Microsoft Azure

Secure DevOps: Application Security Principles and Practices

Application Security Principles

Your name
Your Title
Microsoft

Module Overview

- Secure by design
- Input and output handling
- Error handling
- Usable security
- Authentication and authorization
- Outdated components
- Auditing, alerting and monitoring
- Lab – Broken Web App

Microsoft Azure

Secure by design

Security begins with Secure Design

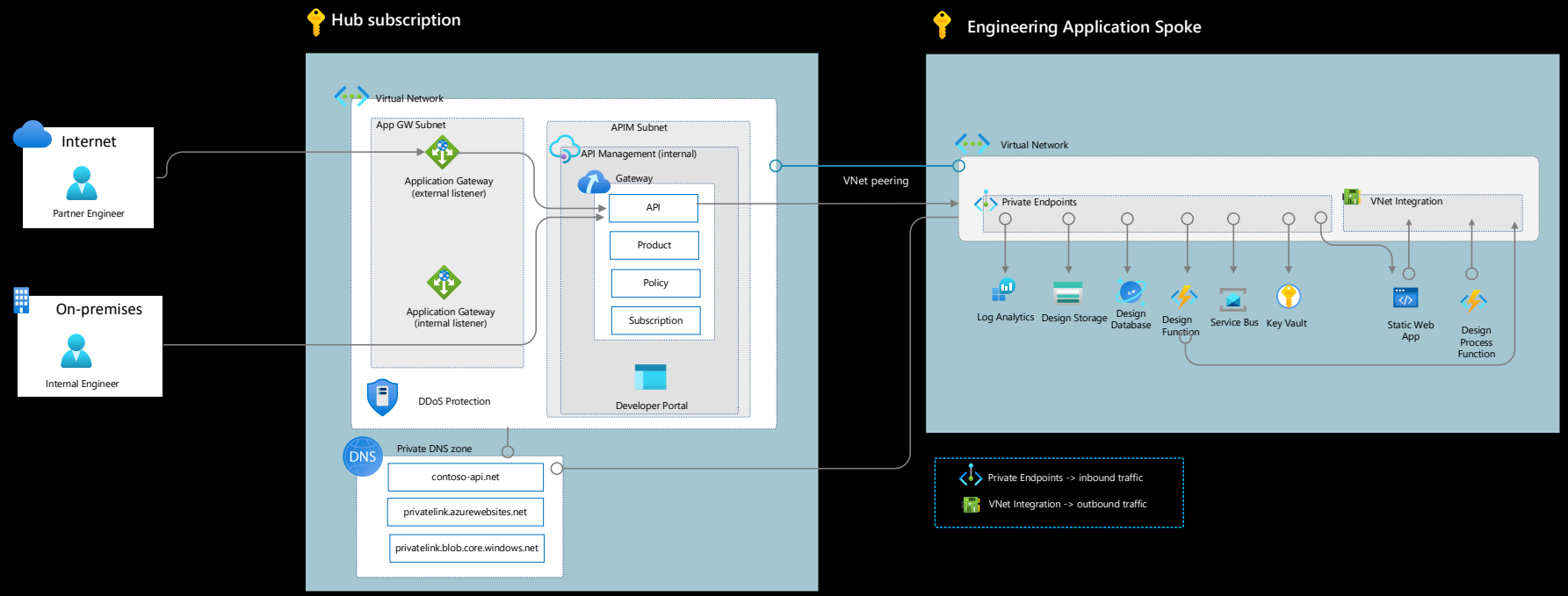
- Defense in Depth - Compensating Controls
- Attack surface reduction
- Secure defaults
- Least privilege
- Zero Trust
- Privacy

Defense in Depth

If one defense layer is breached, what other defense layers (if any) provide additional protection to the application?

