tailscale

don’t trust - verify!

how I found a decade-old bug in gorilla/csrf

Patrick O’Doherty

A brief introduction to me

A decade+ of experience building and securing products.

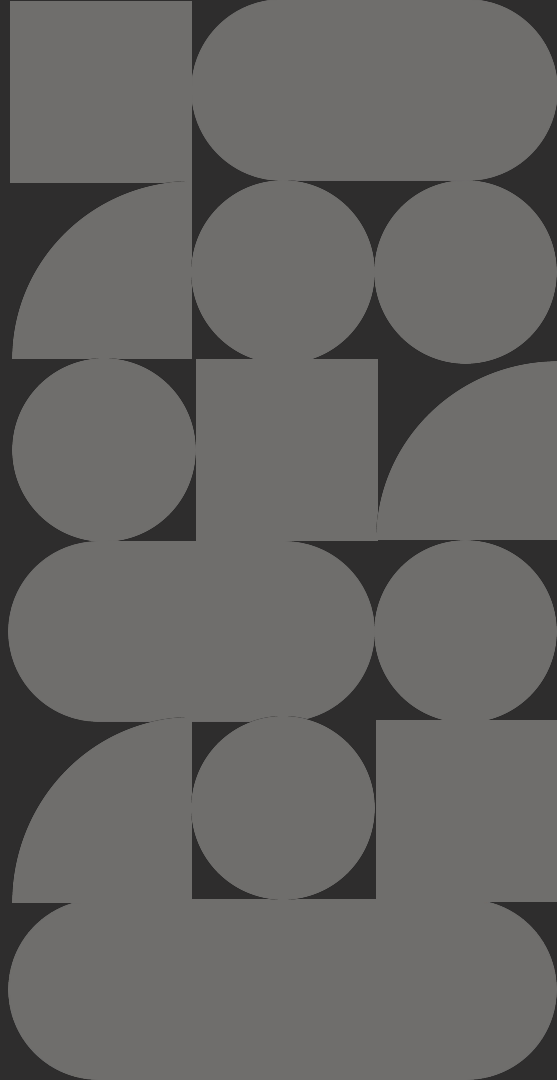
Currently a Member of Technical Staff on the Tailscale security team.

Often found behind a camera viewfinder when away from the keyboard.

A brief introduction to gorilla/csrf

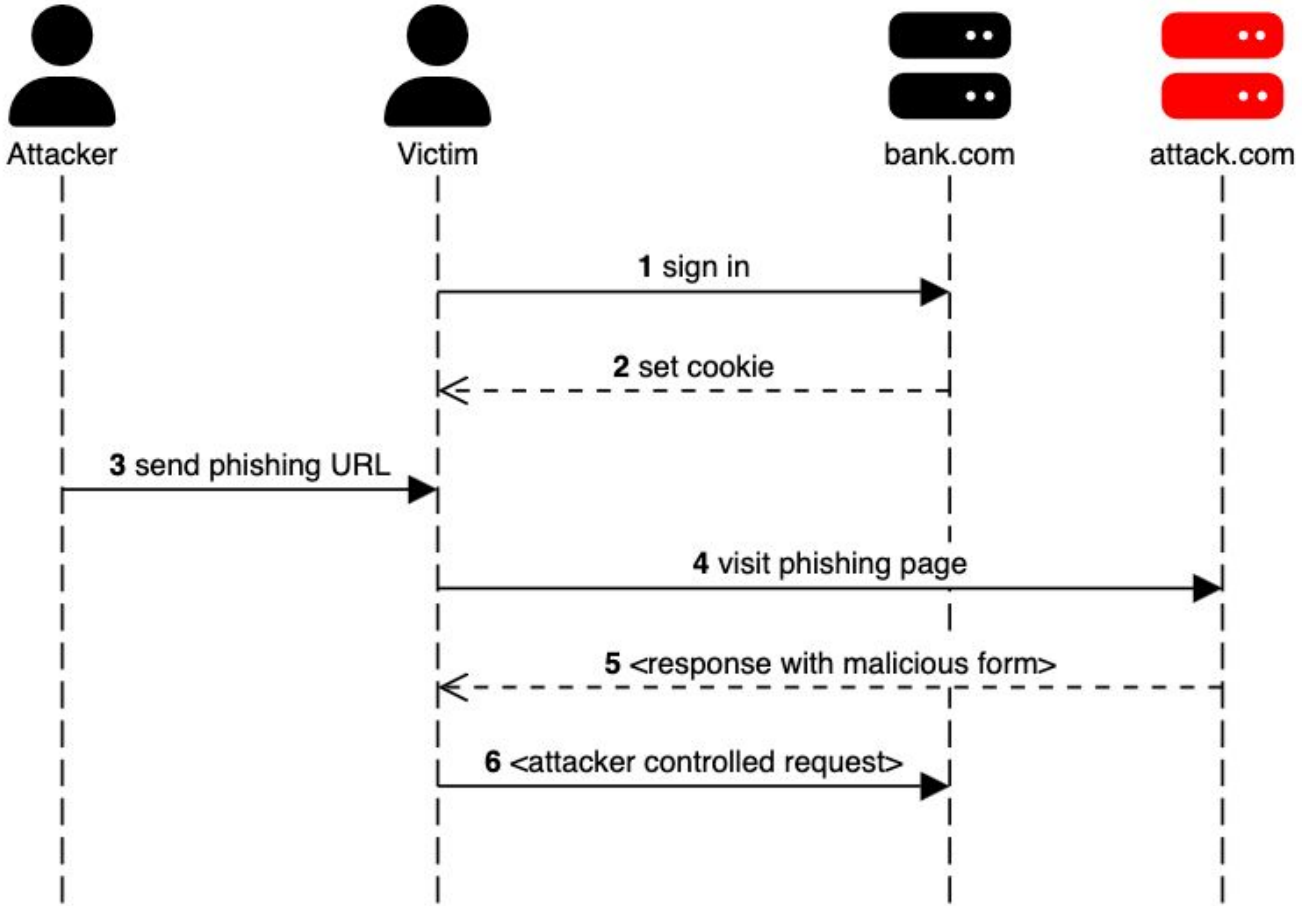
```
// create gorilla/csrf middleware instance
CSRF := csrf.Protect([]byte("$CSRF_SECRET_VALUE"))

// wrap our application HTTP handler
withCSRF := CSRF(handler)


http.ListenAndServe(":8000", withCSRF)
```

