Chapter 14
Security Architecture and Design

THE FOLLOWING CEH EXAM TOPICS ARE COVERED IN THIS CHAPTER:

- ✓ **Security architecture**
- ✓ **Service-oriented architecture**
- ✓ **N-tier application design**
- ✓ **Database structures**
- ✓ **Security models**

Developing strategies for an enterprise, including how to develop applications, takes a lot of consideration. There are many elements to think about, and you don't have to start from scratch. No matter what you are developing or implementing, it's important to consider users and, perhaps more important, data classifications and their necessary security requirements. Not all data is created equal, after all. Just as not all data is created equal, not all users are created equal. Because of that, when developing a security program, it's useful to have a security model. This means defining how you are going to allow users to interact with data and with each other. There are several ways of approaching the design of security models, and some of them relate to how data is classified.

Security models are a way to ensure that security policies are adhered to. Developing a security model is one way of defining behaviors to be implemented. This can help to guide application development and deployment as well. This may be especially true when it comes to defining architecture. There are different ways to implement applications today, especially as we move further away from a straight native application deployed on a single client system.

Additionally, there are security architectures to consider when it comes to application development and deployment. Developing a security architecture means using a top-down approach to ensuring that security policies, standards, and controls are all consistent and coherent. Security policy starts at the top of the organization, setting the direction based on business needs and requirements. It's important to keep in mind that security should be a business enabler.

Data Classification

Not all data is created equal. Data classification is an important step when organizing security systems and controls. Data classification is used to identify and organize information that has similar security control needs. This allows appropriate access controls to be created. When you start thinking about security models, it is important to consider your data classifications, because security models are used to help identify who gets to look at what data. You can't do this without first classifying and organizing your data resources.

Different organizations will use different types of classification schemes, depending on the data resources in place. This is also something you will hear a lot when it comes to the government and perhaps especially the military. Table 14.1 lists common classifications in use within the government.

TABLE 14.1 Governmental Data Classifications

Classification	**Description**
Top secret	The highest level of data classification. Only a limited number of people will be able to look at data classified as top secret.
Secret	The exposure of secret information would cause serious damage to national security.
Confidential	The exposure of confidential information would cause damage to national security.
Restricted	Exposure of restricted data would have undesirable effects.
Official	This is information that relates to government business and may not be an indicator of the potential for harm if the information were lost or exposed.
Unclassified	Unclassified information can be viewed by everyone. This may include declassified information that was once considered a higher classification, but the threat posed by its exposure has subsided.

Within each of these classification levels there may be additional compartments. Top secret information is not necessarily available to everyone who has been cleared at a top secret level. Just as not all data is created equal, not all people are created equal. Information sometimes needs to be compartmentalized even beyond these data classification levels. This is not the only way of categorizing data, of course. Organizations may have different ways of thinking about their information. Table 14.2 shows another way of classifying data.

You can see this is a much simpler way of categorizing data. Even simpler than this would be two classifications: restricted and public. Again, it depends entirely on the type of data that may exist within an organization. It may also depend on the people who exist within an organization as to how many levels of classification you need. If you have a small organization and everyone is essentially equal, such as a startup that consists of only partners in the company, perhaps you need to have only two data classification levels.

TABLE 14.2 Simple Data Classification

Classification	Description
Restricted	Data that is internal or sensitive. Loss or disclosure of this data could cause significant damage to an organization.
Private	Data that may also be internal or sensitive, but the loss or disclosure of this data would cause only moderate damage to an organization.
Public	Data that is classified public would incur no damage to an organization if it were disclosed or lost.

Classification should always be done based on business requirements and needs. As with so many things, it's quite a bit easier to perform classification early in the life of an organization. Larger organizations with a considerable amount of data will find it much harder to perform classification simply because of the volume and scale of the data. In some cases, regulations may suggest a classification scheme.

Security Models

Security models are used to help enforce access controls. A security model defines who can perform what action on data. It is an extension of the data classification levels that an organization has identified. For example, you wouldn't want to store public data mixed with sensitive data. People who should have access to public data generally shouldn't have access to sensitive information. This is not a property that goes in both directions, of course. Someone who has access to top secret data should have access to public data but definitely not the other way around. These are the sorts of relationships that are defined in the models presented in the following sections.

State Machine

The state machine model is based on a finite-state machine. A finite-state machine is a way of describing a set of computation steps. At any point in time, the machine is either at a single state or in transition between two states. This is an abstract way of evaluating computational models. You can see a simple state machine in Figure 14.1. It shows two states, state A and state B. In this model, it's possible to transition from state A to state B and from state B to state A. Depending on the model, it's not always possible to transition in both directions. Think of this as modeling an automatic door. The door has two states, open and closed. There are also transitions from open to closed and closed to open.

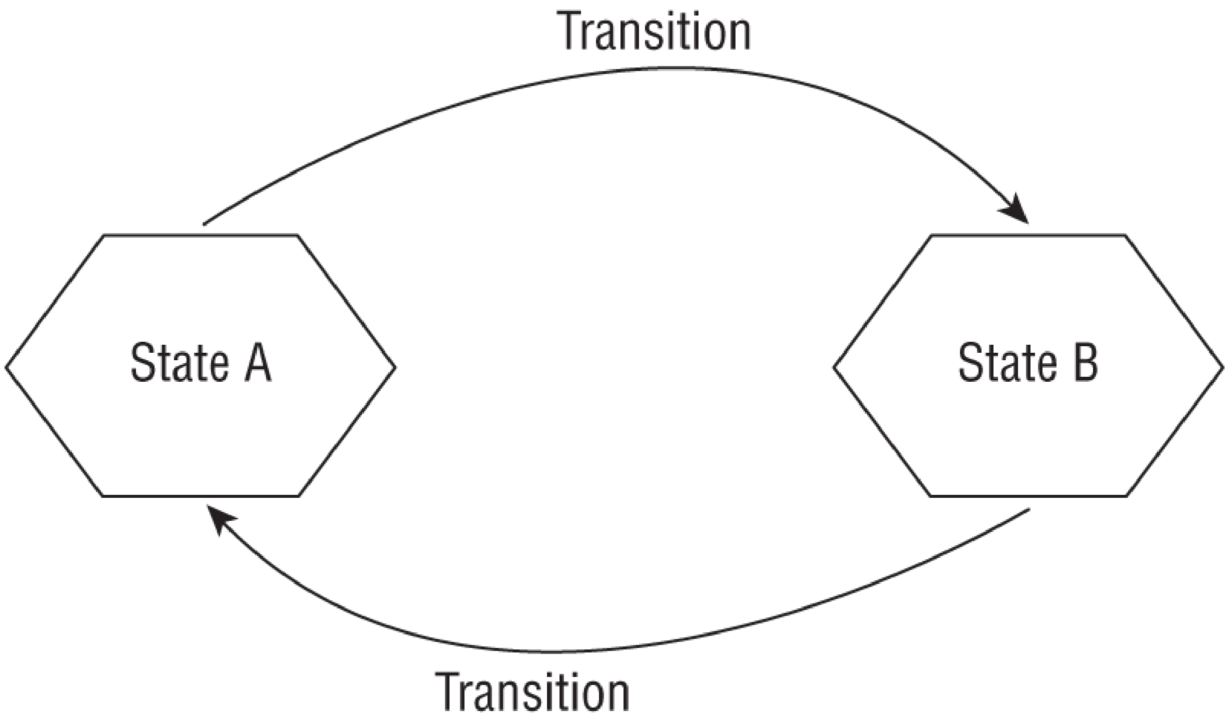


FIGURE 14.1 Basic state machine

A state machine model is used to identify when the overall security of a system has moved to a state that isn't secure. This requires that all possible states of the system have been identified. This should include the actions that would be possible to move a system into a particular state and all the possible state transitions. When a system makes a transition that is unexpected or an action results in an unexpected state, this can be monitored and alerted.

This model is far more abstract than the others discussed here since it doesn't actually define any expected behaviors. It's simply a way to model an environment, requiring that someone define expected behaviors and states.

Biba

The Biba model is named after the man who developed it in 1975, Kenneth Biba. The goal of the Biba model is data integrity. Because of this, this model may be referred to as the Biba integrity model. Keep in mind that security is not always about protecting the confidentiality of the data. The integrity of data is just as important. Data that has been altered unexpectedly or by the wrong people can also be damaging to an organization. This may be more complex than you would think. There are three objectives when it comes to ensuring data integrity:

- Unauthorized parties cannot modify data.
- Authorized parties cannot modify data without specific authorization.
- Data should be true and accurate, meaning it has both internal and external consistency.

The second one may appear to be confusing. Just because you have a login to a system or a network does not mean you have access to all data. The first objective is about ensuring that someone from the outside who doesn't have access can't modify data. The second is about someone from the inside not being able to modify data they shouldn't be modifying or perhaps shouldn't even be seeing.

In the Biba integrity model, data has classification levels, but people also have classification levels. These are also referred to as integrity levels. A user can only write to an integrity level at or below their own. The inverse of this may make more sense. A user can't write to an integrity level above their own. Write is not the only thing you can do to information. You have to factor in read as well. Reading can be done only at or above an integrity level. So, a quick way of thinking about this is read up, write down.

The Biba model defines three sets of rules:

- The Simple Identity Property says a subject at one level of integrity may not read a data object at a lower integrity level.
- The * (star) Identity Property says a subject at one level of integrity may not write to data objects at a higher level of integrity.

- The Invocation Property says a process from below may not request access at a higher level. The process can have access only at its own level or below.

Of course, not all security models are focused on data integrity. This means you may end up seeing some conflict with ideas presented from one model to another.

Bell–LaPadula

Bell–LaPadula is used in government or military implementations, and the intent is to protect confidentiality. Like Biba, it is a state machine model, meaning that subjects and objects exist in a single state at a point in time. Bell–LaPadula also identifies a *secure state* so all transitions should move from one secure state to another secure state. A secure state means that subjects are in compliance with objects as defined by security policy. Only appropriate levels of access are allowed within this model because they are defined and can be evaluated.

