path. We need to drop this entry.  
            traversal = true  
            break  
        }  
    }  
    if traversal {  
        continue  
    }  
    if h.Linkname != "" {  
        tr.symlinks[hName] = true  
    }  
}
```

Sanitize path

# google/safearchive source code

Since we are security experts, we can verify that the implementation is secure, right? ;)

```
if tr.securityMode&PreventSymlinkTraversal != 0 {
    hName := sanitizer.SanitizePath(h.Name)
    hName = strings.TrimSuffix(hName, "/")
    n := strings.Split(hName, "/")
    traversal := false
    for i := 1; i <= len(n); i++ {
        subPath := strings.Join(n[0:i], "/")
        if tr.symlinks[subPath] {
            // a symlink has already been seen on this path. We need to drop this entry.
            traversal = true
            break
        }
    }
    if traversal {
        continue
    }
    if h.Linkname != "" {
        tr.symlinks[hName] = true
    }
}
```

Remove absolut path prefix

# google/safearchive source code

Since we are security experts, we can verify that the implementation is secure, right? ;)

```
if tr.securityMode&PreventSymlinkTraversal != 0 {
    hName := sanitizer.SanitizePath(h.Name)
    hName = strings.TrimSuffix(hName, "/")
    n := strings.Split(hName, "/")
    traversal := false
    for i := 1; i <= len(n); i++ {
        subPath := strings.Join(n[0:i], "/")
        if tr.symlinks[subPath] {
            // a symlink has already been seen on this path. We need to drop this entry.
            traversal = true
            break
        }
    }
    if traversal {
        continue
    }
    if h.Linkname != "" {
        tr.symlinks[hName] = true
    }
}
```

check for symlinks from previous archive entries

# google/safearchive source code

Since we are security experts, we can verify that the implementation is secure, right? ;)

```
if tr.securityMode&PreventSymlinkTraversal != 0 {
    hName := sanitizer.SanitizePath(h.Name)
    hName = strings.TrimSuffix(hName, "/")
    n := strings.Split(hName, "/")
    traversal := false
    for i := 1; i <= len(n); i++ {
        subPath := strings.Join(n[0:i], "/")
        if tr.symlinks[subPath] {
            // a symlink has already been seen on this path. We need to drop this entry.
            traversal = true
            break
        }
    }
    if traversal {
        continue
    }
    if h.Linkname != "" {
        tr.symlinks[hName] = true
    }
}
```

If an entry is a symlink,  
keep book of it.

# google/safearchive source code

```
if tr.securityMode&PreventSymlinkTraversal != 0 {
    hName := sanitizer.SanitizePath(h.Name)
    hName = strings.TrimSuffix(hName, "/")
    n := strings.Split(hName, "/")
    traversal := false
    for i := 1; i < len(n); i++ {
        subP := strings.Join(n[:i], "/")
        if t := tr.Traverse(subP); t == nil {
            continue
        }
        if t.IsDir() {
            continue
        }
        if t.IsSymlink() {
            continue
        }
        if traversal {
            continue
        }
        if h.Linkname != "" {
            tr.symlinks[hName] = true
        }
    }
}
```

to drop this entry.

Feels secure right?



# google/safearchive source code

```
if tr.securityMode&PreventSymlinkTraversal != 0 {  
    hName := sanitizer.SanitizePath(h.Name)  
    hName = strings.TrimSuffix(hName, "/")  
    n := st  
    travers  
    for i :
```

It depends on the underlying  
filesystem ;)

rop this entry.

```
    }  
    if trav  
  
    }  
    if h.Linkname != "" {  
        tr.symlinks[hName] = true  
    }  
}
```

# google/safearchive source code

NTFS (Windows) & APFS (Mac) are case insensitive filesystems by default

```
etc -> /etc  
Etc/passwd
```

Exploit the  
bug

# google/safearchive source code

NTFS (Windows) & APFS (Mac) are case insensitive filesystems by default

```
etc -> /etc      Exploit the  
Etc/passwd      bug
```


The bug got [CVE-2024-10389](#) assigned and remediated in the next patch cycle.

```
hName = strings.TrimSuffix(hName, "/")  
if tr.securityMode&PreventCaseInsensitiveSymlinkTraversal != 0 {  
    hName = strings.ToLower(hName)  
}  
  
n := strings.Split(hName, "/")
```

# Related bugs and open research

Short file names on Windows (**DOWNL0~1 == Downloads**) [[ref](#)]

[WorstFit: Unveiling Hidden Transformers in Windows ANSI!](#) w/Orange Tsai

Windows filepath handling is a complete own rabbit hole 

# Summary & Recommendation

Investigating a problem area while implementing a safe library is great way to learn!

Big names does not mean no problems – no shit :)

Low-level system interactions need to be implemented with care.

Stick with best practices implementation, e.g., [google/safearchive](#) or [hashicorp/go-extract](#)

# Questions?

Thank you for your attention!

slides: [s.gurke.io/dc18](https://s.gurke.io/dc18)  
contact: [jan@nody.cc](mailto:jan@nody.cc)