#3

1人：PB19111701

1 Same Origin Policy

Question:

We discussed in lecture how the DOM same-origin policy defines an origin as the triple (protocol, domain, port). Explain what would go wrong if the DOM same-origin policy were only defined by domain, and nothing else. Give a concrete example of an attack that a network attacker can do in this case, but cannot do when using the standard definition of the same-origin policy.

Answer:

If the DOM same-origin policy were only defined by domain, websites with the same domain would be able to access the DOM of each other, which means compromise of one of those websites would result in the compromise of all those websites.

Here is a concrete example, an user now opens two websites with the same domain, one of which runs in HTTP and the other runs in HTTPS. The user fills and commits his username and password in the website running in HTTPS. If DOM same-origin policy were only defined by domain, the other website running in HTTP can access the DOM of the website running in HTTPS, thereby obtaining the secret password of the user. Now, if an attack eavesdrops on the website running in HTTP, since it's not encrypted, the attacker is possible to get the password of the user as well. In this case, a network attack, which is infeasible with a standard same-origin policy, is completed.

2 Cross Site Script Inclusion (XSSI) Attacks

Question:

Consider a banking web site bank.com where after login the user is taken to a user information page: https://bank.com/accountInfo.html

The page shows the user's account balances. Here accountInfo.html is a static page: it contains the page layout, but no user data. Towards the bottom of the page a script is included as:

```
<script src="//bank.com/userdata.js"> </script> <!--*-->
```

The contents of userdata.js is as follows:

```
displayData(
    {
        "name": "John Doe",
        "AccountNumber": 12345,
        "Balance": 45
    }
)
```

The function displayData is defined in accountInfo.html and uses the provided data to populate the page with user data.

The script userdata.js is generated dynamically and is the only part of the page that contains user data. Everything else is static content.

Suppose that after the user logs into his or her account at bank.com the site stores the user's session token in a browser cookie.

a

Consider user John Doe who logs into his account at bank.com and then visits the URL https://evil.com/. Explain how the page at evil.com can cause all of John Doe's data to be sent to evil.com. Please provide the code contained in the page at evil.com. The code can be pseudocode.

Answer:

Since script can be loaded in a cross-site way, `evil.com` can steal all of the user's data with the same code in `bank.com`

```
<script src="//bank.com/userdata.js"> </script>
```

Given that the user has logged into his/her account at `bank.com`, sensitive data stored in dynamic script can be easily stolen via XSSI attacks.

b

How would you keep accountInfo.html as a static page, but prevent the attack from part (a)? You need only change line (*) and userdata.js. Make sure to explain why your defense prevents the attack.

Hint: Try loading the user's data in a way that gives bank.com access to the data, but does not give evil.com access. In particular, userdata.js need not be a JavaScript file

Answer:

1. Store data in a JSON file instead of in a dynamic script;
2. Use POST instead of line (*) to get confidential data from JSON;
3. Set cookie as same-site (or use other protection methods, e.g. set a secret validation token) to avoid CSRF;

In this case, since the data is stored in JSON file, the attacker can't include/reference the data using XSSI attack as part (a) does. Moreover, since methods have been adopted to prevent CSRF attack, the attacker can't access the data using POST either. Consequently, the risk of XSSI attack is eliminated.

3

Question:

Two new extensions to DNS have been recently ratified by the Internet standards community: DNS-over-HTTPS and DNS-over-TLS. The protocols work similarly to DNS except that DNS queries are sent over a standard encrypted HTTPS or TLS tunnel.

a

What is one DNS attack that DNS-over-HTTPS protects against? (3 points)

Answer:

DNS Spoofing

b

What is one DNS attack that DNS-over-HTTPS does not protect against? (5 points)

Answer:

1. DNS Cache Poisoning;
2. DNS Rebinding

c

Do DoH or DoT prevent DNS from being used as DDoS amplifier? Why or why not? (3 points)

Answer:

DoH/DoT **can** prevent DNS from being used as DDoS amplifier. Typically, using DNS as DDoS amplifier requires that the connection is a UDP connection, so that the victim is not able to distinguish regular DNS response with malicious DNS packet. The adoption of DoH/DoT makes sure that a DNS response will be accepted/processed only if a TCP connection is set up before, thereby preventing DNS from being used as DDoS amplifier.

d

Do DoH or DoT protect against DNS rebinding attacks? Why or why not? (4 points)

Answer:

DoH/DoT **can't** prevent against DNS rebinding attacks. Even if DNS is transferred over HTTPS/TLS, the content of the DNS response is still up to the DNS server. If the DNS server is malicious, it can still provide the user with wrong IP address after the last DNS response has soon expired. In this case, a DNS rebinding attack will still occur.