As noted, Bell–LaPadula is focused on confidentiality rather than integrity. As a result, the model defines properties that are different from those that were defined in the Biba model. The Bell–LaPadula properties are defined as follows:

- The Simple Security Property says that a subject at one security level may not read an object at a higher security level.
- The * (star) Property says that a subject at one security level may not write to an object at a lower security level.
- The Discretionary Security Property uses an access matrix to indicate discretionary access.

Bell–LaPadula provides for two mandatory access control rules (Simple Security Property and * Property) and a single discretionary access control rule. Mandatory access control is handled by the operating system. Rules are created based on policy, and they can't be manipulated by users of the system. Discretionary access control allows users to set their own access rules based on their needs and the data they control. With discretionary access control, the enterprise doesn't have control over what subjects are allowed to access objects.

Clark–Wilson Integrity Model

The Clark–Wilson model is another one that focuses on integrity rather than confidentiality. This doesn't mean, though, that it is the same as the Biba model, even if the end result is intended to be the same. Unlike Biba, Clark–Wilson does not rely on a state machine. It also doesn't make use of subjects and objects exclusively. Clark–Wilson adds in programs. Rather than a state machine, where a subject would act on an object directly, Clark–Wilson expects that subjects act on data objects only through the use of programs. Also, where Biba is used to protect information at higher levels of classification, Clark–Wilson is used and is applicable across all levels of data classification. The overall objective of Clark–Wilson is to maintain the consistency of the data as it moves from one state to another, meaning every transaction leaves the object in a state that makes sense and is usable.

Rather than defining a state machine, as noted, Clark–Wilson defines data items and only allows access through a small number of known programs. Essentially, the program handles the transaction on the object. The model relies on the subject, object, and access program to be known. If any subject other than those allowed attempt to access the object, even through the known program/transaction, it's a violation. The same is true for attempting transactions that are not defined by attempting to use other programs. This would also be considered a violation. Because all three elements are necessary, they are called Clark–Wilson triples.

In addition to triples, Clark–Wilson includes a set of rules. These rules require some additional nomenclature. There are Constrained Data Items (CDIs) and Unconstrained Data Items (UDIs). These are handled by Transformation Procedures (TPs), and the validity of the data in a certain state is checked by an Integrity Verification Procedure (IVP). There are nine rules specified by Clark–Wilson. Some of them are certification rules (CRs) and some are enforcement rules (ERs). Two of these rules are related to externality and integrity of the data, and these are CRs.

Three rules check to see whether TPs are allowed to act on a given CDI, and these are ERs. Another CR relates to making sure the entire triple is allowed. Additionally, there are rules for ensuring that there is logging and also for new data entering a system, specifying that it does not need to be constrained. Finally, there is a rule making sure that only the certifier of a TP can change qualifications of the TP.

Application Architecture

There was a time when applications ran on a single system. This goes back to the mainframe days when users would connect to the mainframe and run their programs there. Everything related to the program, including any data requirements the program may have had, existed on a single system. There were controls in place on that system to prevent unauthorized access, especially since there was a single point of entry to that system. Following along to the days of the personal computer, before they were attached to networks, programs ran on a single system. Data was also stored along with the program and the system. Once systems became connected to networks, possibilities were expanded. Data could be stored somewhere other than on the system where the program was running. Also, the user didn't even need to be on the same system where the program was running. Everything could be segmented.

Beyond the segmentation, there is also network geography to consider. At one point, everything may have lived on a company's network. This is no longer the case, since there are service providers that can offer computing resources. This makes application architecture possibilities very broad. Even the way applications are constructed has changed. We no longer have systems where applications run necessarily. Cloud-based providers, where systems reside at the service provider's premises and are consumed virtually, are starting to change how applications can be architected.

A classic application architecture is the n-tier design model. Application developers have been using this design for years. However, this is changing. A service-oriented architecture takes the idea of a monolithic design by breaking out components into services. The services then interact with each other. At the end of the day, the result is the same. The difference is how the data is handled and how the application ends up being constructed. Finally, especially when you think about the previous sections, there are database considerations. Even these considerations are starting to change.

n-tier Application Design

The n-tier design is a tiered application model. In some cases, you may see a version of this referred to as the model–view–controller (MVC) design, even if there may be other elements in those multiple tiers. These tiers are also referred to as the Presentation, Application, Business, and Data Access layers. You can see an example of this design in Figure 14.2. The Presentation layer, which is the same as the view, is at the client, where the laptop is in the figure. The next layer is the Application layer. This is where the logic resides. Programmable functionality lives in the Application layer. This may sometimes be broken up into the Application and Business Logic layers.

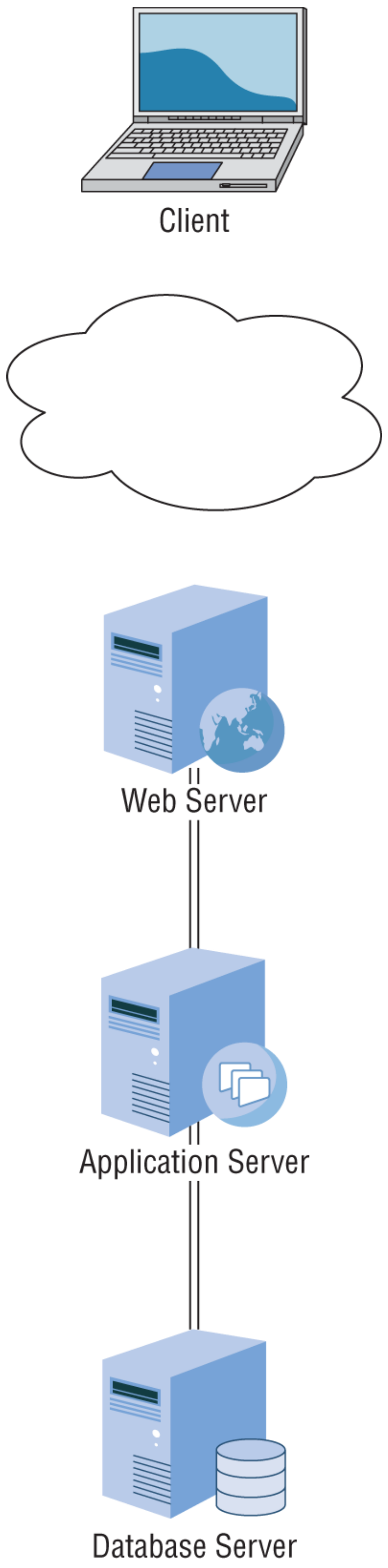
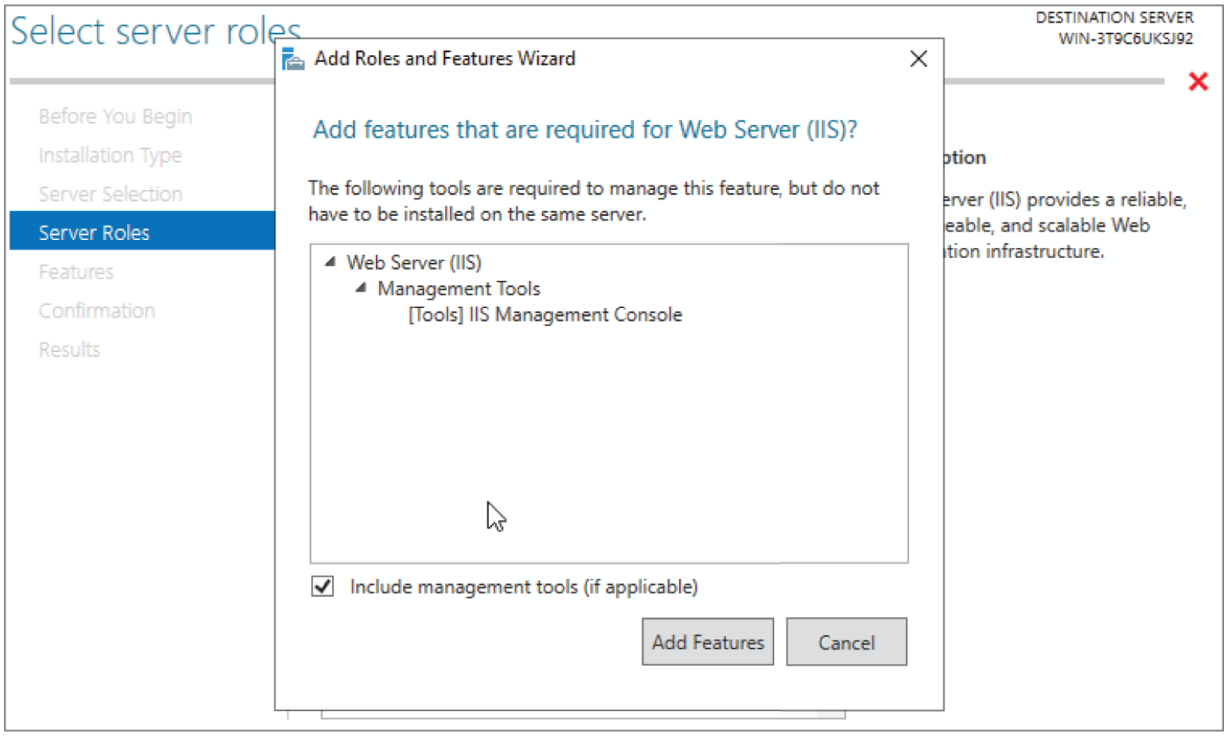

Client
Web Server
Application Server
Database Server

FIGURE 14.2 Multitier application design

In a four-tier model where the Application layer and the Business Logic layer are separated, the Application layer provides service control. In essence, the Application layer makes determinations as to which rules should be applicable and calls the appropriate business logic rules. The business logic rules are provided by the needs of the business and indicate how the application should process data that is provided to it. The business logic is what the application is all about. Everything else is a way of handling input and output. The Application layer, or the Application and Business Logic layers, are equivalent to the controller layer in the MVC design.

