Toptal Secure Web Application Design Project

Table of Contents:

Introduction

AWS Security compliance

Shared Responsibility

Software Production Process

Security framework
- Identify
- Protect
- Detect
- Respond
- Recover
- Additional guidance

CIS Benchmarks

Application Architecture
- Cloudfront
- WAF
- S3
- Route53
- ALB
- API Gateway (write)
- RDS cluster (write master)
- RDS cluster (read)
- RDS cluster (standby)
- API Gateway (read)
- Elasticache
- API Gateway (write, async)
- SQS
- Lambda worker
- DynamoDB
- IAM

Responding to attacks

Long-Term Roadmap

Vulnerability management
Security awareness
Pipeline maintenance
Continual review of attack surface
Change management and Change Advisory Board (CAB)
Ongoing legal review
Measure cyber security performance
Monitor for third party risk
Communicate the state of the security of the platform upwards
Conclusion

Introduction

This document describes a secure implementation of the provided example web application architecture. This architecture will not only be secure but allow the company to achieve high development velocity and focus on their core competency while minimizing or transferring risk while protecting the confidentiality, integrity, and availability (CIA) of our assets.

Being that the company has been specified to use cloud managed services only, we have chosen to use Amazon Web Services and will be using their services and functionality wherever possible to meet the needs of this project. This not only has security benefits but makes things operationally simpler given the complexity of the environment. We do note that there are many other non-AWS tools commercially available that could be slotted in for any one of these AWS tools and might be a better fit depending on the specific details of the situation such as choice of programming language, framework, license model, etc.

We will get as much infrastructure and architecture as possible out of scope for the company, including the potential architectural vulnerabilities which were identified as potential data breach targets by the independent security researcher.

As this is a fictional exercise with much left open for interpretation, a number of assumptions have been made. In the real world, the details matter. Certain issues such as expense, cloud vendor lock-in considerations, specific security tool feature sets etc. were not listed among the requirements and as such are not considered here whereas in the real world this is often an issue. Instead, we have taken the most direct route to the solution given the specifications.

AWS Security compliance

AWS meets various compliance requirements including SOC 2 Type 3 (Security, Availability, Processing Integrity, Confidentiality, and Privacy) which would be most applicable to a SaaS as described in this document. Legal shall ensure that our contract with AWS contains the appropriate language to ensure their continued compliance and regular audits and that all of the AWS services which we are using are in scope. The relevant compliance artifacts shall be obtained from AWS on a regular basis (every 6 months for SOC) from:

https://aws.amazon.com/compliance/soc-faqs/

Shared Responsibility

It is important to understand the AWS shared responsibility model (or that of any other cloud provider that one may choose to use). This is often represented in a RACI (responsible, accountable, consulted and informed) chart. AWS is responsible for the "security of the cloud" and the customer is responsible for security "in the cloud". There are various levels of cloud abstraction such as IaaS, PaaS, SaaS, etc. The further up we go the more the cloud provider is responsible for and the less we are responsible for. In this architecture we are going as far up that ladder as possible.

https://aws.amazon.com/compliance/shared-responsibility-model/

Software Production Process

We will be using a DevSecOps software production process using container based technology which is the basis of the Lambda serverless runtime. This is not only an efficient way to deploy software which allows rapid cycle times to get feedback from stakeholders to the developers quickly implemented and deployed but it also allows "shift left" of security. By implementing the correct tools in the CI/CD pipeline the developers can effectively be given certain security expertise via software which makes security more scalable and responsive.

https://aws.amazon.com/codepipeline/

https://aws.amazon.com/getting-started/hands-on/set-up-ci-cd-pipeline/

In addition to building/deploying software the pipeline shall implement certain security tools to cover the major risks.

Elastic Container Registry (ECR) implements container scanning for security vulnerabilities including in the libraries and executables used in building the container. This reduces the risk of having an exploitable vulnerability in our application which could compromise our CIA.

AWS CodeGuru is used in the pipeline as a form of Static Application Security Testing (SAST) to find poor programming practices which can lead to vulnerabilities.

https://aws.amazon.com/codeguru/

Amazon Macie is used to scan throughout the cloud environment as well as in the development pipeline to discover sensitive data such as secrets, passwords, keys, etc. and promptly alert the developer as to their presence so that the credential can be properly secured and rotated.

https://aws.amazon.com/macie/

Any secrets will be stored in Secrets Manager and never hard coded into applications or configuration files. Not only does this minimize expose of secrets but it allows for fast, easy, reliable rotation of secrets by changing the secret in SMS and then running the pipeline to re-deploy with the new secret.

https://aws.amazon.com/secrets-manager/

Developers shall have secure programming training and appropriate secure programming standards and policy. For example, the use of only parameterized SQL queries or an object relational mapper which parameterizes the queries automatically, automated vulnerability scanning, etc shall be specified by policy for all application development efforts.