What is CSRF exactly?



Cross-**S**ite **R**equest **F**orgery

CSRF Attack Flow



Form submission vs XMLHTTPRequest

	Forms	XMLHTTPRequest
Respects CORS and SOP	⚠️	✅
Sends Cookies by default	⚠️	✅

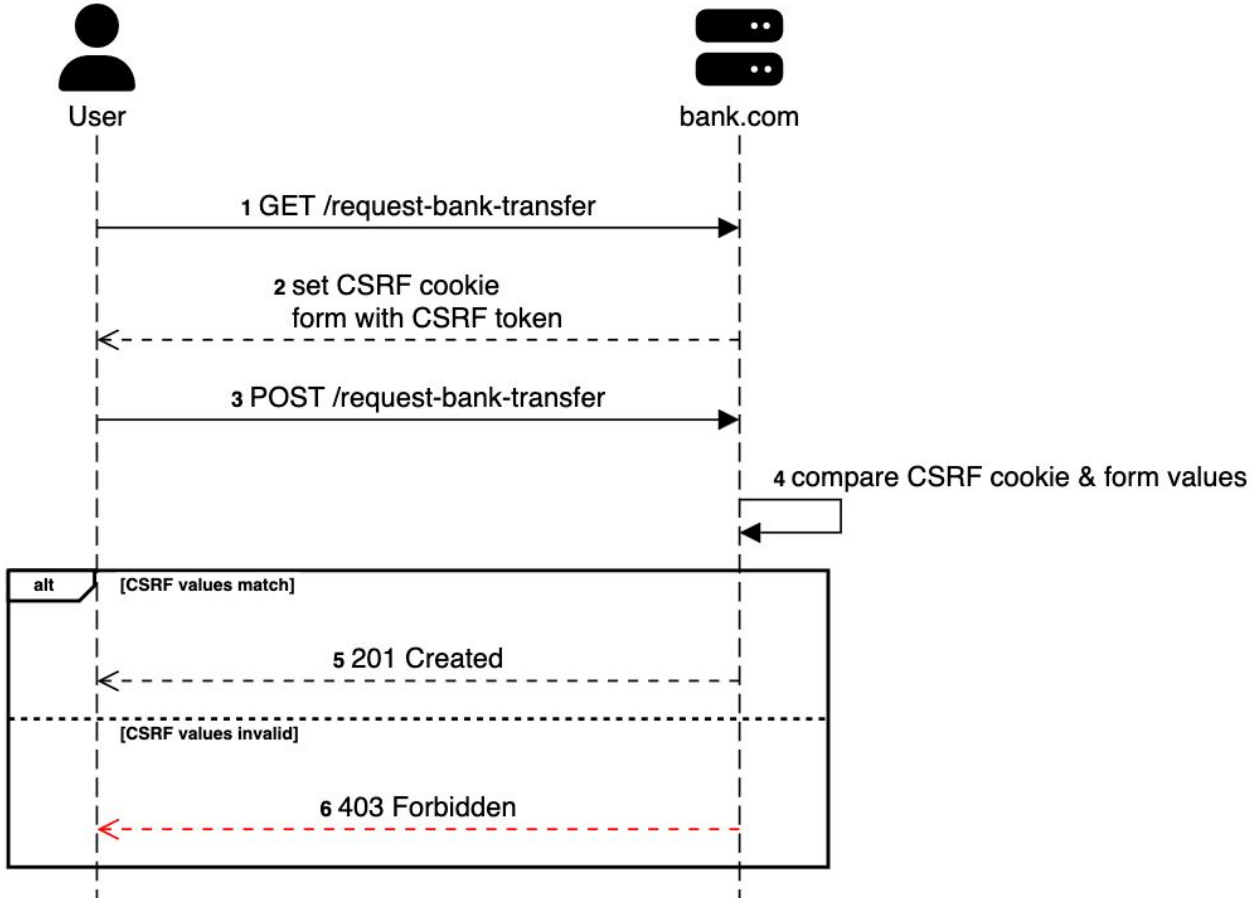
SOP (Same-Origin Policy): restricts how scripts loaded from one origin can interact with resources from another