Examples:

- Perimeter networks
- Identity and network
- Secure enclaves



Attack surface reduction

Attack Surface

Any part of an application that is accessible by a human or another program

Attack Surface Reduction

Minimize the number of exposed attack surface points a malicious user can discover and attempt to exploit

Examples:

- Turn off meta data endpoints (Swagger) in production
- No extra functionality
- Do not expose exception details

Secure defaults

Deploy applications in more secure configurations

Customers get a safer experience with your application

It is up to the user to reduce security and privacy levels

Examples:

- Reusable IaC components with locked down configurations
- Security defaults settings in Azure Active Directory

Least privilege

A program or service should be given only those privileges it needs in order to satisfy its requirements – no more, no less

- If a program doesn't need an access right, it should not be granted that right
- Think of it as *'need to know'* rule

If an application is compromised, then the potential damage that the malicious person can inflict is contained and minimized accordingly

- Higher privilege is something the attacker desires

Zero Trust - Compartmentalization

Minimize the effect of security breach

Increase intra and inter-application security

Trust boundaries can be built to isolate all components

- Where is my app running?
- From where does it load data?
- How is it validated?
- How are communication channels protected?
- Network defenses
- Identity defenses

Thinking backwards

Software engineering is about creating a solution for what users want to do

- Users want functionality, features and performance
- Do they want security?

Secure system development requires thinking about what the users want to do and what users should not be able to do and then creating a solution for that

- Users should not be able to invade or tamper with resources that are not allocated to them

Additional thoughts

Don’t reinvent the wheel

- You are probably not a security expert
- Microsoft and others provide many great security libraries
- Don’t create your own cryptography

Avoid security by obscurity

- Nothing on a device not controlled by you is secret (phones, consumer laptop)
- Do not rely on hiding something
- Do no rely on something being kept a secret forever

Security is by design

- Use threat modeling
- Azure blueprints, patterns, …

It’s your job, not the cybersecurity team’s

- They are not doing your job
- Do not rely on manual and pen tests to find your mistakes
- Use analysis tools to avoid making simplistic mistakes

Microsoft Azure

Input and output handling

Top three security issues

1 and 2 : input handling

- Input is the ‘r00t of all evil’ ! (defender motto)
- Never assume that input is valid; all input is suspect, until proven otherwise
- Never rely on client-side validation (very low trust); validate on server

Vulnerable statement

```
string sql = "SELECT * FROM users WHERE name = '" + username + "';";
```

Setting the ‘username’ to

```
a';DROP TABLE users;
```

Renders the following SQL statement

```
SELECT * FROM Users WHERE name = 'a';DROP TABLE users;--
```

3 - everything else

- AuthN, AuthZ, Least Privilege
- Auditing, Logging, SIEM
- Cryptography
- Secret handling
- Error handling
- Outdated components
- …

Where?

Never rely on client-side validation!

- JavaScript ‘onSubmit’ or ‘onPostBack’ events can be changed on client
- Use JavaScript as a functional validation to improve user experience only
- Adversaries will write custom code to break into your system

Defend your trust boundaries (be paranoid)

- Validate and/or sanitize all external input
- Validate return values of requests to services and data queries

Objective

- System does not perform tasks it is not designed to
- Non-malicious mistakes made by users are also caught and reported
- Make it harder for an attacker to penetrate a system

Input validation

Process only what you know you can process

- Reject and log everything else

Validate for type, length, range, format

- Define a strategy for your team/company
- Use regex if unavoidable

Safelist input

- For each user input field, there should be validation on the input content.
- Only accept data that meet a certain criteria

Do

- Take advantage of operating system-provided remediations
- OWASP 10 covers many input validation vulnerabilities (A1, A7 and others)
- Ensure semantic consistency (reusable validation libraries)
- Prefer whitelists over blacklists
- For input that needs more flexibility, blacklisting can also be applied where known bad input patterns or characters are blocked

Output handling

Conduct contextual output encoding

All output functions must contextually encode data before sending the data to the user

- For example, data placed in the URL context must be encoded differently than data placed in a JavaScript context within the HTML page

Input and output handling (cont.)

