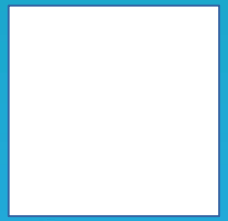
Go AMS Meetup April ‘24

What is new in go?



Arno Overgaauw

Github @ArnoSen

me@km42.nl

Timeline 2024

- Feb 6: release of Go 1.22.0
- Feb 27: last AMS meetup with a presentation by Micklei on his favorites of Go 22
- March 5:
- Go 1.21.8 (11 issues patched of which 1 CVE)
- Go 1.22.1 (19 issues of which 1 CVE)
- April 4:
- Go 1.21.9 (4 issues of which 3 CVE’s)
- Go 1.22.2 (12 of which 3 CVE’s)

Do upgrade!

- 1.21.9 / 1.22.2 has a fix for CVE-2023-45288: resources can be exhausted when a remote actor sends large compressed headers
- a max header limit was enforced but only after the headers were parsed (and the compute spend on the decompression)

How was it fixed?

```
for {
»       frag := hc.HeaderBlockFragment()
»       if _, err := hdec.Write(frag); err != nil {
»       »       return nil, ConnectionError(ErrCodeCompression)
»       }
```

```
1579 »       for {
1580 »       »       frag := hc.HeaderBlockFragment()
1581
1582 »       »       // Avoid parsing large amounts of headers that we will then discard.
1583 »       »       // If the sender exceeds the max header list size by too much,
1584 »       »       // skip parsing the fragment and close the connection.
1585 »       »       //
1586 »       »       // "Too much" is either any CONTINUATION frame after we've already
1587 »       »       // exceeded the max header list size (in which case remainSize is 0),
1588 »       »       // or a frame whose encoded size is more than twice the remaining
1589 »       »       // header list bytes we're willing to accept.
1590 »       »       if int64(len(frag)) > int64(2*remainSize) {
1591 »       »       »       if VerboseLogs {
1592 »       »       »       »       log.Printf("http2: header list too large")
1593 »       »       »       }
1594 »       »       »       // It would be nice to send a RST_STREAM before sending the GOAWAY,
1595 »       »       »       // but the struture of the server's frame writer makes this difficult.
1596 »       »       »       return nil, ConnectionError(ErrCodeProtocol)
1597 »       »       }
1598
1599 »       »       // Also close the connection after any CONTINUATION frame following an
1600 »       »       // invalid header, since we stop tracking the size of the headers after
1601 »       »       // an invalid one.
1602 »       »       if invalid != nil {
1603 »       »       »       if VerboseLogs {
1604 »       »       »       »       log.Printf("http2: invalid header: %v", invalid)
1605 »       »       »       }
1606 »       »       »       // It would be nice to send a RST_STREAM before sending the GOAWAY,
1607 »       »       »       // but the struture of the server's frame writer makes this difficult.
1608 »       »       »       return nil, ConnectionError(ErrCodeProtocol)
1609 »       »       }
1610
1611 »       »       if _, err := hdec.Write(frag); err != nil {
1612 »       »       »       return nil, ConnectionError(ErrCodeCompression)
1613 »       »       }
```

Do upgrade

Credits to Bartek Nowotarski

About 35% of websites support HTTP/2

There was an issue open for 1.20 as well. However the window was missed.

Since April 26, 2022 the Go project is a CVE Numbering Authority (CNA).

nowotarski.info

HTTP/2 CONTINUATION Flood: Security Advisory

– Vulnerabilities

– 2024

- **HTTP/2 CONTINUATION Flood** A class of vulnerabilities I disco multiple HTTP/2 implementations:
  - amphp/http (CVE-2024-2653),
  - Apache HTTP Server (httpd) (CVE-2024-27316),
  - Apache Tomcat (CVE-2024-24549),
  - Apache Traffic Server (CVE-2024-31309),
  - Envoy proxy (CVE-2024-27919, CVE-2024-30255),
  - Golang (CVE-2023-45288),
  - nghttp2 (CVE-2024-28182),
  - Node.js (CVE-2024-27983),
  - Tempesta FW (CVE-2024-2758) and more.