Infrastructure as Code (IaC) tools such as Terraform will be used to configure the AWS environment.

Between the Terraform code, the containers forming the Lambda functions and the automated build and deploy pipelines we can safely and reliably iterate quickly to deploy new features, fix bugs, or fix vulnerabilities.

Security framework

Lacking any knowledge of the particular industry in which this web application operates which might imply some other framework, we have selected the NIST security framework as a good general-purpose framework. Being that HIPAA and FedRAMP are also ultimately based on NIST this would seem to be a good choice also just in case those become relevant.

NIST focuses on the “5 functions”:

https://www.nist.gov/cyberframework/online-learning/five-functions

You can read a more in-depth treatment of the 5 functions in the link above but here is a summary of how these will apply in our architecture:

Identify

We will identify software and data assets within the application architecture environment that we wish to protect.

We will identify how these things support the business.

We will identify appropriate policies to govern them.

We will identify vulnerabilities in our software, libraries, configuration, etc.

We will identify a risk management strategy for the environment including risk tolerances.

We will identify supply chain risk etc.

Note that without specifics and guidance from management it is not possible to get too far into the details of some of these things. But we do specify how we will identify vulnerabilities among other things and there is a brief discussion of supply chain risk in this document.

Protect

We will protect the environment by implementing appropriate access controls such as those established in the CIS benchmarks and other guidance such as that provided by NIST SP-800-171.

We will train our staff to protect the environment through security awareness and secure programming training.

We will protect the environment by implementing hardening measures from the appropriate CIS benchmarks and by implementing monitoring tools such as those provided by the cloud service provided.

Detect

We will detect security vulnerabilities and unusual activity using tools provided by the cloud service provider based on logs, scans of the environment, and scans run in the CI/CD pipeline.

We will detect anomalies through continuous security monitoring.

Respond

We will respond using sound incident response policy and runbooks.

We will respond where possible through security automation.

Recover

We will recover by ensuring that we can always revert back to a known good state.

This can be done via restoring data backups or by re-deploying known good code.

We will ensure that we can recover by regularly testing our recovery capabilities.

Additional guidance

In addition, there are the NIST 800 series of “Special Publications” which can be used to guide the security program. Of particular interest is NIST SP-800-171.

https://www.nist.gov/blogs/manufacturing-innovation-blog/what-nist-sp-800-171-and-who-needs-follow-it-0

This guidance is intended for commercial entities hosting confidential but not classified information. Particularly those who want to be able to service US Federal Government clients and contracts. But it is also a good basis for a security program for a commercial entity anywhere in the world even if we never intend to deal with the US government due to the solid risk management fundamentals it employs.

CIS Benchmarks

While the NIST guidelines describe how to run a security program in general they tend to be rather descriptive of how things should be rather than descriptive of exactly how things should be configured. This is where we like to bring in the technology specific subject matter expertise of another organization such as CIS. The CIS benchmarks provide a great resource for hardening certain technologies including many of the fundamental AWS technologies such as EC2, S3, RDS, among others. However, there are currently over 200 AWS services offered and they are not all covered in the CIS benchmarks. We will utilize the CIS benchmarks to harden the environment where they are available.

In this case we will use the “CIS Amazon Web Services Foundations Benchmark v1.5.0” benchmark which covers several of the AWS technologies which we are using.

There are benchmarks for other technologies which are no doubt used in this environment (likely things like docker, kubernetes, nginx, etc) but they are mostly under the control of the cloud service provider and it is their responsibility to harden those technologies. If our own applications are using a piece of software which is under our direct control (maybe we have nginx in a container) then we shall use the appropriate CIS benchmark to configure and harden it to the greatest extent practical.

The CIS benchmarks provide security controls applicable to all 5 phases of the NIST security framework.

In addition to the various specific AWS services defined in the application architecture below, the CIS benchmarks cover the proper configuration of Logging, Monitoring, and Networking in sections 3,4 and 5 respectively of “CIS Amazon Web Services Foundations Benchmark v1.5.0”
These security controls shall all be implemented wherever applicable.

The logging and monitoring specifically allow us to detect and respond to security incidents per NIST and the networking configuration hardening protects the environment.

Application Architecture

As mentioned previously, this application architecture is heavily dependent on AWS services. Each of the services below replaces a corresponding piece in the original architecture.