- Use parameterized SQL queries CWE - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (4.9) (mitre.org)
- Prevent insecure deserialization CWE - CWE-502: Deserialization of Untrusted Data (4.9) (mitre.org)
- Use tokens to prevent forged requests CWE - CWE-352: Cross-Site Request Forgery (CSRF) (4.9) (mitre.org)
- Prevent Server-Side Request Forgery (SSRF) CWE - CWE-918: Server-Side Request Forgery (SSRF) (4.9) (mitre.org)
- Set the encoding for your application CWE - CWE-172: Encoding Error (4.9) (mitre.org)
- Validate the source of input CWE - CWE-346: Origin Validation Error (4.9) (mitre.org)
- X-Frame-Options or CSP headers CAPEC - CAPEC-103: Clickjacking (Version 3.8) (mitre.org)
- Validate uploaded files CWE - CWE-434: Unrestricted Upload of File with Dangerous Type (4.9) (mitre.org)
- Prevent tabnabbing CWE - CWE-1022: Use of Web Link to Untrusted Target with window.opener Access (4.9) (mitre.org)

Microsoft Azure

Error handling

Exceptions

Handle all expected exceptions

Unexpected exceptions are security bugs (always)

- Often missed input validation
- Layer 7 DoS attacks

Log exception details

- Use to alert security team
- Don’t leak secrets into logs

Error messages leak details!

Use Simple Error Messages

- Show simple error messages not contain too much information
- Write detailed information to log files
- Especially: do not show SQL error messages

Use custom error pages for web

Do not leak secrets and highly confidential data to logs

- Be aware of logging on NVAs, especially query parameters
- GDPR trap

Microsoft Azure

Usable security

Security without usability

Usable security requires examination of the entire user experience. It requires a holistic view of the demands placed on the users, and a user-focused strategy

When a system's security features are difficult to access and/or apply, users will make mistakes or forgo protection all together.

Accessibility and usability

Easy to use != easy to break

Importance of usability for secure software | Signiant Blog

Microsoft Azure

Authentication and authorization

Passwordless authentication

UX Considerations

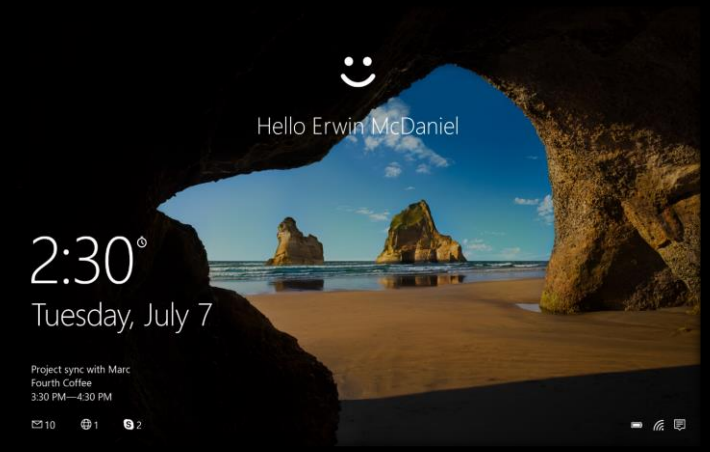
People find passwords hard to remember, this is especially difficult for people with certain disability

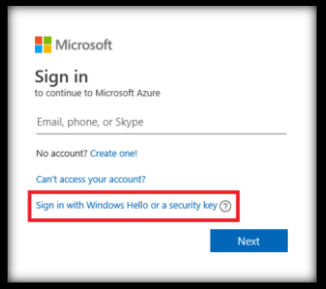
Increasing password length and complexity and forcing periodic password changes exacerbates people's frustration and reduce user experience.

Security Considerations

More than 60% of data breaches due to hacking use "stolen credentials" including usernames & passwords.

Passwords are susceptible to attacks targeted against individuals (e.g., social engineering, phishing, password stuffing and password-slurping malware).