Application servers are generally language dependent. The application server provides a framework for applications to provide services to users. This is usually handled through web protocols like the HyperText Transport Protocol (HTTP). The application server may even provide the framework for a web server. Web requests are sent into the application server and from there are vectored into the application code as necessary, based on the request. Java is a common language used for application servers. Also, the .NET languages like Visual Basic and C# are handled in an application server, which comes with Microsoft's Internet Information Server (IIS). Figure 14.3 shows adding a role to a Windows Server to provide a web server. Windows Server 2022 comes with the .NET Framework bundled, so it is no longer necessary to install it as part of the IIS installation.


Select server roles

DESTINATION SERVER
WIN-3T9C6UKSJ92

Add Roles and Features Wizard

Add features that are required for Web Server (IIS)?

The following tools are required to manage this feature, but do not have to be installed on the same server.

```
◢ Web Server (IIS)
    ◢ Management Tools
          [Tools] IIS Management Console
```

☑ Include management tools (if applicable)

ption

erver (IIS) provides a reliable, eable, and scalable Web ation infrastructure.

FIGURE 14.3 IIS application server

The data layer is the model in the MVC design. It is the way data is organized and connected. Typically, this element is a database. Relational databases are commonly used as the backend for web applications. A relational database helps with the data model, since this type of database is about defining relationships between the different data objects. This definition, or schema, is a way of providing some visualization to the data model for the application. The Structured Query Language (SQL) is used to create the database and its component tables as well as add data and retrieve the data. Actions on the database are accomplished through the use of SQL statements, called queries.

Relational databases use an entity-relationship model. Entities are things like tables and fields in the tables. A table is a component of a database that defines a collection of related data. Think about the different data elements as columns in a spreadsheet. These columns describe the components of the table, and all the rows are the instances of data in the table. Using a relationship database, fields within one table can relate to fields within another table. You can see a simple example of this in Figure 14.4. This is called an *entity-relationship diagram*, and it shows the relationship between the tables in the database.

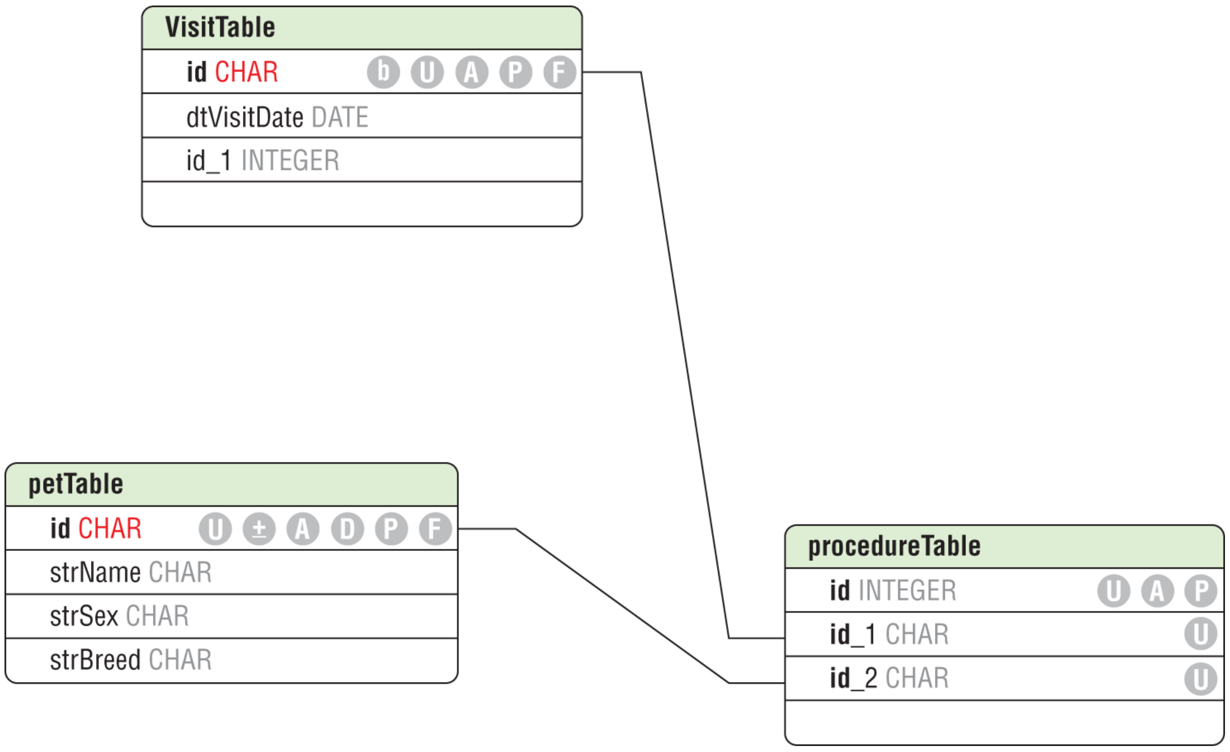


FIGURE 14.4 Entity-relationship diagram

These relationships keep associated data mapped together without needing to have a single enormous database table with a lot of repeated data. For example, in Figure 14.4, you can see that there is a table that defines a pet. This table has a unique identifier field, a name, a sex, and a breed. Then you have a visit table that includes the visit date. Since each pet will have multiple visits, you need a way to map those two rows together. This can be done in a third table. You include the key identifiers from the rows in the other two tables. This table keeps that mapping from the other two data structures. This entity-relationship diagram is a way of thinking about the model, and its implementation is the data layer of the multitier architecture.

Service-Oriented Architecture

A service-oriented architecture takes a different view of applications. Rather than thinking about applications from end to end (user to data store), it looks at the different functions needed to make applications function. These functions can be implemented as microservices. The architecture of the application is the connection between the different services as a way of implementing all the features required by the end user or the application consumer. These services are generally abstracted, meaning they operate as black boxes. The programmers who make use of a service may have no idea how it works internally. All they know is how to consume the service in a useful way.

This means the communications protocols are well defined. They may use communications protocols like remote method invocation (RMI), remote procedure call (RPC), or even representational state transfer (REST). These existing communications protocols mean the programmer doesn't need to create their own protocols to communicate from one service to another. The services make use of these protocols to communicate to other services through the use of libraries rather than needing to create entirely new communications methods.

Another advantage of service-oriented architectures is that once you abstract functions to services and allow the communication to happen outside of just calling a function within the same process space as the calling function, you can place that service anywhere. Fast networks and fast processors make this further abstraction possible. It also potentially limits the reach of a compromise. Once a service has been compromised, the only thing an attacker gets is access to that memory space. There may be nothing in that memory space or even that system other than ephemeral data that isn't even complete, depending on the implementation of the services and the overall design of the application. It does, though, mean that often services are exposed externally, so an attacker could probe from one service and system to another.

Figure 14.5 shows an example of an application design using a service-oriented architecture. At the very top of the diagram is the web interface. Applications don't always have a traditional interface any longer. It's often just as easy to use a web interface, and it's certainly often easier for users to consume a web interface rather than needing to install native-code applications to a system. Beneath the dividing line is a collection of services that could be consumed by any component of the application. These may include a login service, a user management service, a user profile service, a data modeling service, and a storage service. This means the application is really modular, which has long been considered a good approach to software design. When it's modular and abstract like this, any piece can easily be pulled out and replaced.

