**Secure EC2 Instances**

1. One would need a **VPC**. A VPC is short for **Virtual Private Cloud.** A VPC is a dedicated virtual network for your AWS account. It's logically separated from the rest of the AWS Cloud's virtual networks.
2. Another way to secure ECT is to implement **MFA**, which stands for **multi-factor authentication.** This is to prevent any users that does not have the rights to login into accounts without a security token.
3. **For each subnet, create a separate route table.** A route table is made up of a set of rules called routes that control where network traffic from your subnet or gateway goes. Which ties into number one (VPC) because each subnet must be associated with a route table to see where the traffic going.
4. We have **VM-Series firewall**. The Amazon Web Services (AWS) public cloud and AWS GovCloud both support the VM-Series firewall. It can then be set up to secure access to applications hosted on EC2 instances and stored in an AWS Virtual Private Cloud (VPC).

**Common vulnerabilities**

1. **Use of Public Subnets in Excess:** This strategy, however, can be highly harmful. Public subnets are routed to Internet gateways, which make them available to the public. As a result, any sensitive data stored on the subnet is exposed to attack. Which result to what I said above about “**Private Subnets**”
2. **Database Origin Servers That Have Been Revealed:** Unless particularly requested, the IP addresses of database origin servers should never be made visible to any person or application.
3. **Request Forgery on the Server (SSRF):** Is an attempt to obtain access to instance metadata by abusing lawful AWS features. An attacker may be able to obtain credentials for an IAM role associated with the instance and achieve privileged access to the target application if they are successful.

**Link**

[Use Case: Secure the EC2 Instances in the AWS Cloud (paloaltonetworks.com)](https://docs.paloaltonetworks.com/vm-series/9-1/vm-series-deployment/set-up-the-vm-series-firewall-on-aws/use-case-secure-the-ec2-instances-in-the-aws-cloud)

[Message from Cobalt](https://cobalt.io/blog/aws-cloud-security)

**React.js security**

**Common vulnerabilities to be aware of**

* **Cross-Site Scripting -** This is a client weakness that can result in major issues. When an attacker can fool a website into executing arbitrary JavaScript code in the browsers of its users, this is known as an XSS assault.
* **The reflected XSS attack -** It can really happen when a user clicks on a link with involves text information that the browser converts into code. This could be a form field that necessitates an user requests, for example.
* **A stored XSS attack -** When an attacker gains access to a server, and the code performed here on server is capable of generating information for the client's site, this occurs. Uploading comments and photographs to servers are common avenues for such assaults. A good example is the Samy worm, which took advantage of a MySpace XSS flaw. It was one of the most quickly spreading viruses in history.
* **Insecure randomness and links -** For web developers, this means that if the content of links is depending on data given by the user, an attacker can use JavaScript to inject malicious code. An attacker script will be run in the browser if the user clicks on this URL. This isn't limited to React projects, and the answer is application-specific. You may also need to make server adjustments. It's best to utilize a whitelisted protocol (for example, http:) for the links and screen the HTML entities. This isn't limited to links. However, in React applications, they are the most likely target. If an attacker can conjure up the right conditions, any element can become subject to this attack. If an attacker has control over the URI (Uniform Resource Identifier) value of an element, it can be vulnerable to this attack.

**React.js Best Practices**

* Remove or disable any markup that could potentially include instructions for running the code as a defense against XSS vulnerabilities. This includes HTML elements like script>, object>, embed>, and link>.
* Jscrambler is a tool that protects React.js source code.
* Exploit script injection flaws in React.js apps.
* Make use of snippet libraries such as ES7 React, Redux, and JS Snippets, among others. They'll add extra security and ensure that your code is bug-free.

**Link**

[React.js security best practices. While React is considered to be quite… | by UppLabs | Medium](https://upplabs.medium.com/react-js-security-best-practices-62b9a281cc42)

**Django security Best Practices**

1. **Use SSL**

* Put your site behind a secure HTTPS connection. Someone could snoop your (or your users') password if you aren't utilizing HTTPS when in a coffee shop, an airport, or another public location. In the Django docs, you can learn more about enabling SSL and any other actions you might need to take.

1. **Change the URL** - /admin/ should be replaced with anything else as the default admin URL. The instructions are in the Django documentation, but in a nutshell, change admin/ to something else in your URL conf:

* Even more protection can be achieved by hosting the admin on a separate domain. If you need even more security, serve the admin through a VPN or from a private location.

1. **Use 'django-admin-honeypot' -** Install the django-admin-honeypot library on your old /admin/ URL to collect attempts to attack your site after you've relocated it to a different URL (or even elected to host it on its own domain). When someone tries to get in to your previous /admin/ URL, django-admin-honeypot generates a phony admin login screen and emails your site administrators.

* The attacker's IP address will be included in the email created by django-admin-honeypot, so if you detect repeated login attempts from the same IP address, you can restrict that address from using your site for further security.

1. **Use two-factor authentication -** When you demand a password plus something else to authenticate a user for your site, you're using two-factor authentication (2FA). Apps that need a password and then text you a second login code before allowing you to log in are likely employing two-factor authentication (2FA).

The reactjs and Django best practices are SOME of many. Using these as of today can help secure your apps that much more if implementing these best practices.

**Link**

[10 tips for making the Django Admin more secure | Opensource.com](https://opensource.com/article/18/1/10-tips-making-django-admin-more-secure)