CORS (**Cross-Origin Resource Sharing)**: server HTTP headers to permit cross-origin HTTP requests

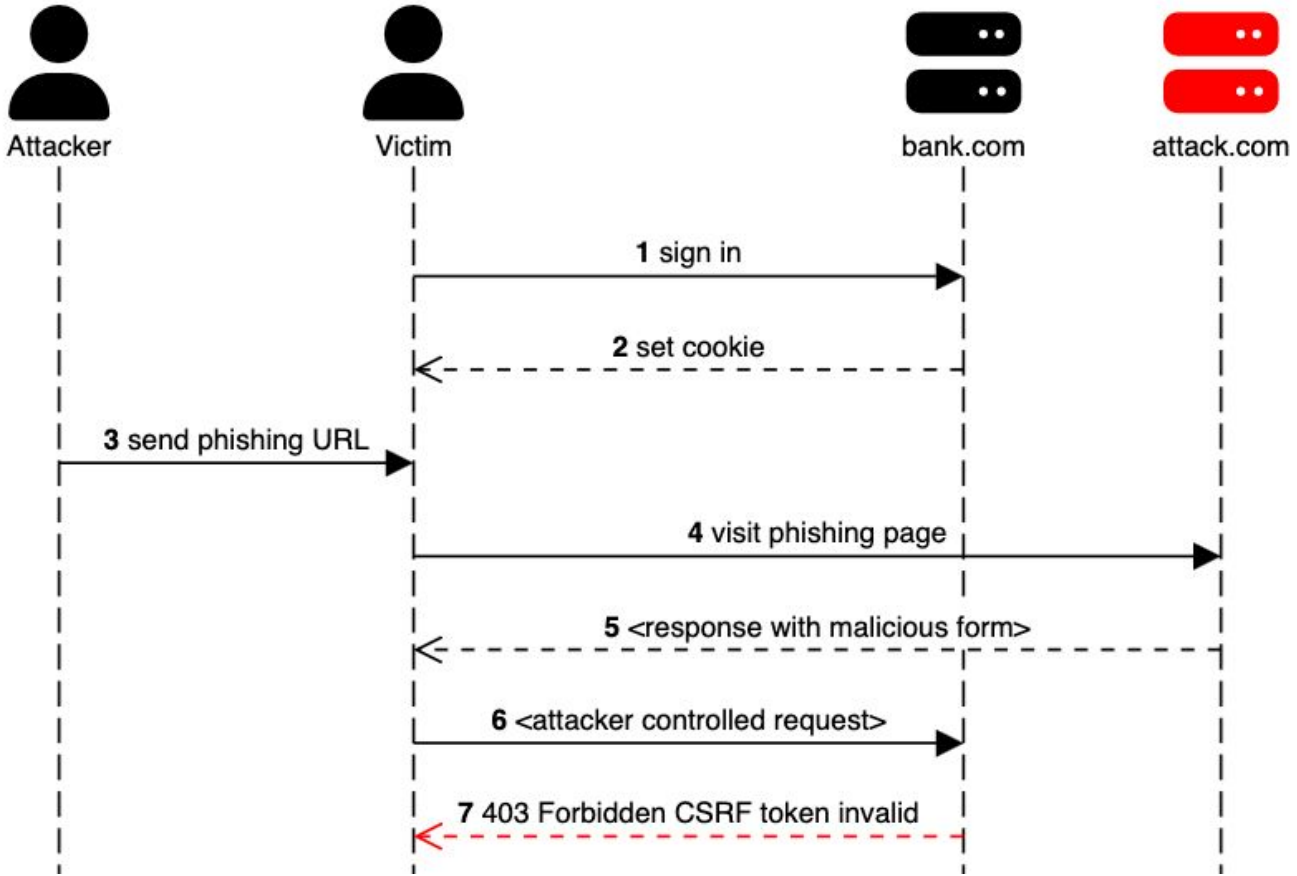
Origin: combination of scheme, host, and optional port e.g. `https://www.example.com:443`