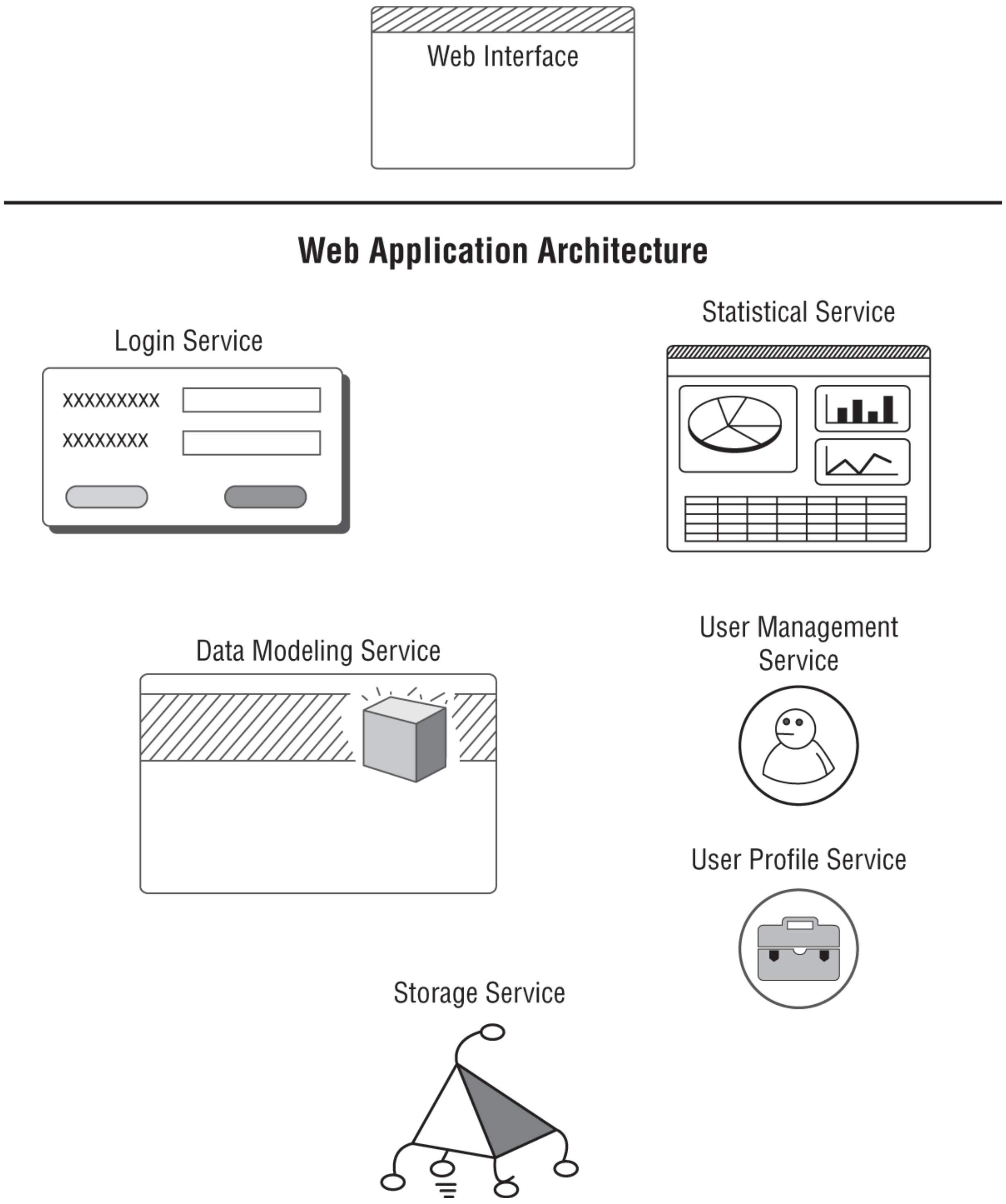


FIGURE 14.5 Service-oriented architecture

The idea of service-oriented architectures has been around for well over a decade. One reason its popularity has been increasing in recent years is the use of containers. A container is a way of isolating an application from other applications and services. This isolation is often done at the kernel level through the use of namespaces. A namespace tells the kernel what memory space any application is in. The kernel prevents any application from accessing the memory space or trying interprocess communication with an application from another namespace.

Unlike a virtual machine, which is a full operating system running on top of another operating system, a container uses the same kernel as the host operating system. This means you are not running a full operating system inside a container. The only thing that exists in the container is the application and any library dependencies that are necessary for the application or service to function correctly. This keeps the overhead associated with containers low. The container will also get its own network interface, so it does appear to other systems as though it's a separate device altogether. A container can be run inside a virtual machine. While it's technically still possible to run a virtual machine inside another virtual machine, there is overhead from the multiple layers of abstraction of interrupts and memory devirtualization. With modern computers, it may not be noticeable, but it's still overhead.

Containers alongside microservices or service-oriented architectures will change the deployment model for applications. There is no longer a need to stand up completely different systems, virtual or physical, when all you need to do is create containers for the different services a complete application comprises. Containers and microservices also make it easier to implement an elastic design, where services are started as they are needed by the clients rather than running all the time. This keeps costs down for the customer while also keeping applications responsive to clients.

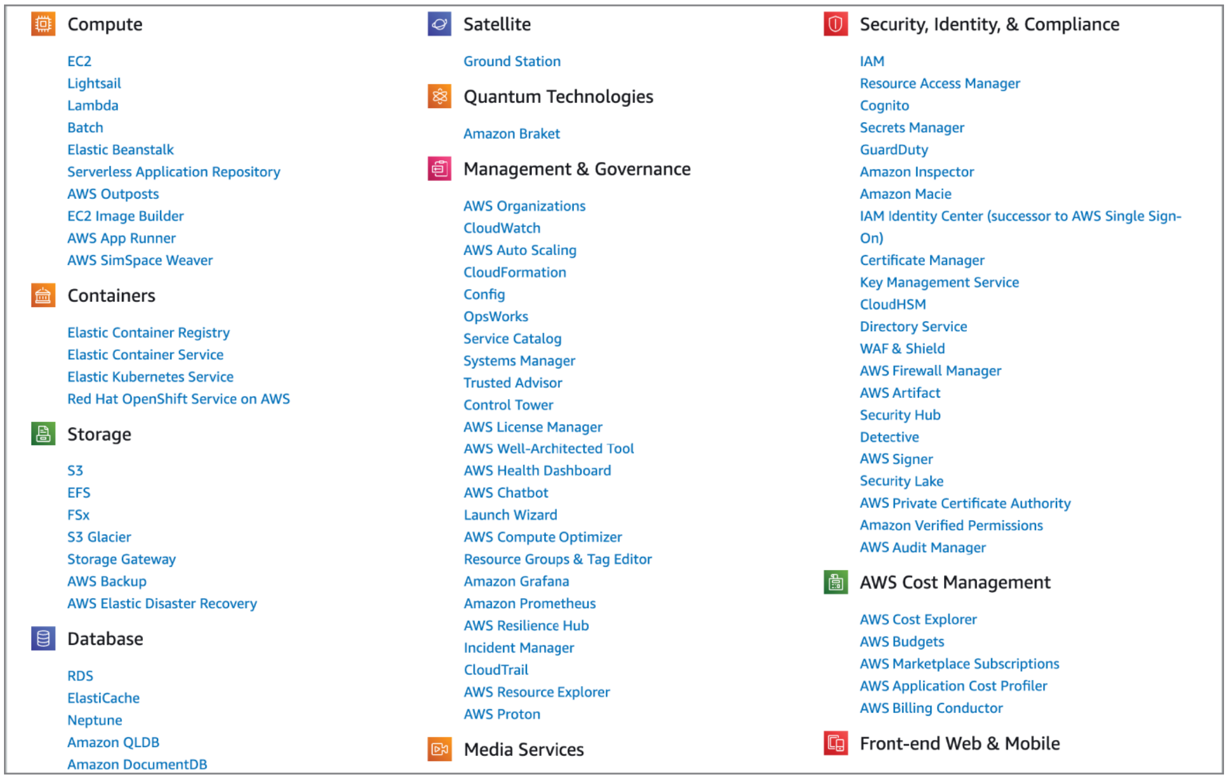
Cloud-Based Applications

In essence, using cloud computing providers is just another way of outsourcing infrastructure. This is not to say that's all a cloud provider offers. You can deploy a traditional multitier application at a cloud provider, and you would get a transference of the risk that may come from exposing a network-based application to attackers in the outside world. This exposure could allow an attacker to gain access to that application and then pivot to other systems within the enterprise. Moving to a cloud-based deployment can help alleviate that exposure.

There are better ways to make use of a cloud-based application model, though. For a start, cloud providers like Microsoft, Amazon, and Google allow for the creation of containers to deploy applications to. This means you don't have to worry about how you might deploy containers within your environment. Your cloud provider will take care of all the infrastructure for you. While containers are comparatively simple, they do require some management. The cloud provider handles the management. Figure 14.6 shows a list of many of the services Amazon Web Services (AWS) offers. One of them is the Elastic Kubernetes Service (EKS). Kubernetes is an orchestration service providing management of a collection of virtual machines and containers.

An advantage to using cloud providers is the array of security services you can take advantage of when architecting an application. There are log services, log watch services, and services to create a trail of events that can be useful in the case of the application being compromised. The event logs and available audit trails will be important for an investigation to determine point of entry and extent of compromise.

Additionally, providers like AWS will handle all identity and access management, making them considerably more capable than smaller organizations that don't have as many services at their disposal. Above and beyond what AWS offers natively, which is substantial, many vendors also offer their own services as AWS instances. The services may include unified threat management appliances, firewalls, intrusion detection systems, and many other types of security devices. Rather than hardware appliances, these are all virtual instances running the same software as a hardware device would.



Compute

EC2
Lightsail
Lambda
Batch
Elastic Beanstalk
Serverless Application Repository
AWS Outposts
EC2 Image Builder
AWS App Runner
AWS SimSpace Weaver

Containers

Elastic Container Registry
Elastic Container Service
Elastic Kubernetes Service
Red Hat OpenShift Service on AWS

Storage

S3
EFS
FSx
S3 Glacier
Storage Gateway
AWS Backup
AWS Elastic Disaster Recovery

Database

RDS
ElastiCache
Neptune
Amazon QLDB
Amazon DocumentDB

Satellite

Ground Station

Quantum Technologies

Amazon Braket

Management & Governance

AWS Organizations
CloudWatch
AWS Auto Scaling
CloudFormation
Config
OpsWorks
Service Catalog
Systems Manager
Trusted Advisor
Control Tower
AWS License Manager
AWS Well-Architected Tool
AWS Health Dashboard
AWS Chatbot
Launch Wizard
AWS Compute Optimizer
Resource Groups & Tag Editor
Amazon Grafana
Amazon Prometheus
AWS Resilience Hub
Incident Manager
CloudTrail
AWS Resource Explorer
AWS Proton

Media Services

Security, Identity, & Compliance

IAM
Resource Access Manager
Cognito
Secrets Manager
GuardDuty
Amazon Inspector
Amazon Macie
IAM Identity Center (successor to AWS Single Sign-On)
Certificate Manager
Key Management Service
CloudHSM
Directory Service
WAF & Shield
AWS Firewall Manager
AWS Artifact
Security Hub
Detective
AWS Signer
Security Lake
AWS Private Certificate Authority
Amazon Verified Permissions
AWS Audit Manager

AWS Cost Management

AWS Cost Explorer
AWS Budgets
AWS Marketplace Subscriptions
AWS Application Cost Profiler
AWS Billing Conductor

Front-end Web & Mobile

FIGURE 14.6 AWS service offerings

All of this is still focused on more traditional application design, however. Cloud services may also open the door to newer ways to conceive of applications. One of these ideas is going entirely serverless. What you may have noticed in Figure 14.6 is something called Lambda. This is a service offered by AWS that an application developer can use to create a function that has no server or container associated with it. If you string a number of serverless functions together, you can create an application. If, by chance, one of the functions has a vulnerability and it could be compromised, there is no server on the backend to take advantage of. Figure 14.7 shows a possible AWS architecture that may include no servers or even containers.