CVEs in 1.21.9 / 1.22.1

- net/http, net/http/cookiejar: incorrect forwarding of sensitive headers and cookies on HTTP redirect [CVE-2023-45289]
https://github.com/golang/go/issues/65065

- net/http: memory exhaustion in Request.ParseMultipartForm [CVE-2023-45290]
https://github.com/golang/go/issues/65383

- crypto/x509: Verify panics on certificates with an unknown public key algorithm [CVE-2024-24783]
https://github.com/golang/go/issues/65390

CVE-2023-45289 incorrect forwarding of sensitive headers and cookies

- The rule is that sensitive headers and hookies are forwarded when going from foo.com to subdomain.foo.com (or vice versa) but not to bar.com

- the error resulting in the CVE was in determining if X is a subdomain of Y and it was failing for:

http://[::1%25.foo.com]/ is a subdomain of http://foo.com/ : TRUE

- ::1 is the IPv6 localhost address but it can be any (public) IPv6 address
- %foo.com is a (valid) zone identifier (see RFC 68749 Representing IPv6 Zone identifiers)

CVE-2023-45289 incorrect forwarding of sensitive headers and cookies

```
// ...ch domains must already be in canonical form.
func isDomainOrSubdomain(sub, parent string) bool {
	if sub == parent {
		return true
	}
	// If sub contains a :, it's probably an IPv6 address (and is definitely not a hostname).
	// Don't check the suffix in this case, to avoid matching the contents of a IPv6 zone.
	// For example, "::1%.www.example.com" is not a subdomain of "www.example.com".
	if strings.ContainsAny(sub, ":%") {
		return false
	}
	// If sub is "foo.example.com" and parent is "example.com",
	// that means sub must end in "."+parent.
	// Do it without allocating.
	if !strings.HasSuffix(sub, parent) {
		return false
	}
	return sub[len(sub)-len(parent)-1] == '.'
}

func stripPassword(u *url.URL) string {
```

es

+10

```
		// Only len(suffix) is used to determine the jar key from
		// here on, so it is okay if psl.PublicSuffix("www.buggy.psl")
		// returns "com" as the jar key is generated from host.
	}
	prevDot := strings.LastIndex(host[:i-1], ".")
	return host[prevDot+1:]
}

// isIP reports whether host is an IP address.
func isIP(host string) bool {
	if strings.ContainsAny(host, ":%") {
		// Probable IPv6 address.
		// Hostnames can't contain : or %, so this is definitely not a valid host.
		// Treating it as an IP is the more conservative option, and avoids the risk
		// of interpeting ::1%.www.example.com as a subtomain of www.example.com.
		return true
	}
	return net.ParseIP(host) != nil
}
```

CVE-2023-45289 incorrect forwarding of sensitive headers and cookies

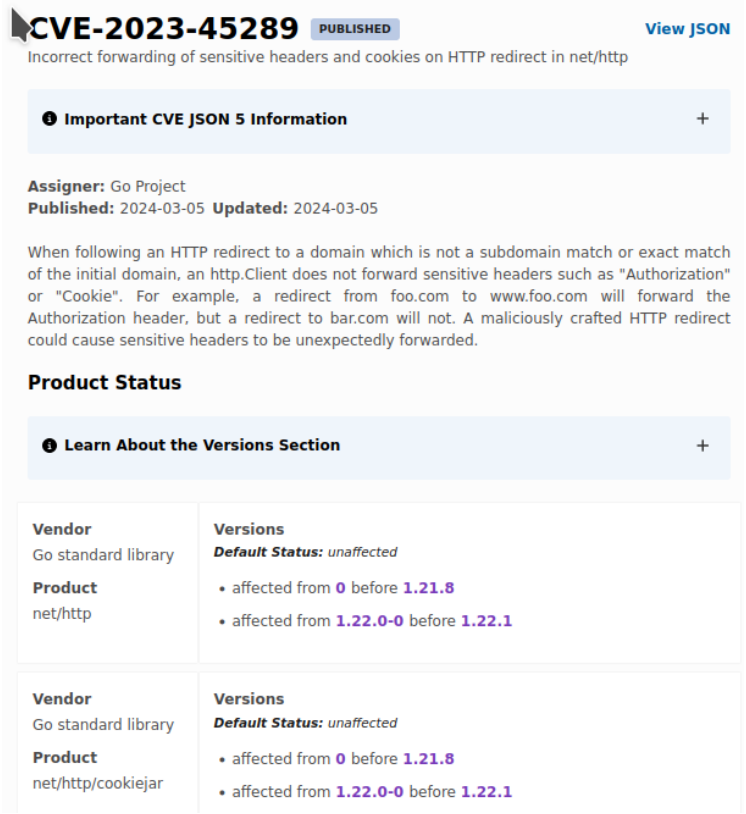
- https://mattermost.com/blog/patching-gos-leaky-http-clients/ goes into the details