Site: combination of scheme and **root domain** i.e. `https://~~www.~~example.com`

How “double-submit” CSRF tokens work



CSRF Attack Flow

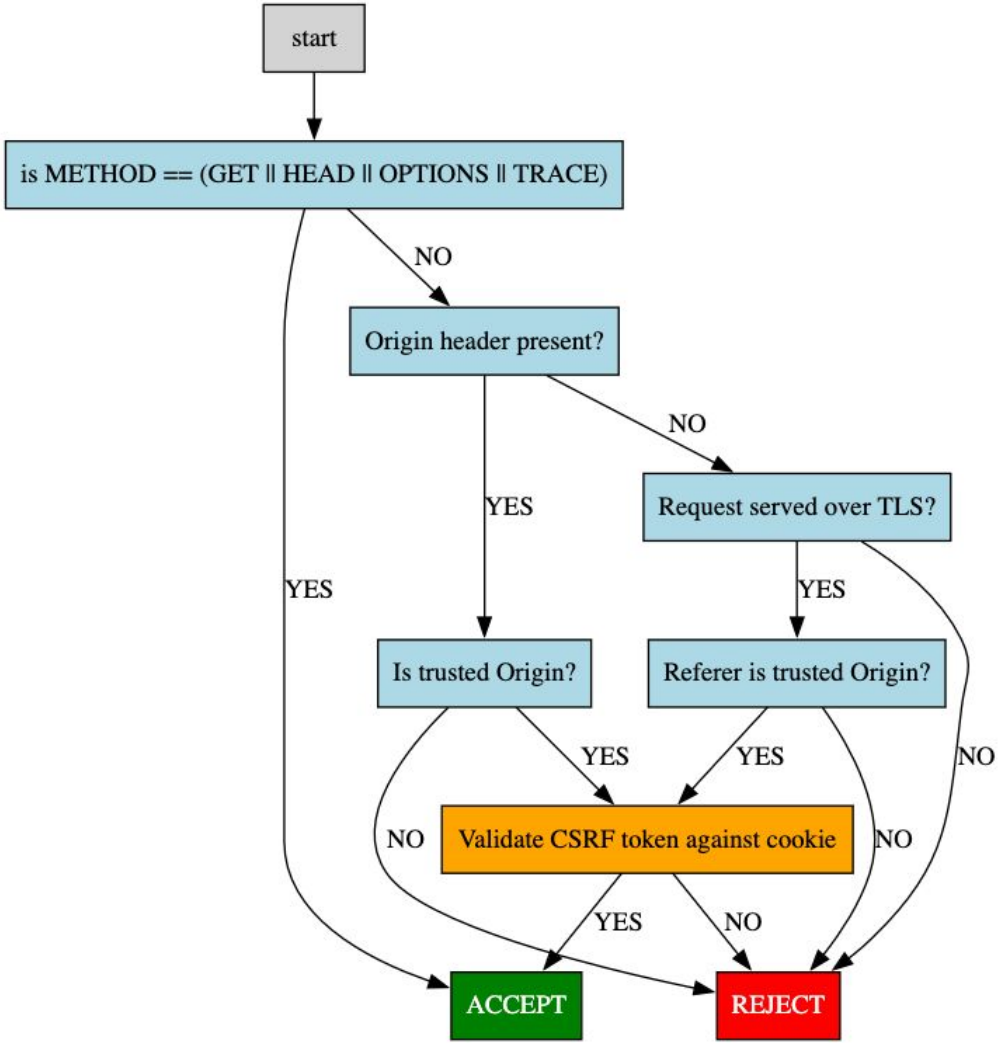


CSRF tokens are not enough

Need to consider same-site attacks

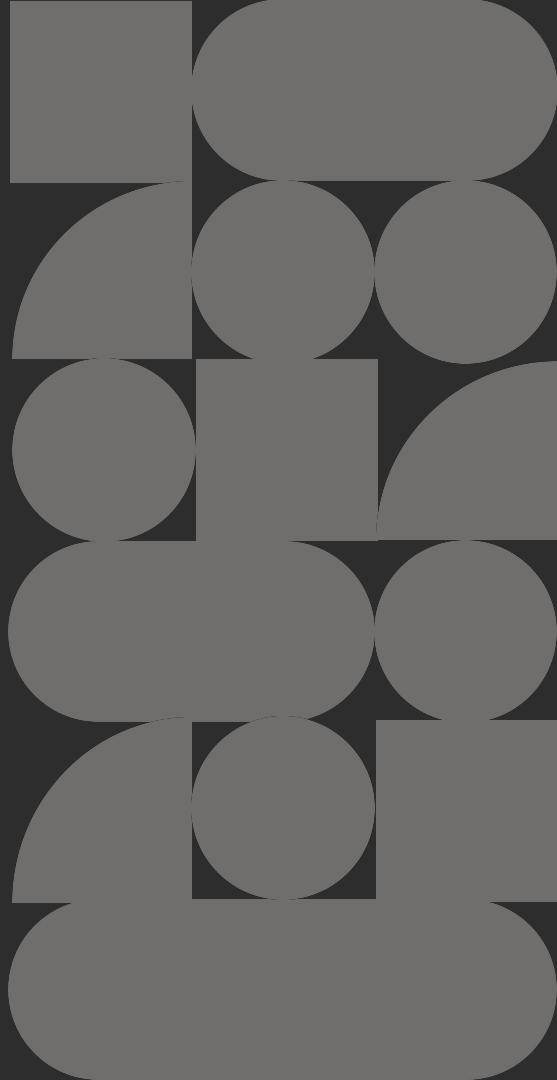
⚠ Machine-in-the-Middle HTTP against bank.com?
An attacker who performs a MiTM attack can set a cookie for secure.bank.com to a value of their choosing.

⚠ XSS on domain that shares the same site?
An attacker who compromises e.g. **events.bank.com** can set cookies on the common **banks.com** domain.


start
is METHOD == (GET || HEAD || OPTIONS || TRACE)
NO
YES
Origin header present?
NO
YES
Request served over TLS?
YES
NO
Is trusted Origin?
Referer is trusted Origin?
YES
YES
NO
NO
Validate CSRF token against cookie
NO
YES
NO
ACCEPT
REJECT

The discovery:

It starts with an email



“What would happen if an attacker gained XSS on tailscale.com?”

“Could they use this to pivot and perform a CSRF attack against login.tailscale.com?“