The architecture shows a user connecting to an API gateway. This provides the means to access functions that are provided by the Lambda services you can see in the middle. On the backend, there is a NoSQL database for application storage, including user-based state information. It is a simplistic design, of course. There are far more complex ways to design cloud-based architectures. There are also multiple ways to store data associated with the application. This may include storage on the client side using a virtual connection from the application network to the client network. However, there are also multiple ways to do file-based storage or databases with cloud service providers.

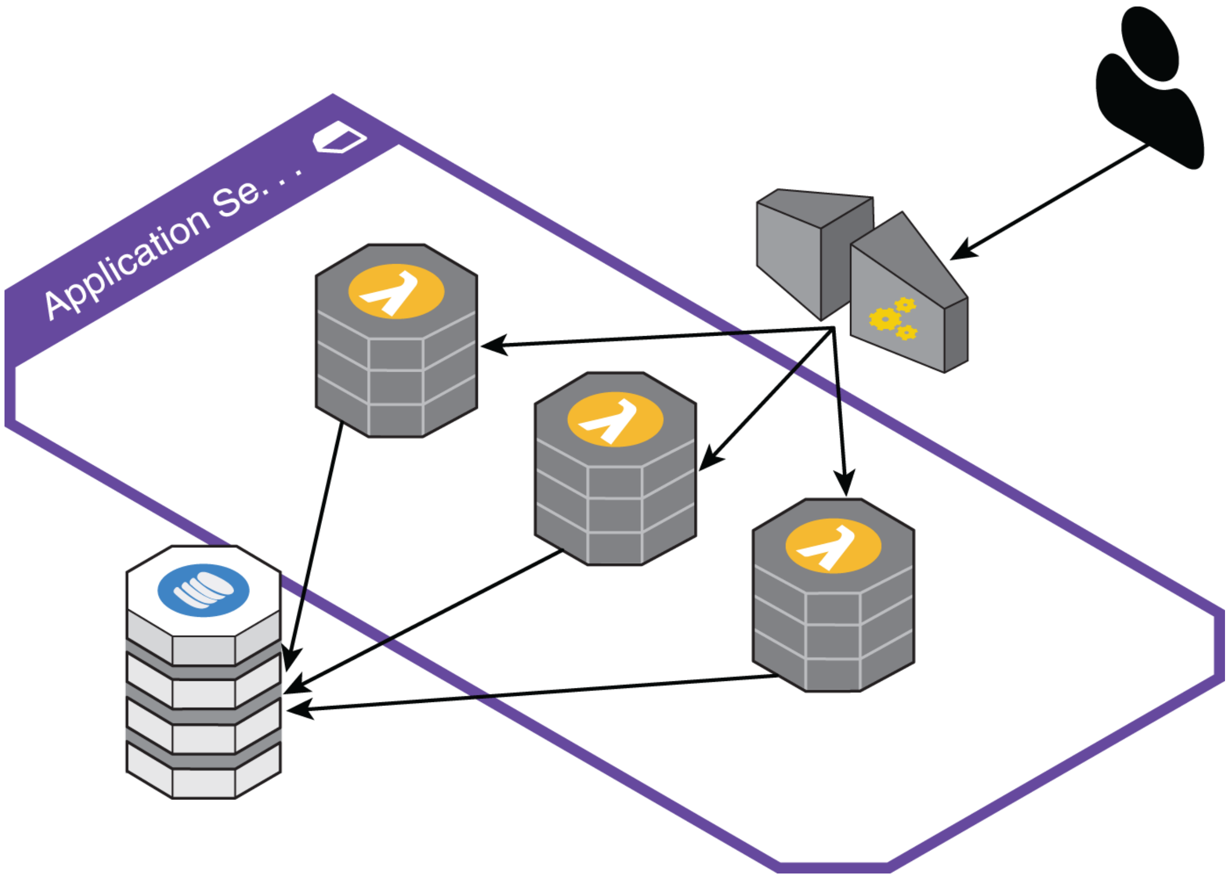


FIGURE 14.7 AWS serverless architecture

NOTE Most of the cloud providers offer serverless computing options. These are essentially ways for the consumer to provide application code without having to worry about where the code executes. The provider makes sure the code is executed at the right time, isolated not only from code from any other customer but also from any other instance of the code. These serverless computing models are generally event-driven, meaning the code gets called in response to an event within the application.

Database Considerations

Databases have been relational for decades. The language used to interact with a relational database is SQL, which was developed in the 1970s. There are a number of common SQL (relational) databases that you may run across. Oracle has long been a big name in SQL databases. Not only does it have its own enterprise-grade database, it also acquired MySQL, a popular open source database server, several years ago. MariaDB was forked from MySQL in 2010, before the Oracle acquisition. In some cases, MariaDB has taken the place of MySQL in Linux distributions. Microsoft's SQL Server is also a common database server, particularly in organizations that are already heavily invested in Microsoft infrastructure.

No matter which SQL server you are using, it is actually a server, meaning it is a service that runs, accepting connections. The connections can be over the local network, meaning there is an open port that listens for requests. Different servers will use different port numbers. For instance, MySQL uses TCP port 3306, while Microsoft's SQL Server uses port 1433. MySQL will not operate using UDP, though Microsoft's SQL Server can be configured to use UDP for connections from clients.

Listening services are problematic on the network because there is the possibility of an unauthorized client connecting. Strong authentication credentials are especially essential in cases where the connection has to be over the network because the client is on a separate server. It's possible, though, to use named pipes for clients on the same system as the server. This is a type of interprocess communication where there is no port listening for network connections. One process connects to another process using this named pipe.

NOTE SQL injection attacks are accomplished by injecting SQL into a web form, which may require some obfuscation. If an attacker can gain direct network access to the database server, they won't need to manipulate the query. If the attacker can get authenticated access, they can just issue queries and obtain the contents of the database.

While relational database servers have been common for decades, other database systems are starting to become more common. One of these database systems is NoSQL. This type of database typically doesn't use SQL for programmatic access. Beyond that, though, there are wide differences in NoSQL databases. One type of NoSQL database is key/value. You may think of this as a dictionary or an associative array. Both of these are terms you may find used in a programming language like Python, for instance. You might see this represented in JavaScript Object Notation (JSON). Here you can see how you might handle some personal information using JSON, which may be used to store or at least represent some NoSQL data.

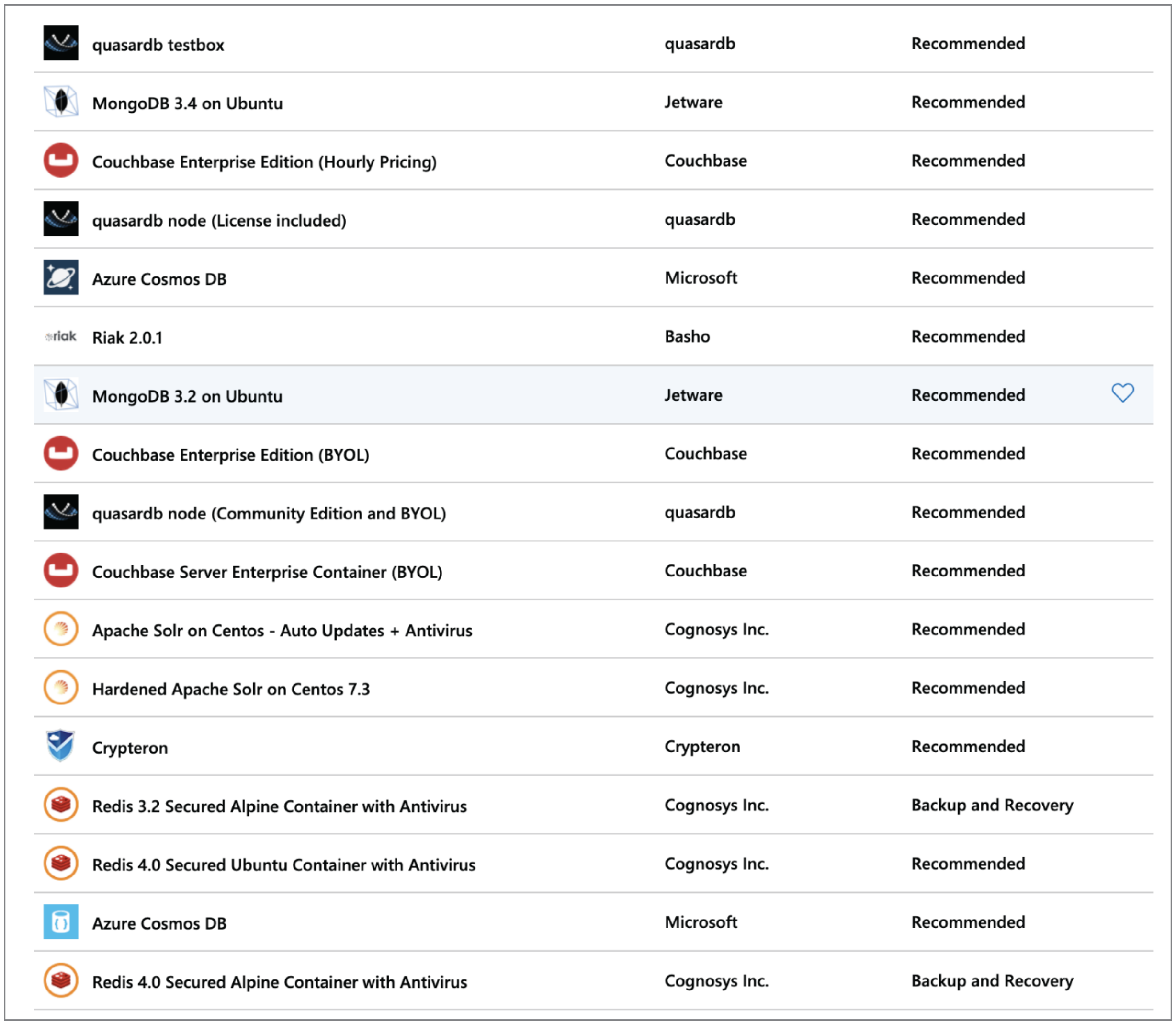
Person Representation in JSON

```
"person": {
    "firstname": "Ric",
    "lastname": "Messier",
    "sex": "Male",
    "age": "old",
    "state": "CO"
}
}
```

With JSON, you can embed complex datatypes. For example, at the very top level, the key is `person`, and the value is a multikey value. This is a complex datatype where the value of `person` is the complete set of data provided. You could have multiple `person` keys, each with different values associated with them. In a traditional relational database, each of the keys under `person` would be columns. You could then have multiple rows, where each of the columns in a row has values. With a key/value database, you aren't necessarily restricted to interacting with a single table at a time, and you can also make use of complex datatypes without having to do complex `JOIN` queries of a relational database.