- the issue was introduced after CL424935 (merged JAN 2023) which allowed forwarding between domains using different ports. E.g. allow bar.com:443 -> sub.bar.com:8443 allowed. In this change, ipv6 addresses were not squarebracketed anymore which lead to the CVE.

Before foo.com -> [::1%25foo.com] was rejected when performing matching suffixes.

- When reading the CVE, it said all Go version are susceptible to this vulnerability

CVE-2023-45289 incorrect forwarding of sensitive headers and cookies



CVE-2023-45289 PUBLISHED View JSON

Incorrect forwarding of sensitive headers and cookies on HTTP redirect in net/http

Important CVE JSON 5 Information +

Assigner: Go Project
Published: 2024-03-05 **Updated:** 2024-03-05

When following an HTTP redirect to a domain which is not a subdomain match or exact match of the initial domain, an http.Client does not forward sensitive headers such as "Authorization" or "Cookie". For example, a redirect from foo.com to www.foo.com will forward the Authorization header, but a redirect to bar.com will not. A maliciously crafted HTTP redirect could cause sensitive headers to be unexpectedly forwarded.

Product Status

Learn About the Versions Section +

Vendor	Versions
Go standard library	***Default Status:*** *unaffected*
Product	• affected from **0** before **1.21.8**
net/http	• affected from **1.22.0-0** before **1.22.1**

Vendor	Versions
Go standard library	***Default Status:*** *unaffected*
Product	• affected from **0** before **1.21.8**
net/http/cookiejar	• affected from **1.22.0-0** before **1.22.1**

https://www.cve.org/CVERecord?id=CVE-2023-45289

CVE-2023-45289 incorrect forwarding of sensitive headers and cookies

When asked about this, the Go team stated:
“In the general case, it is difficult to determine exactly when a particular vulnerability is introduced, so we err on the side of caution and mark all versions before the fix as vulnerable.

In this particular case, as the article you linked mentions, part of the vulnerability affects versions before https://go-review.googlesource.com/c/go/+/424935 was committed. In addition, that commit was part of Go releases that are old enough to no longer be in our support window.

In cases where the introduced version is known to us, and is recent enough to be within the support window, we would include that in the versions list.” (https://github.com/golang/go/issues/66696)

CVE-2023-45290: memory exhaustion in Request.ParseMultipartForm

“When parsing a multipart form (either explicitly with Request.ParseMultipartForm or implicitly with Request.FormValue, Request.PostFormValue, or Request.FormFile), **limits on the total size of the parsed form were not applied to the memory consumed while reading a single form line.** This permitted a maliciously crafted input containing very long lines to cause allocation of arbitrarily large amounts of memory, potentially leading to memory exhaustion.”