These services and configuration actions taken on them are logged through Cloudwatch and Cloudtrail services. Not only is this useful for analytics and debugging but this information is fed to Security Hub and is used for detection and response which align with these phases of the NIST framework.

- Cloudfront

The “front door” architecture and first line of defense will be Cloudfront which will feed requests to AWS WAF. Cloudfront will be the CDN to cache and route requests to S3 or elsewhere as appropriate.

https://aws.amazon.com/cloudfront/

    - WAF

The AWS WAF service can inspect the request and forward it to the API Gateway (described later).

https://aws.amazon.com/premiumsupport/knowledge-center/api-gateway-cloudfront-distribution/

AWS provides rulesets “out of the box” which cover things like the “OWASP Top Ten” and can also be customized with application specific knowledge. Given that we know what inputs to our various API endpoints are supposed to look like and consist of we can apply rules to filter out unexpected inputs which could be malicious. This arrangement proved very helpful in the Log4j attacks which came out in December 2021 and will no doubt come in handy in mitigating risk when similar vulnerabilities come out in the future.

This can be deployed as AWS WAF or WAF and Shield which is a DDoS prevention tool if that is of concern.

This aligns with the “prevent” step of the NIST framework.

https://aws.amazon.com/waf/

https://docs.aws.amazon.com/waf/latest/developerguide/aws-managed-rule-groups-list.html

    - S3

S3 will be used to host static assets and other objects. Publicly accessible S3 buckets can be fronted by Cloudfront. S3 itself is secured by AWS except for the permissions we assign the bucket. AWS Security Hub will help us to ensure that only the appropriate buckets are publicly accessible and will promptly alert if a bucket is made public which should not be.

The use of appropriate permissions and monitoring align with the “prevent” and “detect” phases of the NIST framework.

https://aws.amazon.com/s3/

Securing S3 is covered in section 2.1 of “CIS Amazon Web Services Foundations Benchmark v1.5.0”

- Route53

Route53 is the standard AWS DNS provider. It is already largely secured and monitored by AWS. We would want to monitor for unauthorized DNS changes via Cloudwatch and Security Hub.

This monitoring aligns with the “detect” phase of the NIST framework.

https://aws.amazon.com/route53/

- ALB

The ALB (Application Load Balancer) service allows us to load balance our many millions of incoming requests over an appropriate amount of backend infrastructure. While we could use the API Gateway service (described next) on its own and setup "Custom Domain Name" which effectively points the API Gateway to a Cloudfront Distro (internally) and then point the domain to that CNAME, the API Gateway by default has a soft limit of 10,000 Requests per Second (RPS) per AWS region per account. That is likely sufficient for most purposes but in the event we expected a flood of requests in excess of 10,000 RPS we would have the option of putting another region or account behind the ALB to service the requests. However, dividing the load up over more APIs, if possible, is suggested if we intend to hit these limits rather than additional regions or accounts.

However, and more significantly to security, it is not currently possible to put AWS WAF directly in front of API Gateway. It must be fronted with something else first such as ALB. Plus, without the ALB we would not have “ELB logging” (which is a component of the ALB) so we would lose valuable client details coming to the API such as IP, browser, etc.

The logging functionality of the ALB aligns with the “detect” phase and using the ALB helps us to enable WAF which aligns with the “prevent” phase of the NIST framework.

https://aws.amazon.com/elasticloadbalancing/application-load-balancer/

- API Gateway (write)

The API Gateway is a “serverless” service composed of AWS Lambda functions (or possibly other things, but most typically is a proxy service to AWS Lambda) which implement the functionality of the API. The API Gateway is auto-scaling which is critical to meeting the stated requirement of handling millions of requests.

This is where the majority of our own custom code and business functionality will live.

Rather than running on a traditional Linux server in a datacenter or even an EC2 instance in AWS, we run in an AWS managed environment. This means that we have no OS configuring or hardening or patching to be done. That is all the responsibility of the cloud service provider.

Downtime will be low for new deployments because all of the API functionality behind the API gateway can be updated via a rolling update which is automatically handled by the API gateway.

https://docs.aws.amazon.com/lambda/latest/dg/lambda-rolling-deployments.html

Applications can avoid sharing data by being implemented in separate API Gateway instances or Lambda functions and they can use different databases if we really want to avoid any mixing of data. For even more separation of applications they could even be deployed into separate AWS accounts. A tool such as AWS Control Tower could be used to manage separate logical accounts with security guardrails.