There are other ways of thinking about or representing data using a NoSQL approach. You may use a document storage approach, where semistructured data can be stored. The data may be represented using JSON or the eXtensible Markup Language (XML), for example. Another type of database that may be used is a graph database, which is a highly connected data store where the connections are represented using graphs. Facebook, for instance, is known to use a graph database. Each type of database will impact how the databases themselves are used. The graph database can be really fast for searches, for instance.

None of this indicates how you would interact with them. Many of the database types are available with cloud-based providers like AWS and Azure. You can see a partial list of the NoSQL database providers that are available in Azure in Figure 14.8. Many of the database providers don't listen on the network by default. MongoDB needs to be configured to listen on a network interface, which may be just the local interface. Knowing the type of database an enterprise or application is using may be insufficient to gain access.



quasardb testbox	quasardb	Recommended
MongoDB 3.4 on Ubuntu	Jetware	Recommended
Couchbase Enterprise Edition (Hourly Pricing)	Couchbase	Recommended
quasardb node (License included)	quasardb	Recommended
Azure Cosmos DB	Microsoft	Recommended
Riak 2.0.1	Basho	Recommended
MongoDB 3.2 on Ubuntu	Jetware	Recommended
Couchbase Enterprise Edition (BYOL)	Couchbase	Recommended
quasardb node (Community Edition and BYOL)	quasardb	Recommended
Couchbase Server Enterprise Container (BYOL)	Couchbase	Recommended
Apache Solr on Centos - Auto Updates + Antivirus	Cognosys Inc.	Recommended
Hardened Apache Solr on Centos 7.3	Cognosys Inc.	Recommended
Crypteron	Crypteron	Recommended
Redis 3.2 Secured Alpine Container with Antivirus	Cognosys Inc.	Backup and Recovery
Redis 4.0 Secured Ubuntu Container with Antivirus	Cognosys Inc.	Recommended
Azure Cosmos DB	Microsoft	Recommended
Redis 4.0 Secured Alpine Container with Antivirus	Cognosys Inc.	Backup and Recovery

FIGURE 14.8 NoSQL database offerings with Azure

Not all databases are managed by separate pieces of software like a database server. In some cases, the database functionality is embedded into the application making use of the data. Such is the case in the Firefox web browser. All data used by Firefox, including bookmarks and settings, is stored in a SQLite database. Many other applications, perhaps especially mobile applications, make use of SQLite databases. They are stored as files. The application makes use of library functions to manage the data stored in the data files. As it's a SQL-based relational database, all programmatic access is still handled using SQL queries. To retrieve data, the program would generate a `SELECT` query. Storing data would be handled using an `INSERT INTO` query.

While NoSQL databases and applications using them for persistent storage are not invulnerable, moving to a NoSQL backend keeps SQL injection attacks from being successful. This means the attacks need to move into the application to get access to the database, rather than directly targeting the database server.

Security Architecture

Organizations should consider the use of a security architecture. This may be a confusing term. When some people hear *security architecture*, they may think about the network design and all the security elements that would be placed into the network. This, again, is a bit of a defense-in-depth approach to network design and thinking. Enterprises need to take a much broader approach to architecture. A security architecture should start at the top of an organization so goals and business objectives are identified. This ensures that all security functions within the organization are designed and implemented to support business goals. An architecture should identify the analysis, design, planning, and implementation of security functions.

There was a time when security was more of a siloed function, standing off to the side. A security organization may have had to stand on its own and create its own requirements. This was a result of a general lack of understanding of the value and functionality of security and its importance to the organization as a whole. In the absence of business-generated goals, the goals of the security organization were developed from the bottom up, which means the technical needs may have been put in front of anything else. As data breaches have become far more prevalent over time, businesses have started to recognize the need to take security on as a function of not only executive management but also the board of directors.

To achieve this top-down defined security architecture, the business should take advantage of any business road maps or long-term planning already in place. They should also make use of trends, whether they are industry trends or intelligence related to threats that may be specific to the business or the vertical industry the business belongs to.

The most important factor for businesses to consider is how they handle risk. Risk is the probability of the actualization of loss or damage resulting from the exploit of a vulnerability. Businesses may have different perspectives on risk, based on how they have identified threats or quantified loss or probability of that loss. Overall, the business needs to manage risks to the business, and businesses have become more aware of risks associated with their information resources. Once a business has a handle on risk identification and management as well as the planning that would go into risk management, it would be able to generate requirements that should go into a security architecture.

A business should have a methodology to follow that provides additional guidance. This could be handled by adopting a framework to use. The National Institute of Standards and Technology (NIST) has a Cybersecurity Framework that has been identified to highlight phases in which a business should consider implementing security controls. The phases or categories identified by NIST are identify, protect, detect, respond, and recover. NIST refers to these as the *five functions*, and they are the core of the NIST Cybersecurity Framework. You can see the five functions depicted in Figure 14.9. You'll note that they are in a circle. This is because each function has inputs into the next. The cycle also never ends.

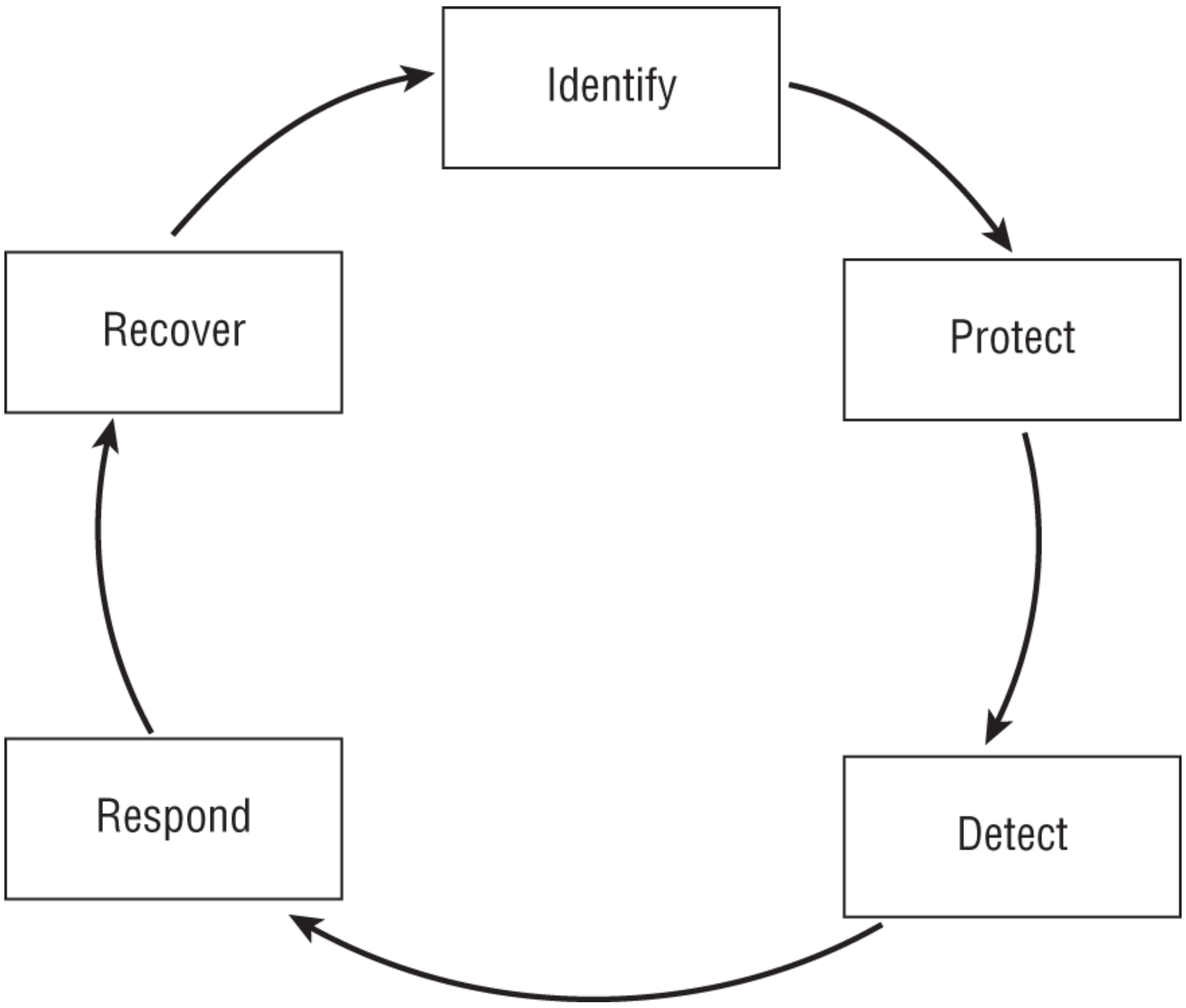


FIGURE 14.9 NIST's five functions

A security control is a means of avoiding, detecting, counteracting, or minimizing security risk. Security controls may include firewalls or intrusion detection systems, for example. They may also include door locks or bollards for physical security. There are a lot of categories that security controls may fall into.