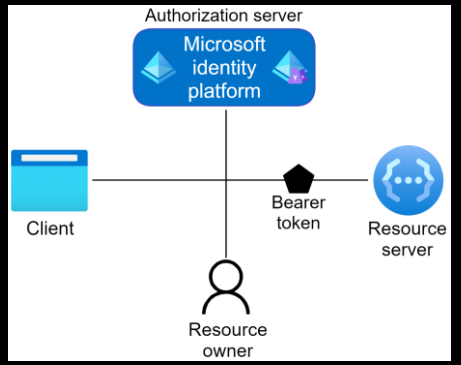




OAuth 2.0 and OpenID Connect

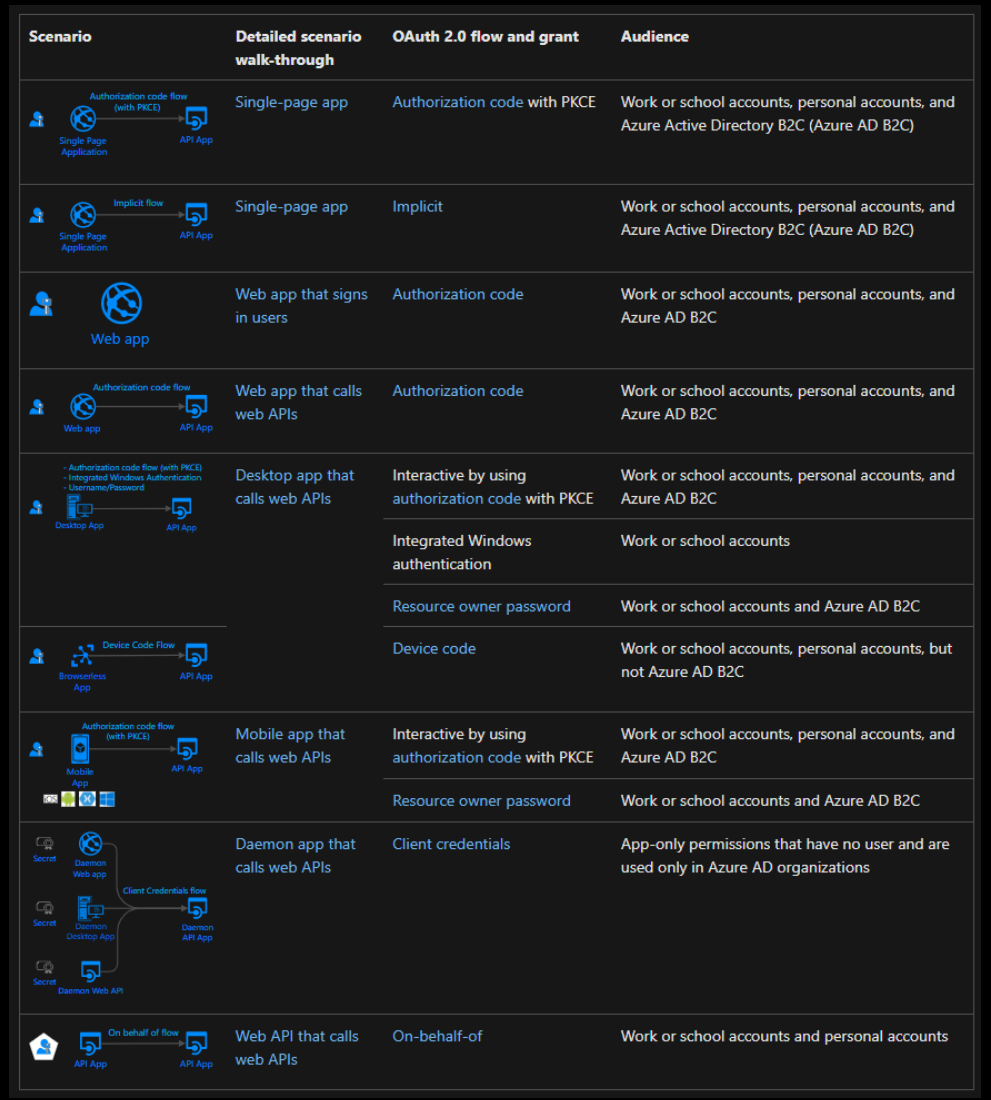
	OAuth2	OIDC
Goal	Resource server can decide whether to service a request	Client app knows the who the user is
Mechanism	Access token	Id token
Who asks for it?	Client	Client
Who validates it?	Resource server	Client
Process	One of OAuth2 defined flow types	Uses an OAuth2 flow type

- A client may not care to know who the user is (e.g. a mobile app) – pure OAuth2 protocol
- A client my not need to call any other APIs (resource servers) but need to know who the user is (e.g. a simple web app) – pure OIDC use of an OAuth2 flow
- Client needs to both know the user and call and API – hybrid flow



