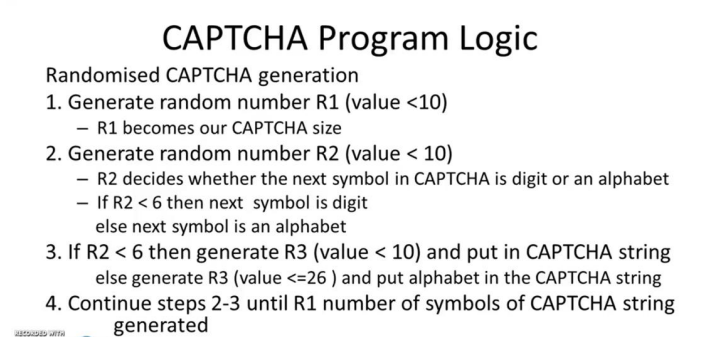
|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Information Security (116U01L602)** | **Semester:** | **VI** |
| **Date of Performance:** |  | **DIV/ Batch No:** | **B2** |
| **Student Name:** | **Akshat** | **Roll No:** | **16010122221** |

|  |
| --- |
| **Title:**Implementation of CAPTCHA |

|  |
| --- |
| **Objectives:** |
| Implementation of the program , Exploring other forms / types of CAPTCHAs |

|  |
| --- |
| **Abstract:** |
| CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a type of security measure known as challenge-response authentication. CAPTCHA helps protect you from spam and password decryption by asking you to complete a simple test that proves you are human and not a computer trying to break into a password protected account. A CAPTCHA test is made up of two simple parts: a randomly generated sequence of letters and/or numbers that appear as a distorted image, and a text box. To pass a the test and prove your human identity, simply type the characters you see in the image into the text box. In this experiment we understand the algorithm to generate CAPTCHA and implement it in any programming language. After implementation, we give a quiz to test our understanding of CAPTCHA |

|  |
| --- |
| **Related Theory:**CAPTCHA stands for the Completely Automated Public Turing test to tell Computers and Humans Apart. CAPTCHAs are tools you can use to differentiate between real users and automated users, such as bots. CAPTCHAs provide challenges that are difficult for computers to perform but relatively easy for humans. For example, identifying stretched letters or numbers, or clicking in a specific area. CAPTCHAs are used by any website that wishes to restrict usage by bots. Specific uses include: • Maintaining poll accuracy—CAPTCHAs can prevent poll skewing by ensuring that each vote is entered by a human. Although this does not limit the overall number of votes that can be made, it makes the time required for each vote longer, discouraging multiple votes. • Limiting registration for services—services can use CAPTCHAs to prevent bots from spamming registration systems to create fake accounts. Restricting account creation prevents waste of a service’s resources and reduces opportunities for fraud. • Preventing ticket inflation—ticketing systems can use CAPTCHA to limit scalpers from purchasing large numbers of tickets for resale. It can also be used to prevent false registrations to free events. • Preventing false comments—CAPTCHAs can prevent bots from spamming message boards, contact forms, or review sites. The extra step required by a CAPTCHA can also play a role in reducing online harassment through inconvenience |
|  |



|  |
| --- |
| **Implementation Details:**  **Captcha Program (Simple text based)**  import random  import string  def **generateCaptcha**():      R1 = random.randint(1, 9)  # Ensure at least one character in captcha      captcha = []      for i in range(R1):          R2 = random.randint(0, 9)          if R2 < 6:              R3 = random.randint(0, 9)              captcha.**append**(str(R3))          else:              R3 = random.choice(string.ascii\_letters)              captcha.**append**(R3)      return ''.**join**(captcha)  captcha = **generateCaptcha**()  **print**("Generated Captcha:", captcha)  user\_input = **input**("Enter the captcha: ")  if user\_input == captcha:  **print**("Captcha matches")  else:  **print**("Captcha didn't match") |
| **Image based Captcha:**  from captcha.image import ImageCaptcha  import random  import string  def **generateCaptcha**():      R1 = random.randint(1, 9)  # Ensure at least one character in captcha      captcha = []      for i in range(R1):          R2 = random.randint(0, 9)          if R2 < 6:              R3 = random.randint(0, 9)              captcha.**append**(str(R3))          else:              R3 = random.choice(string.ascii\_letters)              captcha.**append**(R3)      return captcha  def **createImageCaptcha**(captcha\_text, image\_path='CAPTCHA.png'):      image = ImageCaptcha(width=280, height=90)      data = image.generate(captcha\_text)      image.write(captcha\_text, image\_path)      return image\_path  captcha = **generateCaptcha**()  captcha\_text = ''.**join**(captcha)  image\_path = **createImageCaptcha**(captcha\_text)  **print**("Image-based Captcha generated. Please enter the following captcha:")  **print**(f"Image path: {image\_path}")  user\_input = **input**("Enter the captcha shown in the image: ")  if user\_input == captcha\_text:  **print**("Captcha matches")  else:  **print**("Captcha didn't match") |

|  |
| --- |
|  |

|  |
| --- |
| **Conclusion:** |
| Conclusion:-  In this experiment, we explored the fundamentals of CAPTCHA, its importance in securing digital platforms, and its implementation using both text-based and image-based approaches. By developing a program to generate CAPTCHA and test user inputs, we gained hands-on experience with the underlying algorithms that differentiate human users from automated bots. This practical understanding reinforces the significance of CAPTCHAs in safeguarding online services against spam, automated abuse, and fraudulent activities. The exercise not only enhanced our programming skills but also deepened our appreciation for effective cybersecurity measures in modern computing environments. |

|  |
| --- |
| **Post-Lab Questions:** |
|  |