Since we are talking about security at a business level, these five functions refer to risks to the business. The Identify function, for instance, is about identifying risk to the business, identifying assets, identifying policies used for governance, and identifying a risk management strategy. These actions should be guided by the business to ensure that security and the business are aligned in a mutual understanding of the goals. All of these actions are necessary to provide the information that may be required for the other functions.

The Protect function can't be executed without identifying all of the business assets and their importance to the organization. Protect isn't just about making sure there are controls in place to keep attackers out. This function is about protecting business assets overall, which means there should be capabilities for maintaining software or appliances. This includes ensuring that there is a plan to keep all assets up-to-date. It also includes making sure there is an identity and access management function to ensure that only authorized users gain access to business resources. Business resources include physical resources like buildings and any areas within the building that may house business-critical assets, informational or otherwise.

We don't expect that protection will be perfect. There is no way to keep adversaries out while still remaining operational. This is why it's necessary to have detection capabilities. Detection isn't only about identifying anomalies and events, however. It's necessary to understand the impact of these events. This also goes back to the Identify function. Part of the Identify function should be classifying and prioritizing business assets. This will help teams to understand the impact. If an event is detected, it can be assessed for veracity as well as impact and then prioritized based on those pieces of information (as well as any others the business decides are necessary for prioritization).

Once an event is detected, it should be investigated. If it's determined to be of limited to no impact, it may just be recorded. If the event does pose a potential risk (measurable probability of loss or damage), it would be classified as an incident, which would call the Respond function into action. The Respond function is about isolating the incident and applying mitigation steps to resolve the incident. The Respond function, though, is used not only when an incident occurs. Response requires preparation. Businesses need to have a Respond function that will have plans in place for different contingencies, so they are ready if something were to happen. This function would also require the need to communicate with stakeholders, which would include internal stakeholders as well as, potentially, law enforcement or other external entities.

Finally, the Recover function ensures that normal business operations are restored. It should also include capabilities to ensure that a similar incident doesn't happen again. This may include the capability to hold after-action reviews to collect lessons learned and then implement those lessons learned to continue to improve business practices. This is one reason the five functions may be depicted to be a circle. There should always be feedback loops, where one function informs another, so all functions continue to improve over time.

This may all seem overwhelming. It may be especially true when you consider all of the security controls that are necessary to be implemented to cover all five functions. Fortunately, NIST has some help here. NIST's Special Publication 800-53 is a catalog of security controls NIST recommends for use in the federal government. These same controls can also be used by any organization. They can be used to guide implementation of a security architecture whose goal is to provide support and protection to the business.

NIST is not the only player in the space of providing guidance that can be useful in creating a security architecture. There are several others as well. One is the International Organization for Standardization (ISO). It publishes guidance for information security management systems that is called ISO 27001 (sometimes referred to as ISO/IEC 27001). It provides another way to think about implementing information security systems. ISO 27001 has a simpler cycle than NIST provides with its Cybersecurity Framework. The cycle for ISO 27001 calls for the following: Plan, Do, Check, and Act. These are all pretty self-explanatory, except perhaps Act, which is about addressing anything that may come out of the Check phase, which could include implementing preventive or corrective actions resulting from an audit.

One last way to think about implementing information security controls is through the attack lifecycle. The attack life cycle is thought to be the set of steps an attacker would take to compromise your organization. You can see the attack life cycle in Figure 14.10. The attack life cycle is a concept developed by Mandiant not only to help guide its incident response capabilities but also to inform companies about steps the adversary will take so the organization may be able to better detect what is happening and, as a result, identify the severity of the detected incident. An event indicating initial recon would be far less severe than an event indicating data exfiltration (mission complete).

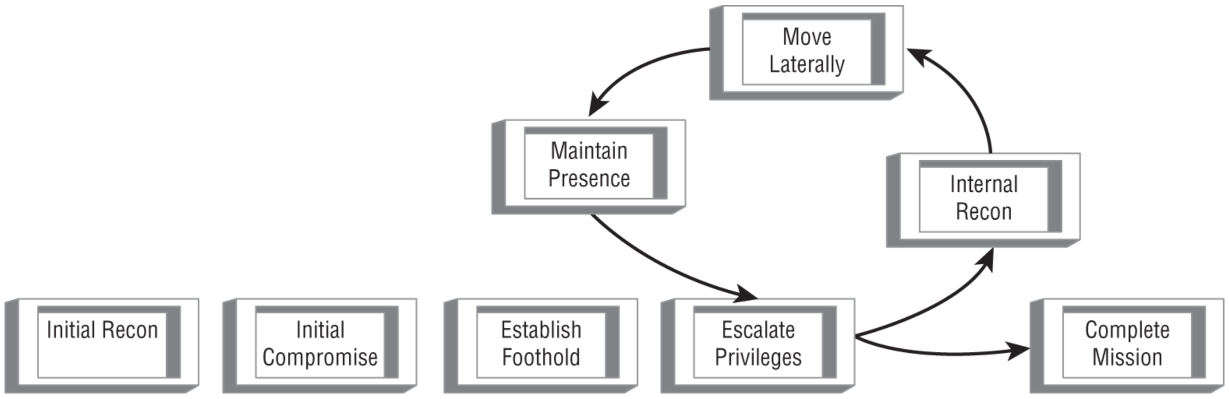


FIGURE 14.10 Attack life cycle

Businesses looking to implement a security architecture may consider implementing controls to address each of the phases in the attack life cycle. They will want to not only protect against but also detect initial recon as well as initial compromise. Some companies may stop here, assuming they will prevent or catch everything. Since the most important actions come later in the attack life cycle, it's important to be able to manage anything happening there as well.

Another way of looking at how attackers operate is the Lockheed Martin Cyber Kill Chain. This model of attack identifies how attackers achieve their objectives. This is a seven-step process that was identified by Lockheed Martin. The process is as follows:

1. **Reconnaissance:** This phase is where the attacker identifies email addresses that may be used for targeting.
2. **Weaponization:** A payload is developed in this phase, including a backdoor that can allow the attacker to get access.
3. **Delivery:** The payload is delivered to the target, via either an email, a website, a USB stick, or some other mechanism.

4. **Exploitation:** This involves exploiting a vulnerability on the target system to execute code.
5. **Installation:** This involves installing malware on the target.
6. **Command and Control (C2):** This involves making use of a command channel to the target system to control it.
7. **Actions on Objectives:** The attacker now has local access to the system and can achieve their original goals.

Zero-Trust Model

The COVID pandemic may have been one of the primary motivations for many organizations to move to a zero-trust model for their overall security architecture. The main reason for this is the need to consider a different perimeter for the enterprise. Traditionally, the perimeter of an enterprise was defined by the ingress and egress points, protected by firewalls. All resources were maintained within the enterprise. Even in cases where clients were outside, there were protected ways, like virtual private networks (VPNs) or even virtual desktop infrastructure (VDI), so these clients could appear to be on the inside of the network even if they were physically outside. The perimeter was maintained since all business communications were contained within the enterprise perimeter.

Now, knowledge workers are often working from home, which puts a strain on network connections, as well as the VPN or VDI systems in use. At the same time, many businesses have been moving to more cloud-based infrastructure, whether it was for their messaging with Office 365 or G Suite or whether it was software-based solutions, including Workday, SAP, and countless others. This meant business data was no longer contained within the physical environment of the business. Figure 14.11 shows clients connecting to multiple cloud-based environments from their home locations, or wherever they happened to be.

The location of controls for businesses has had to change. There may not be a need to have all client traffic come back to the enterprise network if most of the traffic is really going to Microsoft, SAP, Google, or some other provider somewhere. Instead, the connections can go directly from the home office to these cloud-based providers. That still potentially exposes the business resources to compromise if the accounts are compromised or the network connection is not protected in other ways.

The zero-trust approach, documented in much more detail in NIST 800-207, identifies business resources and users of those resources and specifies that you can't expect location to be a factor when it comes to trust. This has commonly been an approach in how businesses protect resources. Systems that are inside a network are trusted, while systems outside may not be trusted. In zero trust, the expectation is systems or locations are not factors in whether resources can be accessed. Sometimes, for instance, devices themselves may be able to authenticate by offering certificates that are specific to the device. Instead of trusting systems or locations, every resource access must be authenticated.