OAuth 2.0 sign-in flows

Flow name	User?	Browser?	Client type	Typical scenario
Client credential	N	N	C	Back-end service/daemon. Use X509 for stronger auth.
Authorization code grant w/ PKCE	Y	Y	C	Web Apps
	Y	Y	P	Mobile apps, SPA
Implicit flow	Y	?	P	No longer a suitable authentication method. Use authorization code flow instead.
Device code flow	Y	N	P	Browser-less public devices (e.g. thermostat, TV)
Extension flow (OBO)	Y	N	C	Multi-tier API. Can also be used to exchange a SAML token for JWT.
Resource owner pwd	Y	N	P	If all else fails, e.g. un-attended use of services requiring attended use (e.g. testing, offline mailbox processing)



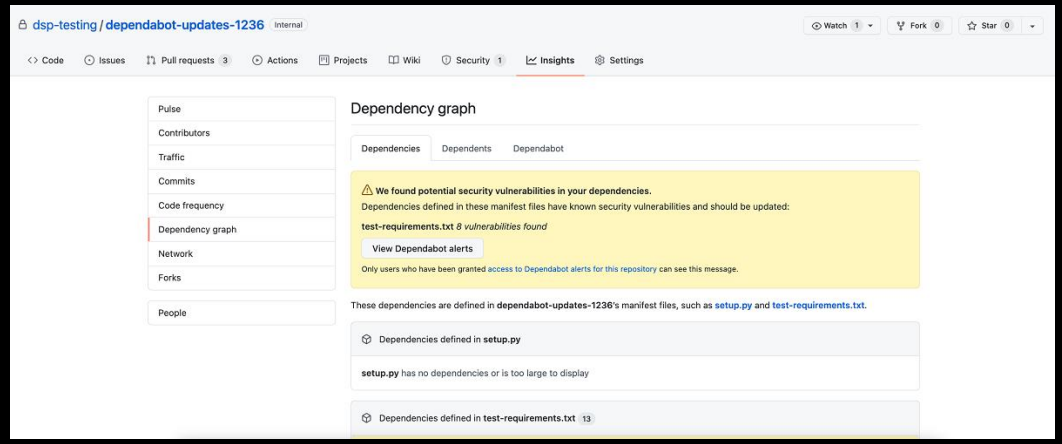
Microsoft Azure

Outdated components

How old are your components?

All software has (security) bugs

- Published at CVE - https://cve.mitre.org
- Often fixed
- Many breaches originated in well-known vulnerabilities
- Zero days are rare



What's your strategy?

- Never? Every 2 years? Every sprint?
- You regularly read CVE? Sure…
- Define your strategy (anything more than every 2 months is bad)

Use tools to manage

- Black Duck, Nexus, DependaBot, WhiteSource, …

Microsoft Azure

Cryptography

Problem space

Data stolen in transit, from memory, at rest

- Encrypt in transit
- Encrypt at rest
- Double encrypt secrets and highly confidential data

Encryption relies on

- Currently safe algorithms (not yet compromised)
- Strong key generation
- Minimum key lengths
- Storing and exchanging the keys securely
- Rotating keys on schedule and after a known compromise (emergency rotation)

Cryptography

Random Number Generation

Hashing

- Makes a unique value from a large data amount
- Fast (especially on graphics processors), unless artificially slowed (iterations)
- One-way transformation

Symmetric encryption

- Single shared key
- Fast!
- How do we share a key

Asymmetric encryption

- Based on extremely large prime number (2048+ bits)
- Public/private key pair – private key is arbitrary choice
- Slow!
- Encryption: Many to one encryption

Algorithm problem

All algorithms will eventually be compromised

- Computers are getting faster
- Quantum computing
- Algorithms have mathematical flaws (similar to code bugs)

Required

- Crypto algorithm standard (your org has one)
- Crypto agility SHA-1 was known to be broken since 2005, practical attacks in 2020

Microsoft Azure

Auditing, alerting and monitoring

Auditing and logging

Intentional security logging
Protect your logs
Backup logs regularly
Analyze logs near real-time
Use a SIEM and SOAR system

Intentional logging

Good Intentions

- Authentication/Authorization
- Key application events, such as requests for high value transactions
- Read/writes of critical data (but not the data itself)
- Exceptions