(Reported as well by Bartek Nowotarski)

CVE-2023-45290: memory exhaustion in Request.ParseMultipartForm

```
func (r *Reader) ReadLineBytes() ([]byte, error) {
	line, err := r.readLineSlice()
	if line != nil {
		line = bytes.Clone(line)
	}
	return line, err
}

func (r *Reader) readLineSlice() ([]byte, error) {
	r.closeDot()
	var line []byte
	for {
		l, more, err := r.R.ReadLine()
		if err != nil {
			return nil, err
		}
		// Avoid the copy if the first call produced a full line.
		if line == nil && !more {
			return l, nil
		}
		line = append(line, l...)
		if !more {
			break
		}
	}
	return line, nil
```

```
func (r *Reader) ReadLineBytes() ([]byte, error) {
	line, err := r.readLineSlice(-1)
	if line != nil {
		line = bytes.Clone(line)
	}
	return line, err
}

// readLineSlice reads a single line from r,
// up to lim bytes long (or unlimited if lim is less than 0),
// eliding the final \r or \r\n from the returned string.
func (r *Reader) readLineSlice(lim int64) ([]byte, error) {
	r.closeDot()
	var line []byte
	for {
		l, more, err := r.R.ReadLine()
		if err != nil {
			return nil, err
		}
		if lim >= 0 && int64(len(line))+int64(len(l)) > lim {
			return nil, errMessageTooLarge
		}
		// Avoid the copy if the first call produced a full line.
		if line == nil && !more {
			return l, nil
		}
		line = append(line, l...)
		if !more {
			break
		}
	}
	return line, nil
```

CVE-2024-24783: crypto/x509: panics in verify with unknown pub key algo

```
892 // for failed checks due to different intermediates having the same Subject.
893 const maxChainSignatureChecks = 100
894
895 func (c *Certificate) buildChains(currentChain []*Certificate, sigChecks *int, opts *VerifyOptions)
    (chains [][]*Certificate, err error) {
896 »       var (
897 »       »       hintErr  error
898 »       »       hintCert *Certificate
899 »       )
900
901 »       considerCandidate := func(certType int, candidate potentialParent) {
902 »       »       if alreadyInChain(candidate.cert, currentChain) {
903 »       »       »       return
904 »       »       }
905
906 »       »       if sigChecks == nil {
907 »       »       »       sigChecks = new(int)
908 »       »       }
909 »       »       *sigChecks++
910 »       »       if *sigChecks > maxChainSignatureChecks {
911 »       »       »       err = errors.New("x509: signature check attempts limit reached while verifyi
    ng certificate chain")
912 »       »       »       return
```

```
892 // for failed checks due to different intermediates having the same Subject.
893 const maxChainSignatureChecks = 100
894
895 func (c *Certificate) buildChains(currentChain []*Certificate, sigChecks *int, opts *VerifyOptions)
    (chains [][]*Certificate, err error) {
896 »       var (
897 »       »       hintErr  error
898 »       »       hintCert *Certificate
899 »       )
900
901 »       considerCandidate := func(certType int, candidate potentialParent) {
902 »       »       if candidate.cert.PublicKey == nil || alreadyInChain(candidate.cert, currentChain) {
903 »       »       »       return
904 »       »       }
905
906 »       »       if sigChecks == nil {
907 »       »       »       sigChecks = new(int)
908 »       »       }
909 »       »       *sigChecks++
910 »       »       if *sigChecks > maxChainSignatureChecks {
911 »       »       »       err = errors.New("x509: signature check attempts limit reached while verifyi
    ng certificate chain")
912 »       »       »       return
```

My conclusions after reviewing the CVE's

- the devil is in the detail
- the developers of Go are also just human
- stick to the standard library when possible to prevent running into edge cases (e.g. who thinks of the use of zone identifiers in IPv6 addresses?)
- be aware that some object(properties) can be nil when outside of the 'happy flow'
- think again how to check for limits: do it before you do any work or commit resources and be precise

Stay informed of new Go releases

- subscribe to the #announcements Slack channel
- check go.dev regularly
- create a (free) account on gopherwatch.org and subscribe for changes in github.com/golang/go
- listen to the Cup-A-Go podcast
- read in to bugs being reported and proposals made a the is golang issue tracker (https://github.com/golang/go/issues)
- subscribe to the weekly go newsletter at golangweekly.com

If you found a security bug, report it to security@golang.org. (full procedure at go.dev/doc/security/policy#reporting-a-security-bug)

What may be in Go 1.23

- iterators: new iter package with which you can use your own objects in a range clause
- slices.Repeat
- ?