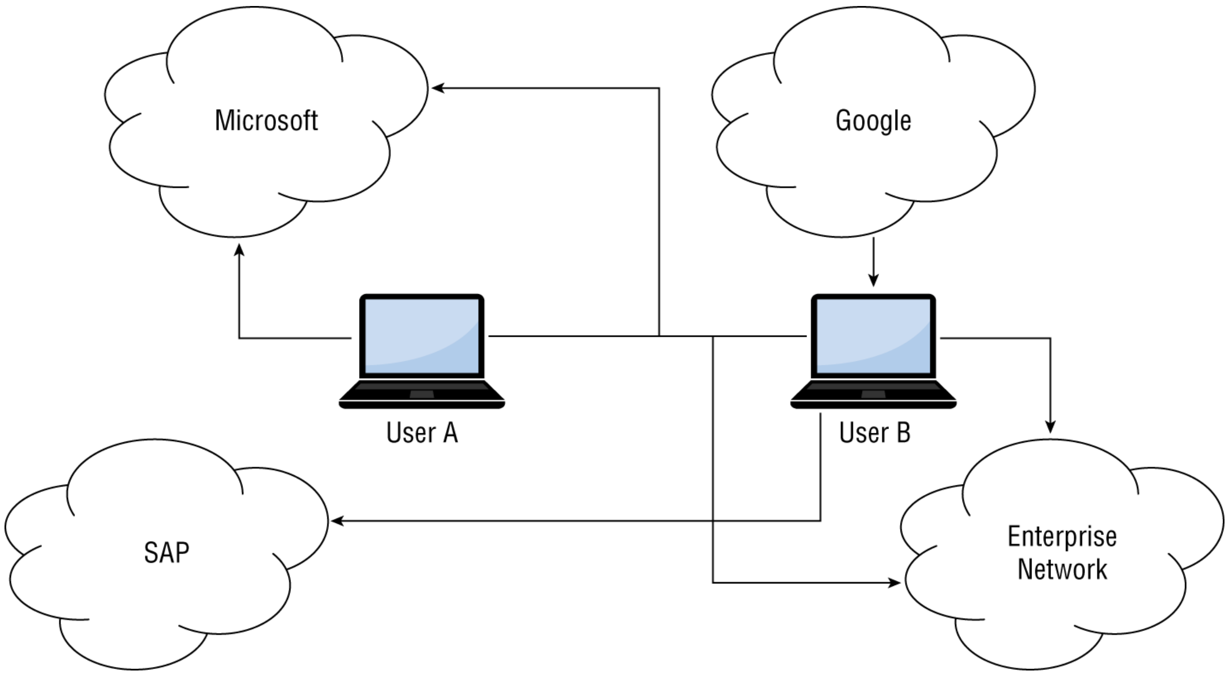


FIGURE 14.11 Cloud-based business communications

This approach requires a strong identity and access management (IAM) system. This is a centralized approach to controlling user accounts and applying necessary policies to protect against account compromise. Additionally, accounts need to be protected with multifactor authentication (MFA). MFA helps to protect unauthorized access by requiring users to not only provide usernames and passwords but an additional factor as well. This may be a physical token, as in a YubiKey, that is possessed by the user.

It may also be a one-time password provided by a device or application that is only on a device owned by the user. Often, this is someone's phone, though physical tokens that generate these one-time passwords may also be used.

NOTE MFA is not a perfect protection against account compromise, though. MFA-bypass techniques include the attacker continuing to have requests sent to the device, hoping to get an approval. This is possible if the application allows push requests, meaning a request comes to the device or application and the request has to be approved. If short-messaging service (SMS) is used as an additional factor, there are ways for attackers to get access to those as well, meaning they are vulnerable.

Moving the clients outside the control of the enterprise also changes the location of protections. In the case where clients are all inside the network, detecting malicious behavior was easier because the enterprise had control over all the network communications infrastructure so it could be monitored. That is no longer true when clients are on their home networks. While the clients are generally under the control of the enterprise, the networks they are on are not. This pushes monitoring and other detective controls out of the network, if they were there, and onto the endpoint.

While endpoint detection and response (EDR) applications may be a common way to address this, since they are useful whether the endpoint is on the enterprise network or at home, they are not the only way. What an EDR solution does not offer is a guarantee that all communications, no matter where they are to, are protected. The EDR solution may also not know as much about the systems the endpoints are trying to communicate with, including malicious command and control systems on the Internet. This may require additional controls.

Finally, another consideration for moving the perimeter out of the physical enterprise is management of the endpoints. Monitoring, updates, and many other management applications are often on-premise, so the communication between those on-premise management systems and the endpoints needs to be considered, again with the zero-trust model being considered, especially since these essential tasks may be happening in the open on public networks rather than completely under the control of the business.

Summary

Data classification is an essential activity. It helps to identify all data resources as well as prioritize their sensitivity or importance. This action is needed in order to implement a security model. You would be unable to implement Biba, for instance, if you didn't know sensitivity or priority, since Biba needs to know who can read up and who can write down. The Biba security model is about data integrity. The same is true for the Clark–Wilson integrity model. Other models, like the Bell–LaPadula model, aren't as concerned with integrity as they are with confidentiality. Integrity ensures that data isn't altered or corrupted by unauthorized users. Confidentiality ensures that data isn't seen by those who are not authorized. Security models are necessary for implementing mandatory access controls.

Applications are designed. As a result, they generally follow known architecture or design patterns. Native applications that are stand-alone have no external architecture but may have an internal architecture. This means, since they are stand-alone, they don't rely on other systems or services. A common application architecture, though, is the n-tier, or multitier, architecture. The multitier architecture is composed of the Presentation, Application, Business, and Data Access layers. Sometimes the Application and Business layers are consolidated to just handle logic for the application, based on business requirements. This would be a three-tier application. This is an implementation of an MVC application design.

Web applications will generally use a multitier architecture. The browser is the Presentation layer (view). There is likely an application server, whether it's running Java, .NET, PHP, or some other language, that handles business and application logic (controller). Finally, there is probably a datastore, perhaps in the form of a database, that is the Data Access layer (model).

Modern applications are still using multitier architectures, but they are also often broken up into functions or services. When an application is viewed or designed this way, it is said to have a service-oriented architecture (SOA). This means the overall application is broken up into services, and the services interact with one another. It provides modularity so any service can be replaced with another service with the same input/output specifications without altering the rest of the application. Recently, this approach has been adapted into a microservice architecture. Microservices are further enabled through the use of containers like Docker or Kubernetes.

Sometimes, these containers are implemented through the use of a cloud provider. Traditional application architectures can also be implemented using a cloud provider, and you could end up with a hybrid approach where pieces of your application are on your premises while others are implemented using a cloud provider. Cloud providers are also beginning to expose application developers to new ways of considering their application design. This includes such things as serverless functions. The functions are connected in order to create the overall application, but there is no server underneath that an attacker could gain access to. Similarly, the use of containers has sometimes led to automated infrastructure, so containers and virtual machines are built up and torn down on demand. An attacker who gained access to such an environment might have to keep starting over when the system they had access to suddenly went away, including any files that had been put in place by the attacker.

Often, applications need to store data. It may be temporary data, or it may be persistent data. Traditionally, application data has been stored in a relational database accessed using SQL. Modern applications are moving away from this approach and starting to use NoSQL databases, which may use semistructured documents or key-value associative arrays.

Businesses in general need to think about a security architecture. This is not related to application or even network design or architecture. Instead, it is a set of data and methodologies that guide the overall implementation of security within the organization. NIST recommends the five functions of identify, protect, detect, respond, and recover as a way of guiding the organization—organizationally for staffing, but also in terms of how they evaluate information security and any potential risks to the business.

NIST is not the only organization that has security recommendations. ISO 27001 is another set of recommendations for information security management systems. They recommend Plan, Do, Check, and Act. There is also the attack life cycle that identifies phases an adversary works through to gain access to critical business systems or data. These are initial recon, initial compromise, establish foothold, escalate privileges, internal recon, move laterally, maintain persistence, and complete mission.

NIST also has guidance for the implementation of zero trust, which is an approach to ensure stronger authentication is used across all business resources. Too often, businesses trust what is physically under their control. Zero trust removes that approach and guarantees all accesses are authenticated with strong authentication that uses multifactor authentication.

Review Questions

You can find the answers in the appendix.

1. Which of the security triad properties does the Biba security model relate to?
    - A. Confidentiality
    - B. Integrity
    - C. Availability
    - D. All of them
2. How many tiers are there in an n-tier application design?
    - A. Two
    - B. Three
    - C. Four
    - D. Depends on the application
3. What type of database may JSON be most likely to represent?
    - A. Relational
    - B. SQL
    - C. Key-value

    D. Document-based

4. How many functions are specified by NIST's cybersecurity framework?

    A. None

    B. Three

    C. Five

    D. Four

5. How many steps are there in the ISO 27001 cycle?

    A. Two

    B. Three

    C. Four

    D. Five

6. What is the highest level of classification used by the U.S. government?

    A. Top secret

    B. Confidential

    C. Restricted

    D. Eyes only

7. Which of these is microservices a specific implementation of?

    A. Service object architecture

    B. Micro Channel architecture

    C. Microservices architecture

    D. Service-oriented architecture

8. What is an application referred to if it is only using AWS Lambda functions?

    A. Service-oriented

    B. Virtualized

    C. Serverless

    D. Infrastructure as a service

9. What does the Clark–Wilson model use to refer to objects?

    A. UTI and CDI

    B. CDI and CTI

    C. UDI and CDI

    D. UTI and UDI

10. What type of application virtualization would you use without going all the way to using a hypervisor?

    A. Emulation

    B. AWS

    C. Paravirtualization

    D. Containers

11. What is the first function specified by NIST in its Cybersecurity Framework?

    A. Identify

    B. Protect

    C. Risk management

    D. Defend

12. What is a common middle tier in an n-tier application design?

    A. Web server

    B. Database server

    C. Logic server

    D. Application server

13. What is a common open source relational database server that may be used in web applications?

    A. MongoDB

B. MySQL

C. SQL

D. Oracle

14. Which of the following is true about the Bell–LaPadula Simple Security Property?

    A. A subject cannot write up to an object.

    B. A subject cannot write down to an object.

    C. A subject cannot read up to an object.

    D. A subject cannot read down to an object.

15. What are the phases of the ISO 27001 cycle?

    A. Plan, Identify, Act, Detect

    B. Plan, Detect, Act, Do

    C. Act, Do, Identify, Play

    D. Plan, Do, Check, Act

16. What is an essential element of a zero-trust architecture?

    A. Cloud-based applications

    B. Multifactor authentication

    C. Virtual private networks

    D. Virtual desktop interfaces

17. What type of processing does serverless computing typically use?

    A. Parallel

    B. Procedural

    C. Event-driven

    D. Functional

18. Which of these would be the best approach to implementing multifactor authentication in a zero-trust model?

    A. SMS

    B. Push-based approval
    C. Username/password
    D. OTP

19. What is the Cyber Kill Chain used for?
    A. Identifying attackers
    B. Defining monitoring actions
    C. Describing an attack process
    D. Developing command and control systems

20. What software may be helpful in keeping workstations protected in a zero-trust design if they are not directly on the enterprise network?
    A. Syslog
    B. EDR
    C. SAP
    D. SCCM