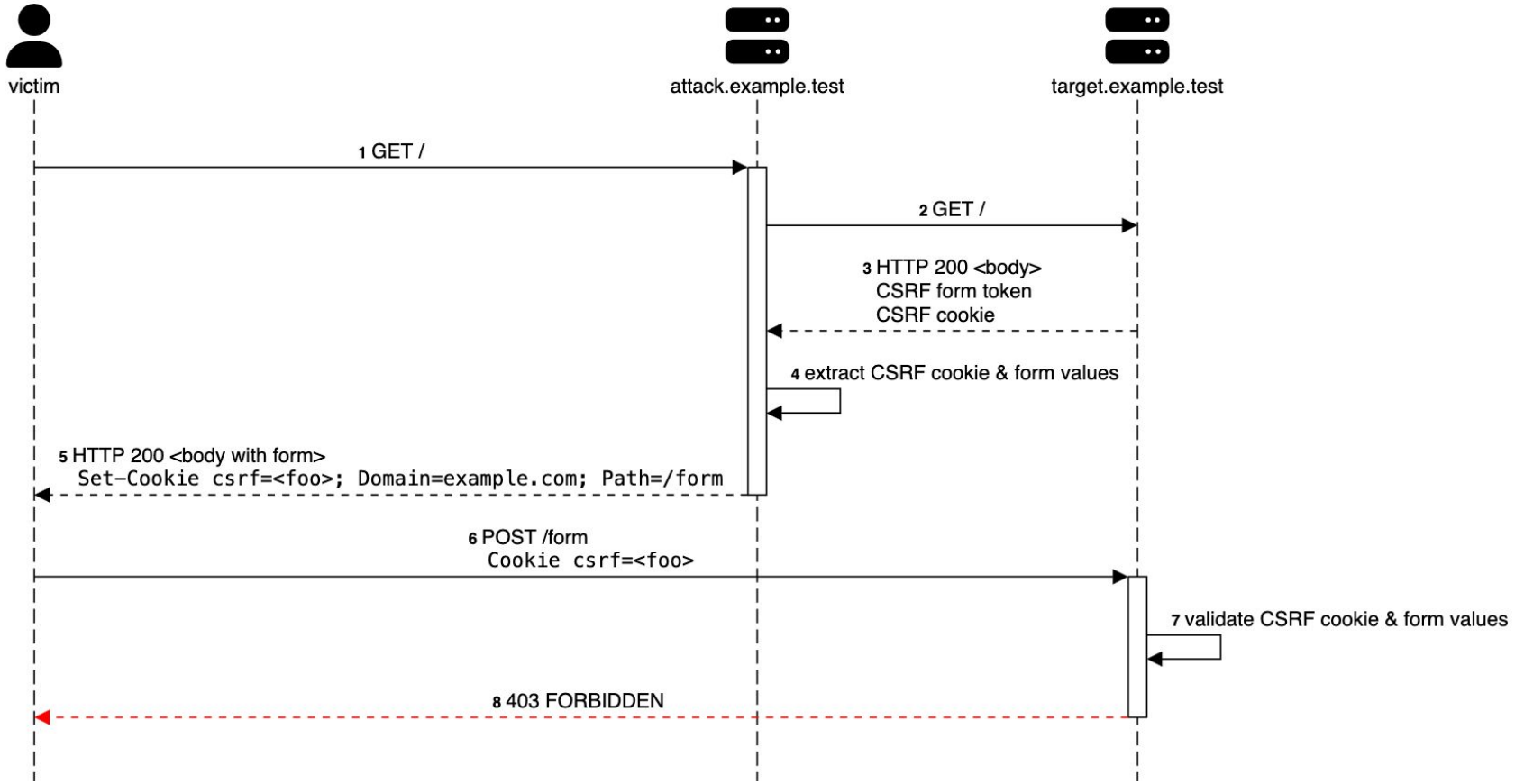
My first answer was completely wrong 🤦

I made a demo to correct my knowledge

Try it yourself @ https://patrickod.com/csrf

Source available @
https://github.com/patrickod/csrf-demo

CSRF demo app expectations



SUCCESSFUL POST REQUEST

What about the Origin checks?

```
[patrickod@gunter csrf (v1.7.2)]$ go test ./... -run Referer
--- PASS: ✅ TestNoReferer (0.00s)
--- PASS: ✅ TestBadReferer (0.00s)
--- PASS: ✅ TestTrustedReferer (0.00s)
--- PASS: ✅ TestWithReferer (0.00s)
PASS
ok   github.com/gorilla/csrf   (cached)
```

The bug hidden in plain sight

```
// Enforce an origin check for HTTPS connections.
if r.URL.Scheme == "https" {
    // return an error if Referer is empty or invalid
    referer, err := url.Parse(r.Referer())
    if err != nil || referer.String() == "" {
        r = envError(r, ErrNoReferer)
        cs.opts.ErrorHandler.ServeHTTP(w, r)
        return
    }
    // match Referer against the current URL
    valid := sameOrigin(r.URL, referer)
    ...
```

Where does `r.URL.Scheme` come from?

Read the documentation carefully 🧐

```
// URL specifies either the URI being requested (for server
