Intro to Cybersecurity – Spring 2023

Homework/Lab #3

Due: Tuesday, May 23, 2023

Highlights

- Expected contribution towards the final score: 6%.
- **You should work on this homework individually or in teams of up to 5 members (highly recommended). One submission per team.**
- **Submit your work in PDF as a group** through the USTC Blackboard
  - Choose one group named as “Homework/Lab #3 [1-50]” on Blackboard (工具->小组) , and submit as the group, which makes it easier for the teaching team to register your score.

1) **Same Origin Policy**

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.

Solution:
In short, a network server can be shared by multiple tenants, and thus the TCP ports on the same host can bind to network applications belonging to different tenants, among which, some can be attackers.

In another word, the same domain and the IP addresses it resolves to may be used to run multiple applications that belong to different owners, and considering them as the same origin can incur security risks. The same is for the protocol field.

Consider a domain benign.com which is resolved to IP address A. In the meantime, an attacker has deployed another web application on the same IP but with TCP port 8080. If you consider benign.com and benign.com:8080 as the same origin, it will incur risks to benign.com. Specifically, an attacker can redirect a victim to benign.com:8080 wherein the attack loads benign.com:80 in an iframe. Since both are considered as the same origin, The JS code of benign.com:8080 will be able to read and write resources loaded for the iframe of benign.com

2) **Cross Site Script Inclusion (XSSI) Attacks (20 points)**

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 in to his or her account at bank.com the site stores the user's session token in a browser cookie.

a) **(10 points)** 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.

b) **(10 points)** 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

Solution

a)

```
function displayData(userData) {

          sendToEvil(userData);

}
```

b) check the origin of the current top window

```
if (window.location == XX) {

          displayData(XX);

}
```

3) **(15 points)** 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)**
b) What is one DNS attack that DNS-over-HTTPS does not protect against? **(5 points)**
c) Do DoH or DoT prevent DNS from being used as a DDoS amplifier? Why or why not? **(3 points)**
d) Do DoH or DoT protect against DNS rebinding attacks? Why or why not? **(4 points)**

Solution

a) It can prevent MITM attacks, as well as cache poisoning through Kaminsky Attack
b) Rebinding attack, Besides, once the DNS server or resolver is compromised, poisoning attacks can still happen
c) Yes, TCP and TLS handshakes must be completed before doing DNS queries, there is no place for the attacker to spoof the source IP addresses
d) No, in rebinding attack, the DNS name server itself is controlled by the attacker

4) CSRF Defenses **(15 points)**

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

Solution:

a) A CSRF attack denotes the process whereby an attacking website forges and initiates HTTP requests towards a victim website on behalf of a victim user.
b) The SOP policy prevents the attack from reading the cookies or accessing the randomly generated token
c) Yes, it can prevent CSRF attacks as long as the token is randomly generated, and each new request will be assigned with a different token. Also, the token should not be predictable and should be transmitted in a secure channel
d) No, an attacker can easily grab such a token, and reuse it when doing CSRF attacks. Channels for grabbing such a token include registering under the target website as a legitimate user and sending requests for fetching such a token
e) Without the SOP policy, attackers can easily access the token as well as the cookie, and therefore, can forge any requests at will

5) Content Security Policies **(15 points)**

Recall that content security policy (CSP) is an HTTP header sent by a web site 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?

b) **(3 points)** What does the following CSP header do:
Content-Security-Policy: frame-ancestors 'none'

What attack does it prevent?

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.

Solution

a) As illustrated in this official wiki, this CSP policy denotes only javascript code from script files of the same origin can be executed on this page, external js files, or any inline js logic will be ignored.

b) As specified in this doc, frame-ancestors 'none' will block any pages from framing the given webpage. It is commonly used to prevent CSRF attacks

c) As explained in this reference, the sandbox directive enables "a sandbox for the requested resource similar to the <iframe> sandbox attribute. It applies restrictions to a page's actions including preventing popups, preventing the execution of plugins and scripts, and enforcing a same-origin policy."

   i) Since a webpage with sandbox directive enforced will be treated as a special origin that is different from its true origin, although it can execute Javascript, the JS code cannot access the cookies registered under its true origin. A real-world scenario can be that a website doesn't want some requests or pages (a webpage displaying an image) from accessing sensitive cookies such as session-related ones

6) **SStealth Port Scanning (15 points)**

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*.
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..

Solution:

a) *You can send two ICMP packets to host P, one at the beginning of the one-minute window, and the other at the end. If the identification field increments by only one, then, host P didn't communicate with other hosts during this time window, otherwise, it has.*
b) *Similarly, host A can send a TCP SYN packet to port n of host V with the source IP forged as host P. If host V opens n, it will send a TCP SYN/ACK back to host P. Since P doesn't expect such a SYN/ACK packet, it will send a RST packet to host V, which will increment the identification number by one. However, if host V doesn't open port n, it will send a RST packet to host P, for which, host P will give no response, and thus will not increase the identification number. By observing whether the identification number changes during this operation, you can decide with some probability whether host V opens port n. You can redo this operation for multiple rounds, to increase the confidence.*

7) **Denial of Service attacks (10 points)**
   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?

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?

Solution:

a) *For a TCP SYN spoofing attack, on a system with a table for 256 connection requests, that will retry 5 times at 30 second intervals, before purging the request from its table, each connection request occupies a table entry for 6 × 30 secs (initial + 5 repeats) = 3 min. In order to ensure that the table remains full, the attacker must continue to send 256/ 3 or about 86 TCP connection requests per minute? Assuming the TCP SYN packet is 40 bytes in size, this consumes about 86 × 40 × 8 / 60, which is about 459 bits per second, a negligible amount.*

b) *On a 0.5-Mbps link,* $500000 \div (8 \times 500) =$ *125 packets, each of 500 bytes, are needed per second. 500 pps are needed to flood a 2-Mbps link, and 2500 pps to flood a 10-Mbps link. Assuming a 60-byte DNS request packet then 125 × 60 × 8 = 60 Kbps is needed to trigger the flood on a 0.5-Mbps link, 240 kbps to flood the 2-Mbps link, and 1.2 Mbps to flood the 10-Mbps link. In all cases the amplification is 500 / 60 = 8.3 times.*