Warning! Be careful when logging sensitive information

- Logs are another source of input
- Sanitize data before logging
- Assume the attacker has access to this data

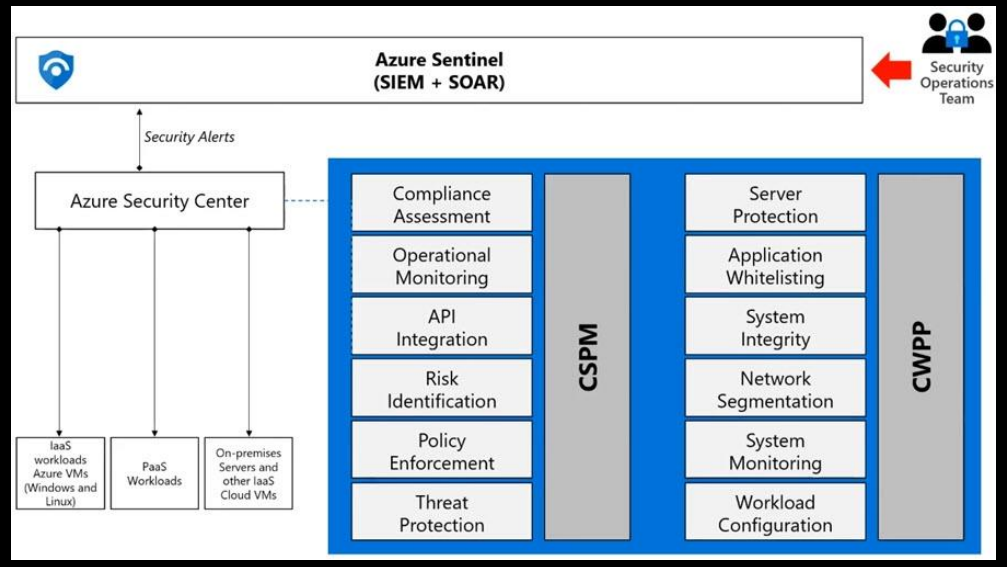
Microsoft Sentinel + Microsoft Defender for Cloud

Microsoft Defender for Cloud

- Cloud Security Posture Management (CSPM)
- Cloud Workload Protection Platform (CWPP)

Microsoft Sentinel

- Security Information and Event Management (SIEM)
- Security Orchestration, Automation and Response (SOAR)



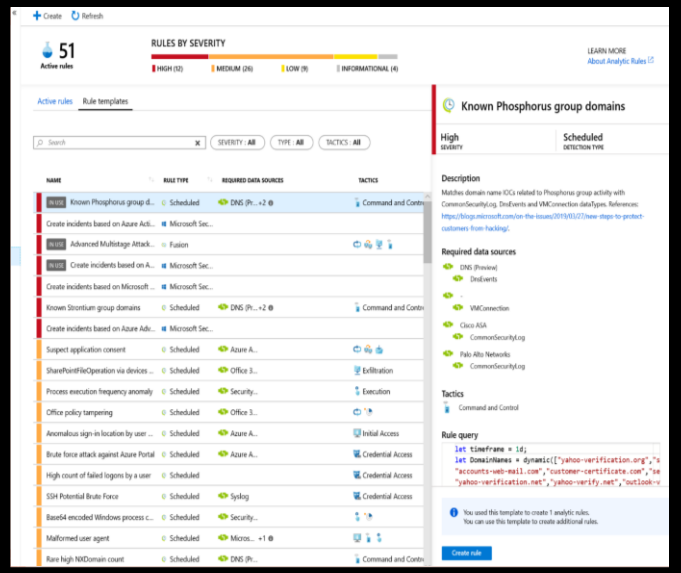
Microsoft Sentinel in brief

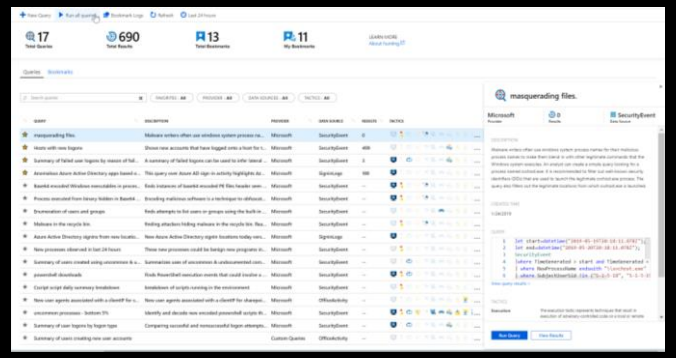
Analyze

- Choose from more than 100 built-in analytics rules
- Customize and create your own rules using KQL queries
- Correlate events with your threat intelligence and now with Microsoft URL intelligence