// requests) or the URL to access (for client requests).
//
// For server requests, the URL is parsed from the URI
// supplied on the Request-Line as stored in RequestURI.
// For most requests, fields other than Path and RawQuery
// will be empty. (See RFC 7230, Section 5.3)
//
// ...
URL *url.URL
```

Read the documentation carefully 🧐

...

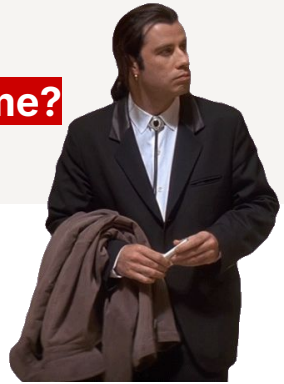
For example, a client wishing to retrieve a representation of the resource identified as

http://www.example.org/where?q=now

directly from the origin server would open (or reuse) a TCP connection to port 80 of the host "www.example.org" and send the lines:

```
GET /where?q=now HTTP/1.1
Host: www.example.org
```

Where is scheme?



Unfortunate bug coincidence in Go’s standard library

```
package main

import (
     "fmt"
     "net/http/httptest"
)

func main() {
     r := httptest.NewRequest("GET", "https://example.com", nil)
     if r.URL != nil {
          fmt.Printf("URL scheme is: %q", r.URL.Scheme)
     } else {
          fmt.Printf("URL is not set on request")
     }
}

[patrickod@gunter tmp ]$ go run ./main.go
URL scheme is: "https"
```