4 CSRF Defenses

a

In class we discussed Cross Site Request Forgery (CSRF) attacks against websites that rely solely on cookies for session management. Briefly explain a CSRF attack on such a site. (3 points)

Answer:

In such case, the attacker can initiate a malicious request (e.g. use POST/GET) to the victim website. As long as there exit in-scoop cookies, the browser will send them along with the corresponding request. Since the victim website relies solely on cookies for session management, the malicious request will be accepted and then executed without any further checking. Therefore, a CSRF attack succeeds.

b

A common CSRF defense places a token in the DOM of every page (e.g., as a hidden form element) in addition to the cookie. An HTTP request is accepted by the server only if it contains both a valid HTTP cookie header and a valid token in the POST parameters. Why does this prevent the attack from part (a)? (3 points)

Answer:

Since the token is unique for every single page, it is difficult for the attacker to obtain the exact token for the POST it wants to initiate. Therefore, even if the attacker succeeds to send the server a malicious POST with the user's cookies, the server will discard the request since it doesn't carries a correct token. In this case, the CSRF attack in part (a) can be prevented.

c

One approach to choosing a CSRF token is to choose one at random. Suppose a web server chooses the token as a fresh random string for every HTTP response. The server checks that this random string is present in the next HTTP request for that session. Does this prevent CSRF attacks? If so, explain why. If not, describe an attack. (3 points)

Answer:

No. The attacker can initiate a request first with a random token. Undoubtedly, the request won't be accepted since it doesn't have a valid token. However, a HTTP response will still be sent nevertheless. Therefore, the attacker can get the secret token for the next session. The attacker can initiate another request with the obtained token immediately, and this forged request will be accepted.

d

Another approach is to choose the token as a fixed random string chosen by the server. That is, the same random string is used as the CSRF token in all HTTP responses from the server over a given time period. Does this prevent CSRF attacks? If so, explain why. If not, describe an attack. (3 points)

Answer:

Yes. Since the token is fixed in a given period of time, it can be securely transmitted in advance. Therefore, it will be almost impossible for the attacker to provide or obtain a valid token, thereby preventing from the CSRF attack.

e

Why is the Same-Origin Policy important for the cookie-plus-token defense? (3 points)

Answer:

Without same-origin policy, the attacker website can get the access to the DOM of the victim website. Therefore, it is possible for the attacker to get the token for a specific request, thereby conducting a CSRF attack successfully.

5 Content Security Policies

Recall that content security policy (CSP) is an HTTP header sent by a website to the browser that tells the browser what it should and should not do as it is processing the content. The purpose of this question is to explore a number of CSP directives. Please use the CSP specification to look up the definition of the directives in the questions below.

a

(3 points) Explain what the following CSP header does:

> Content-Security-Policy: script-src 'self'

What is the purpose of this CSP directive? What attack is it intended to prevent?

Answer:

behavior:

1. scripts can only be loaded from the same domain as the page;
2. no inline `<script></script>` will be executed;
3. no inline `<style></style>` will be executed.

purpose:

to make sure that only the same-site scripts can be executed;

attack it prevents:

XSS attack

b

(3 points) What does the following CSP header do:

> Content-Security-Policy: frame-ancestors 'none'

What attack does it prevent?

Answer:

behavior:

1. no url can frame the current resource, i.e. the current resource can't be loaded inside a frame or an iframe;
2. no inline `<script></script>` will be executed;
3. no inline `<style></style>` will be executed.

attack it prevents:

clickjacking attack

c

(9 points) What does the following CSP header do:

> Content-Security-Policy: sandbox 'allow-scripts'

Suppose a page loaded from the domain www.xyz.com has the sandbox CSP header, as above. This causes the page to be treated as being from a special origin that always fails the same-origin policy, among other restrictions. How does this impact the page's ability to read cookies belonging to www.xyz.com using JavaScript? Give an example where a web site might want to use

this CSP header.

Answer:

behavior:

1. load resources in a sandbox, but allow JavaScript execution inside the sandbox;
2. no inline `<script></script>` will be executed;
3. no inline `<style></style>` will be executed.

How does this impact the page's ability to read cookies belonging to `www.xyz.com` using JavaScript?

Since `www.xyz.com` is treated as if it fails the same-origin policy, now the page which loads `www.xyz.com` inside a sandbox can also access the cookies belonging to `www.xyx.com` using JavaScript.

Give an example where a web site might want to use this CSP header.

> reference: https://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/

`allow-scripts` is required if the page loaded into the frame runs some JavaScript to deal with user interaction.