In a way, this choice aligns with the “Identify” step of the NIST framework in that we identified a risk (managing our own Linux systems) and avoided the risk by using API Gateway instead with its serverless nature. The same could be said of any of the technologies which avoid the risks we have identified by running the infrastructure ourselves and avoiding or transferring it to the cloud service provider.

https://aws.amazon.com/api-gateway/

    - RDS cluster (write master)
    - RDS cluster (read)
    - RDS cluster (standby)

The SQL database components of the architecture can all be provided by an RDS cluster. The SQL write master can be easily configured with an arbitrary number of read replicas to distribute read load.

The initial project requirements stated that the SQL database could be used for reporting. In this architecture we can easily spin up a read-only database dedicated to only reporting. If we are only interested in generating reports on last week or even yesterday's numbers and thus do not need up to date information we could even do a point-in-time restore of the database as of midnight last night and run reports on that so as not to have any of the production write load slowing down reporting. Then after we are finished generating all of our reports we can delete the database instance.

We can avoid sharing application data between RDS instances by deploying separate instances per application. With automation such as Terraform in AWS it is operationally easy to create separate database instances.

https://aws.amazon.com/free/database/

Securing RDS is covered in section 2.3 of "CIS Amazon Web Services Foundations Benchmark v1.5.0"

- API Gateway (read)

The read API gateway works just like the write API gateway except it is configured with the appropriate lambda functions to provide read only data and presumably queries the read replica SQL database which is implemented by the above RDS cluster.

https://aws.amazon.com/api-gateway/

- Elasticache

The API read service can access this memory cache to provide fast lookup responses. Elasticache is compatible with redis and memcached so depending on what was used originally it could be a drop-in replacement.

https://aws.amazon.com/elasticache/

- API Gateway (write, async)

This API Gateway for asynchronous writes works just like the above API write gateway except it feeds into an SQS queue which is read by a Lambda worker which can then put the data into a no-sql database (DynamoDB described below) or into S3 object storage.

https://aws.amazon.com/api-gateway/

- SQS

A “Simple Queuing Service” which is written to by the API Gateway for asynchronous writes. This is read by the Lambda worker described below.

https://aws.amazon.com/sqs/

- Lambda worker

This is a continuously running or scheduled process which takes items from the previously mentioned SQS queue to be put into either the nosql DynamoDB or into S3.

https://aws.amazon.com/lambda/

- DynamoDB

The nosql database which contains data placed by the lambda worker above. Note that depending on the nature of the data and what we intend to do with it, a different nosql may be a better choice as their capabilities and best use cases vary widely.

https://aws.amazon.com/dynamodb/

Security Hub
Cloudtrail which feeds into securityhub, like many other AWS security tools

- IAM

Finally, all of the above has access control via IAM. Roles such as those specified in the initial project specification can be defined such as admin, manager, user for working within the AWS architecture. Or, if these roles are intended to be defined within the application itself somehow, the AWS Cognito service could be used instead. They are related services and offer strong RBAC and authentication of users including MFA etc.

IAM is such a critical and fundamental piece of an AWS environment that proper use of IAM can be related to every phase of the NIST security framework.

Securing IAM is covered in section 1 of "CIS Amazon Web Services Foundations Benchmark v1.5.0"

Responding to attacks

All of these AWS tools produce log data which can supply Guard Duty with data which can be turned into actionable security information such as indicators of attack or indicators of compromise. Add Inspector to identify vulnerabilities and then you have a comprehensive set of security information being presented in Security Hub.

Security Hub is a cloud security posture management service that performs automated, continuous security best practice checks in your AWS environment. Security Hub aggregates security alerts from various AWS services.

https://aws.amazon.com/guardduty/

https://aws.amazon.com/inspector/

https://aws.amazon.com/security-hub/

There should be security policy and incident response plans which address various different scenarios. The actual details of those response plans will depend on the specifics of the environment and the security policy. But it is always best to have as many of these decisions made in advance so that premade runbooks can be employed quickly. It is best practice to create runbooks not only for the day to day operation of the environment but how to handle security incidents.

One available option in this environment is to use Security Hub functionality to implement automated response to security events via Lambda functions.

https://aws.amazon.com/blogs/security/automated-response-and-remediation-with-aws-security-hub/ (Note how this ties in with CIS benchmarks mentioned in this document)

This architecture contains very little mutable infrastructure for an attacker to affect. If someone gets a shell in a container then that container can be killed and redeployed.

It is important to have data backups and a "golden master" which is particularly well protected. The version control system where the code lives and the container registries where container/Lambda functions are deployed from are particularly critical.