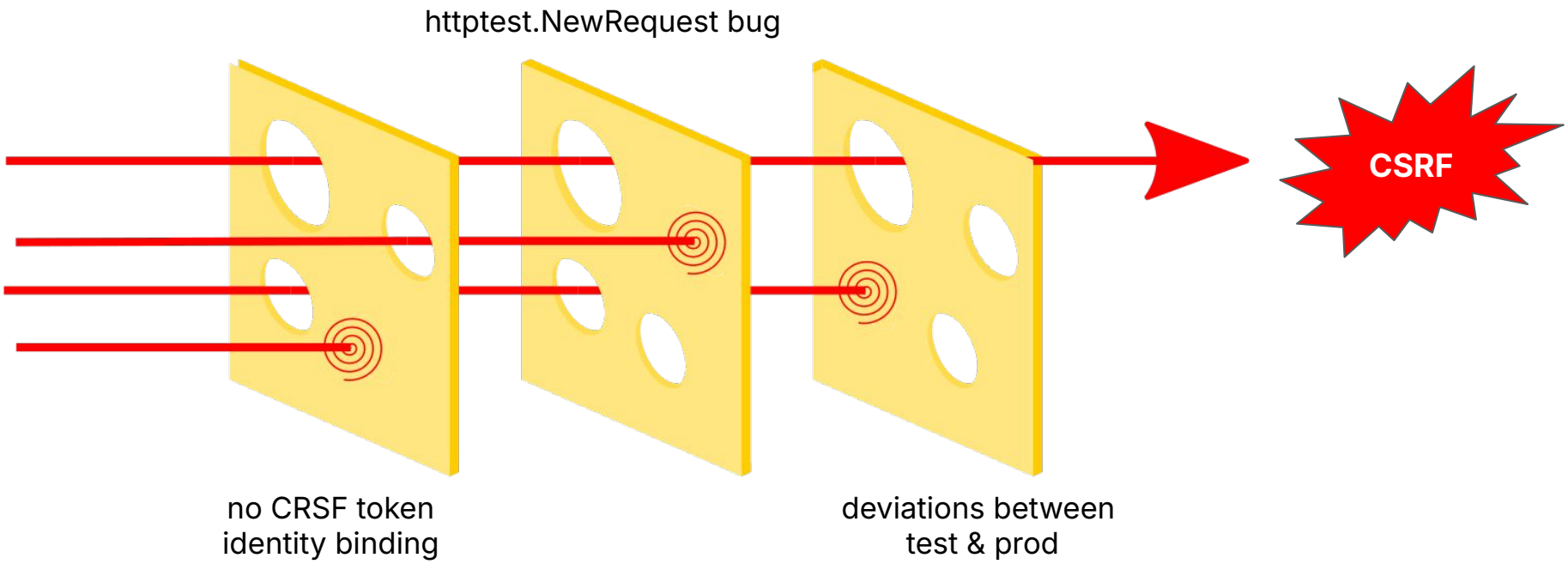
Recap

Origin checks only run when `r.URL.Scheme == “https”`

BUT

`r.URL.Scheme` is not populated in production requests

Swiss-Cheese Accident Model



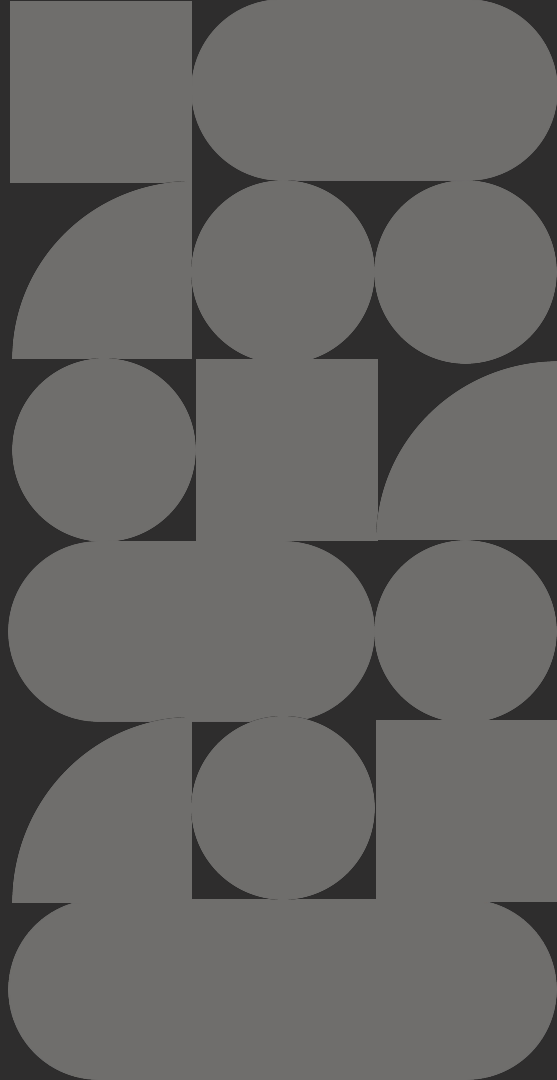
Disclosure Timeline

2015-08-04	Gorilla release candidate published with bug included.
2024-12-15	Vulnerability report made to gorilla project.
2024-01-06	Patch sent to gorilla project
2025-01-23	Report acknowledged; Patch accepted in Github main; Assigned **CVE-2025-24358**
2025-04-13	gorilla v1.73.0 released with patch. **GHSA-rq77-p4h8-4crw** advisory published.