6 Stealth Port Scanning

Recall that the IP packet header contains a 16-bit identification field that is used for assembling packet fragments. IP mandates that the identification field be unique for each packet for a given (SourceIP,DestIP) pair. A common method for implementing the identification field is to maintain a single counter that is incremented by one for every packet sent. The current value of the counter is embedded in each outgoing packet. Since this counter is used for all connections to the host we say that the host implements a global identification field.

a

(5 points) Suppose a host P (whom we'll call the Patsy for reasons that will become clear later) implements a global identification field. Suppose further that P responds to ICMP ping requests. You control some other host A. How can you test if P sent a packet to anyone (other than A) within a certain one minute window? You are allowed to send your own packets to P.

Answer:

In this certain minute window, host A will repeatedly send ICMP ping requests to host P and check the IP ID of the responses. If host P, during the window, sends any packet to another host, the IP ID of the responses will become discontinuous. Host A only needs to check for the existence of those discontinuous IP ID from the ICMP ping responses to become aware of the case that P has sent a packet to another host.

b

(10 points) Your goal now is to test whether a victim host V is running a server that accepts connections to port n (that is, test if V is listening on port n). You wish to hide the identity of your machine A and therefore A cannot directly send a packet to V, unless that packet contains a spoofed source IP address. Explain how to use the patsy host P to test if V accepts connections to port n.

Hint: Recall the following facts about TCP:

- A host that receives a SYN packet to an open port n sends back a SYN/ACK response to the source IP.
- A host that receives a SYN packet to a closed port n sends back a RST packet to the source IP.
- A host that receives a SYN/ACK packet that it is not expecting sends back a RST packet to the source IP.
- A host that receives a RST packet sends back no response.

Answer:

> reference: https://www.icir.org/vern/papers/norm-usenix-sec-01-html/node8.html

During the whole attack, host A will repeatedly send ICMP ping requests to host P and checks the IP ID of the responses. At the same time, host A will send a SYN packet to different ports of host V, and fake the source address of the packet as host P. If the port is a closed port, it will send a RST packet to host P, which will not be responded when host P receives it. However, if the port is an open port, it will send a SYN/ACK packet to host P. Since it is not expected by host P, host P will send a RST packet to host V, increasing the global IP ID by one. Therefore, the attacker will notice a discontinuous IP ID from ICMP ping responses, indicating that the port it has just sent a SYN packet is an open port. In this case, a stealth post scanning succeeds, i.e. the attacker is able to test if the victim accepts connections on a given port without exposing its identity to the victim.

7 Denial of Service attacks

a

(5 points) Using a TCP SYN spoofing attack, the attacker aims to flood the table of TCP connection requests on a system so that it is unable to respond to legitimate connection requests. Consider a server system with a table for 256 connection requests. This system will retry sending the SYN-ACK packet five times when it fails to receive an ACK packet in response, at 30 second intervals, before purging the request from its table. Assume that no additional countermeasures are used against this attack and that the attacker has filled this table with an initial flood of connection requests. At what rate must the attacker continue to send TCP connection requests to this system in order to ensure that the table remains full? Assuming that the TCP SYN packet is 40 bytes in size (ignoring framing overhead), how much bandwidth does the attacker consume to continue this attack?

Answer:

$$\text{rate} = \frac{256}{30 \times 5} = 1.7 \text{ request/second}$$

$$\text{brandwitdh} = \text{rate} \times \text{packet size} = 1.7 \times 40 \times 8 \text{ bps} = 544 \text{ bps}$$

b

(5 points) In order to implement a DNS amplification attack, the attacker must trigger the creation of a sufficiently large volume of DNS response packets from the intermediary to exceed the capacity of the link to the target organization. Consider an attack where the DNS response packets are 500 bytes in size (ignoring framing overhead). How many of these packets per second must the attacker trigger to flood a target organization using a 0.5-Mbps link? A 2-Mbps link? Or a 10-Mbps link? If the DNS request packet to the intermediary is 60 bytes in size, how much bandwidth does the attacker consume to send the necessary rate of DNS request packets for each of these three cases?

Answer:

0.5-Mbps link:

$$\#\text{packet} = \frac{0.5 \times 10^6}{500 \times 8} = 125$$

$$\text{brandwidth} = \#\text{packet} \times \text{packet size} = 125 \times 60 \times 8 \text{ bps} = 60000 \text{ bps} = 0.06 \text{ Mbps}$$

2-Mbps link:

$$\#\text{packet} = \frac{2 \times 10^6}{500 \times 8} = 500$$

$$\text{brandwidth} = \#\text{packet} \times \text{packet size} = 500 \times 60 \times 8 \text{ bps} = 240000 \text{ bps} = 0.24 \text{ Mbps}$$

10-Mbps link:

$$\#\text{packet} = \frac{0.5 \times 10^6}{500 \times 8} = 2500$$

$$\text{brandwidth} = \#\text{packet} \times \text{packet size} = 2500 \times 60 \times 8 \text{ bps} = 1200000 \text{ bps} = 1.2 \text{ Mbps}$$