Computer Security

13r. Review: Homework & Web Security

Paul Krzyzanowski • David Domingo • Ananya Jana

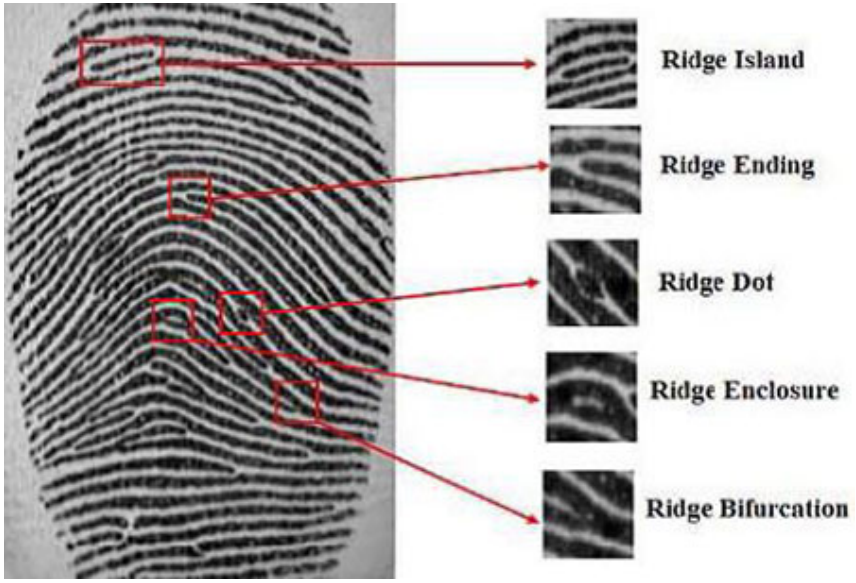
Rutgers University

Spring 2019

Question 1

What are minutiae points in a fingerprint?

"Minutiae can be defined as the points where the ridge lines end or fork."



- Their relative locations are what make fingerprints distinctive
- Registration and verification both require
  - Capturing the image
  - Processing it to identify minutiae points
  - Creating a graph: vertices = types of points, edges = distance between points
- Verification means comparing two graphs
  - But some points may be missing or others may appear
  - Need to make a decision of whether the match is good enough

Question 2

How does invisible reCAPTCHA (reCAPTCHA v3) work?

JavaScript that generates a **reCAPTCHA score** using "adaptive risk analysis."

It looks at the user's interaction with the browser to determine if it's a human or a bot. The score signifies how suspicious the interaction seems.

- The goal is to validate that the server is interacting with a human and not a bot
- A page that uses reCAPTCHA will have a script that:
  - Calls the JavaScript reCAPTCHA library to present the user with the puzzle
  - Gets a token when the user has entered the data and the server scored it
  - Makes a decision on whether the score is high enough for its needs
    - 0.0 = very likely to be a bot . . . 1.0 = very likely to be a human

Question 3

How does reCAPTCHA adapt for mobile devices?

Instead of presenting distorted text and having the user decode it, it presents an image labeling problem ("select all images that ...").

Ideally, reCAPTCHA hopes to avoid asking the user to prove he/she is a human.

On computers

- Look at movements of the mouse, IP addresses, cookies
- If the data looks trustworthy, don't bother the user
- If not sure, pop up a window and ask the user to decipher text

On phones/tablets

- Show the user a collection of images and have them select a specific subset (e.g., all images with a traffic light).

Question 4

What does the author state as the main problem of using a biometric for combined identification and authentication?

If a biometric is stolen, it serves as both the identity & authentication. There is no secret!

In many cases, biometrics are not well-suited for identification or authentication, even though they may try to do both

Question 4 (continued)

Identification

- Biometrics are public data: most of your biometrics (fingerprint, voice, face, iris, …) can be obtained quite easily
- They serve as identification, but they are not guaranteed to be absolutely unique
- Searching a database of biometric identities may be time-consuming since you need to perform pattern matches to determine if the sampled data is close enough

Authentication

- Common biometrics (fingerprint, face shape) have been compromised with little effort
- Cannot be revoked
  - You can change your password or replace a key/fob but not your iris
- Cannot be compartmentalized
  - You can have different passwords for google, amazon, twitter, Instagram, …
  - It is difficult to do that with biometric data

Question 5a

How does U2F operate without a shared secret?

It uses public key cryptography.

U2F uses a hardware authenticator (usually) that communicates with the client software (e.g., a browser) via USB or Bluetooth.

Quick: users don't need to type in one-time passwords

Just touch the authenticator and it will send the right response

Question 5b

What is the technique that U2F uses to authenticate with the server?

It uses challenge-based authentication:

Server sends a challenge (a random bunch of bits).

The authenticator signs the challenge with its private key.

The server validates it with the corresponding public key.

Question 5 discussion

Registration

- Server sends a challenge (a bunch of random bytes)
- Authenticator hardware generates a public/private key pair

  Unique to the server
- The private key never leaves the device
- Authenticator creates a “handle”: device info and public key
- Authenticator sends back:

  { Handle, challenge } signed by the private key
- Server stores key handle in its database

Question 5 discussion

Authentication

- Server sends a challenge (random bytes) and user's key handle
- Authenticator device looks up the private key for that service
- Authenticator sends back:
  { Client info, challenge } signed by the private key for that service
- Server validates the signature against the stored public key

Web security review – some key concepts

Cross-Origin Resource Sharing (CORS)

- Web pages normally enforce the **same-origin policy**
  - JavaScript can only access content from the same origin
    - Images, CSS, iframes within the page, embedded videos, other scripts, …
    - It cannot make asynchronous requests to other sites (e.g.,via XMLHttpRequest)
- CORS allows a server to define other origins (e.g., another domain name) as being equivalent
- Risk:
  - HTTP allows the client to specify the valid origins … that's good
  - The page origin can be specified in the header of the browser's request … that's not good since a malicious client can change it
  - But … it's easy for a server to just ignore the header

Injection Attacks

- Remote code execution attack
  - We studied these outside the web
  - These are data validation attacks when used on web pages
- Possible if user input is injected into a string and later evaluated by an interpreter (e.g., SQL, JavaScript, PHP)
- May be done via:
  - Parameters on the URL that are later used in a query, script, or other interpreted environment
  - User responses sent by submitting a form via HTTP POST and then processed by the server
- Preventing remote code execution
  - Avoid using user-supplied data inside any evaluated code

Cross-Site Scripting (XSS)

Form of an injection attack

- Reflected XSS
  - Non-persistent – the malicious code is not stored on the server
  - HTTP query parameters used without sanitization and contain code
  - Distributed as links on spam email or web sites
    - Links look legitimate because the domain name is a valid, trusted server
  - Attacks may take advantage of existing cookies that will authenticate a user
- Persistent XSS
  - Data provided by the attacker is stored and later presented to users going to valid pages
  - Example: place malicious JavaScript as part of response on a discussion thread
  - When any user navigates to that discussion, they load the JavaScript

Cross-Site Resource Forgery (XSRF)

- User sends unauthorized commands to the server
  - Commands are parameters on the URL

Just like a Reflected XSS attack, the attacker will get the user to click on a link

- The link is to a legitimate site (e.g., bank.com)
- Clicking on the link takes the user to the server (bank.com)
- The browser sends HTTP headers with cookies for bank.com
  - If the user is logged in, there will be an authentication cookie identifying the user
- The parameters on the URL provide a command
- The server thinks the request came from the user … because it did
  - Only the user did not intend to submit it.

The end