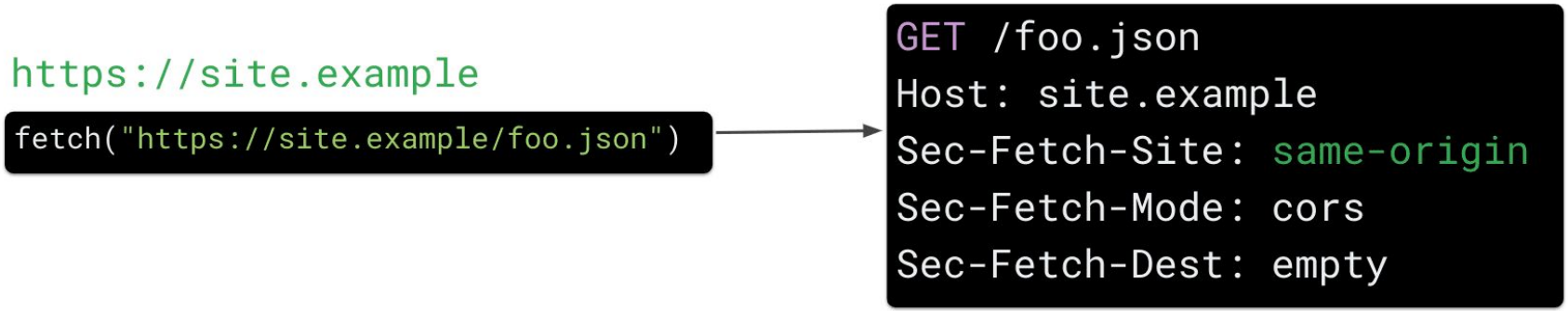
To update

```
go get -u github.com/gorilla/csrf@v1.73.0
```

Is there a better way?



Fetch Metadata Request Headers



Fetch Metadata Request Headers

https://evil.example

```
<img src="//site.example/foo.json" />
```



```
GET /foo.json
Host: site.example
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
```

How it looks in practice

```
switch r.Method {
case "GET", "HEAD", "OPTIONS": // allow idempotent methods
    h.ServeHTTP(w, r)
    return
}

if r.Header.Get("Sec-Fetch-Site") != "same-origin" {
    http.Error(w, "cross-origin request", http.StatusForbidden)
    return
}

h.ServeHTTP(w, r)
```

Browser Compatibility

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	WebView on iOS
`Sec-Fetch-Site`	✓ 76	✓ 79	✓ 90	✓ 63	✓ 16.4	✓ 76	✓ 90	✓ 54	✓ 16.4	✓ 12	✓ 76	✓ 16.4

Tip: you can click/tap on a cell for more information.

✓ Full support

Lessons learned

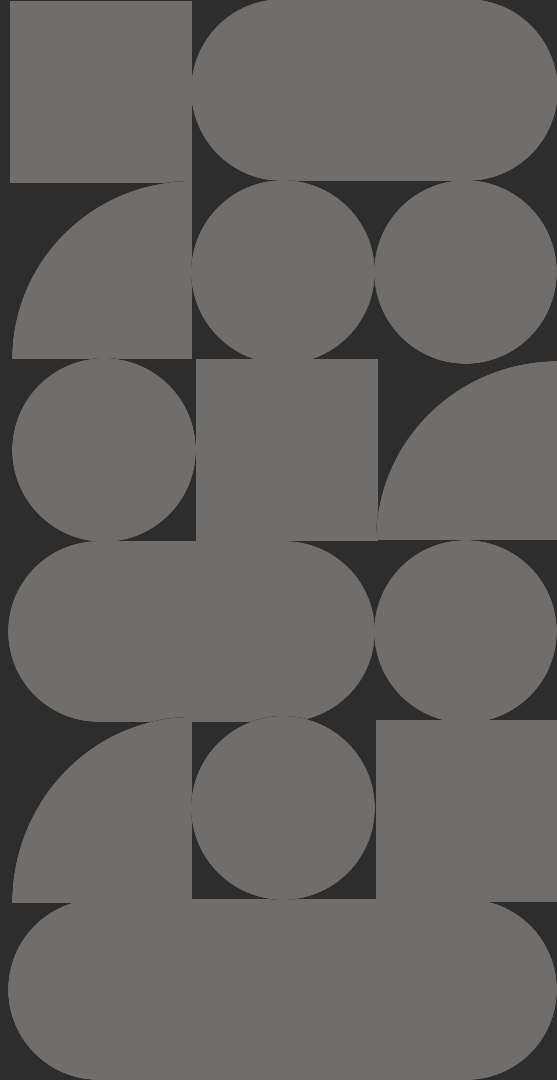
Lessons learned

⭐Unit tests are just the beginning

⭐Validate assumptions about the differences between production and test environments

⭐Fetch Metadata Request Headers are cool - we should use them!

Thank you!



Questions

Q&A (via Slido)

Feedback (2 questions)

Appendix

- Fetch Metadata Request Header W3C working draft: [link]
- GHSA-rq77-p4h8-4crw advisory [link]
- CSRF demo app [link]
- CSRF demo app [link]
- `go vet` to identify unsafe URL.Scheme consumption in request handlers [link]