Depending on how large/critical/valuable this infrastructure and business are, it may be worth considering a disaster recovery plan. This could either be backups in S3, backups in S3 in another account (to provide some small level of isolation), backups (of either just the data or possibly a whole other working infrastructure to fail over to) in another AWS region, or even in another service provider. It all depends on what is at stake and the risk tolerance of the organization.

Depending on how much data we are talking about and how critical it is, it would not be unwise to actually keep an offline copy somewhere which can be referred to in the very unlikely event that we suspect that the version control or container registry is compromised.

Long-Term Roadmap

To ensure the successful long-term operation and security of this platform we must remember the quote from security guru Bruce Schneier that “security is a process, not a product”. While we have involved many security products in this architecture, the process must be one of continuous security. The NIST guidance for operating the security program should continue to be followed.

Vulnerability management

New vulnerabilities are constantly being discovered. Some of these vulnerabilities will inevitably be in our code and infrastructure. We must constantly monitor for these and ensure that our tools continue to advise us of vulnerabilities. They must be patched within an SLA (service legal agreement) specified by policy to keep the risk to an acceptable level. While shift-left works great while active development

Security awareness

Staff will inevitably come and go from the team as time passes and lessons learned long ago eventually fade. All personnel involved in accessing or maintaining the application must undergo regular security awareness training. This includes secure programming training for developers as well.

Pipeline maintenance

Depending on just how mature this application becomes and how long-term we are talking about here, it would be a good idea to exercise the pipeline on a regular basis just to ensure everything continues to work and that we retain the ability to do rapid deployments. Being that the whole process is automated and consumes very little in terms of resources, a build and deployment should be conducted to a test environment on a regular basis. Possibly even once a day. This ensures that all pipeline tools get exercised, everything still works, and code hits the vulnerability scanners (which are presumably constantly updating their vulnerability databases) so that new vulnerabilities can be found and quickly remediated.

Continual review of attack surface

As the application changes and matures the initial security architecture may have to be revisited, particularly as the attack surface changes. New public facing APIs, IP addresses, services utilized which may need to be hardened, etc. should all be periodically reviewed. The use of tools such as Security Hub and others should be ensured long-term use to help with these tasks.

Change management and Change Advisory Board (CAB)

In addition to all changes to code going through a version control/source code management tool such as git, significant changes to the environment should be reviewed by a Change Advisory Board (CAB) on which security should have a seat. Often projects go through some sort of Architecture Review Board (ARB) when they are initiated but then they are handed off and forgotten about with developers left to their own devices. Significant changes in the environment (and “significant” should be defined by a policy) should require additional review by security personnel to ensure additional risk is not unknowingly introduced to the environment.

Ongoing legal review

Ongoing legal review of the cloud service provider’s compliance status and contractual obligations must be conducted. This should be done at least every 6 months in this case where we are relying on the cloud provider’s SOC 2 Type 3 status. Contracts should also be reviewed periodically to ensure that they contain the appropriate language ensuring continued SOC compliance and any other legal necessities.

Measure cyber security performance

Any security program should include some metrics to measure performance. Typical metrics should include things such as:

Number of builds
Number of deploys
Number of vulnerabilities detected
Number of vulnerabilities remediated
Number of vulnerabilities beyond SLA
Number of secrets detected (should be immediately remediated)

Among others depending on whatever is relevant and useful to measuring the performance and risk level of the environment.

Monitor for third party risk

There are a number of third parties involved including not only the cloud service provider but also the provider of other software, libraries, APIs, and services. Supply chain attacks are becoming more common. We should be vetting our suppliers, ensuring that their code is trusted, continues to be supported, and holding them to legal and contractual security requirements wherever possible. These things should be reviewed periodically, effectively being a supply chain risk assessment.

Communicate the state of the security of the platform upwards

Management should be kept informed of the ongoing long-term status of the security of the platform. Should there be any significant failures of any of the controls listed here they should be immediately communicated to management. Regular updates on the trends of the security metrics being measured should be communicated also. If any help is needed in terms of security controls, incident response, or trends in an unfavorable direction management should have the information they need to be able to step in to provide the needed support.

Conclusion

In this document we have laid out an alternative secure architecture which addresses the initial concerns of the independent security researcher. This has been accomplished by combining this security concern with the company’s desire to use cloud managed services to not only implement additional security controls but to also architect the application in such a way as to reduce the scope of the company’s security responsibilities and attack surface. The solution aligns with the NIST framework and also invokes additional security controls through CIS guidance including allowing the detection of and response to security incidents as called out in the initial project requirements. All while providing a scalable solution which meets the performance requirements.