Hunt

- Run built-in threat hunting queries - no prior query experience required
- Customize and create your own hunting queries using KQL
- Integrate hunting and investigations





Microsoft Sentinel in brief

Automate

- Build automated and scalable playbooks that integrate across tools
- Choose from a library of samples
- Create your own playbooks using 200+ built-in connectors
- Trigger a playbook from an alert or incident investigation

Incident Management

Assign an Incident to an Analyst

Open a Ticket (ServiceNow/Jira)

Keep Incident Status in Sync

Post in a Teams or Slack Channel

Enrichment + Investigation

Lookup Geo for an IP

Trigger Defender ATP Investigation

Send Validation Email to User

Remediation

Block an IP Address

Block User Access

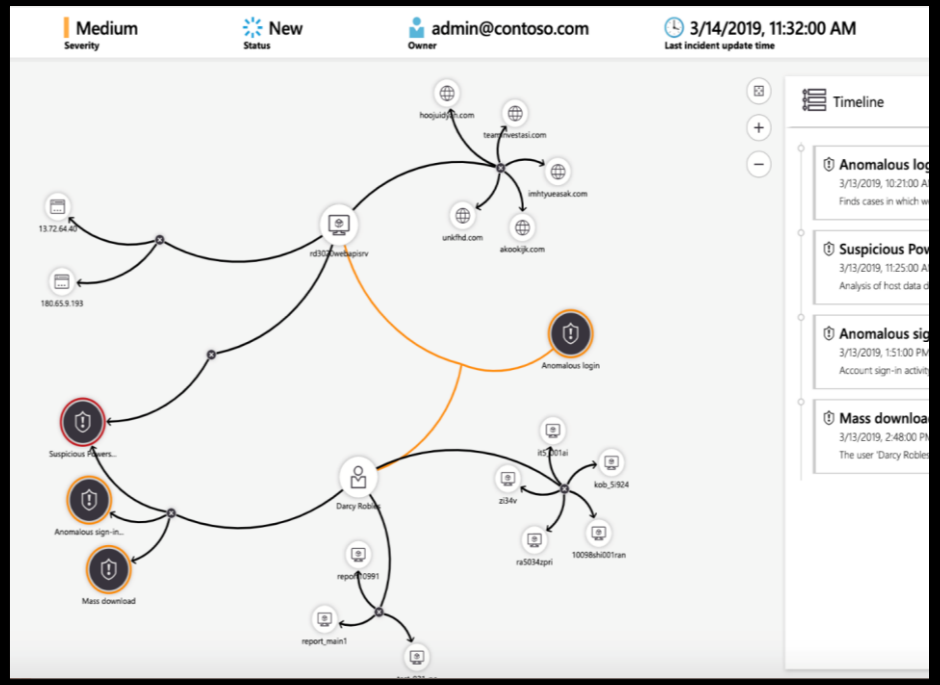
Trigger Conditional Access

Isolate Machine

Microsoft Sentinel in brief

Investigate

- Navigate the relationships between related alerts, bookmarks, and entities
- Expand the scope using exploration queries
- View a timeline of related alerts, events, and bookmarks
- Gain deep insights into related entities – users, domains, and more



Base level Monitoring

Minimum

- Infrastructure Monitoring
- APM Style (e.g.: Application Insights or comparable)

Modern Monitoring

- SRE concept based monitoring
  - SLI, SLO, Error budgets

  https://learn.microsoft.com/en-us/azure/site-reliability-engineering/articles/devops

Other:

Microsoft Defender for Cloud | DevOps Security

Lab - Broken Web App