**Name**: Michael Young (myoung10)

**Assignment**: 4-2

|  |  |  |  |
| --- | --- | --- | --- |
| **Coverage** | **Category** | **High Level Area** | **Scenarios/Risk Areas (***for the purposes of the homework instead fill in examples***)** |
| xyz | **Application Security** | Security "Expectations" of Application | * The application could use brute-force attempts to crack a valid authentication to access the API Gateway. * The application could masquerade as another application and steal sensitive information exclusive to that system. |
| xyz | REST API | * **-(REALISTIC SCENARIO)-**The REST API has a public IP without whitelisting the traffic coming into it. Therefore, an attacker could leverage these endpoints from any system, causing unthrottled usage/potential DOS. * **-(REALISTIC SCENARIO)-**The REST API can be vulnerable to Replay attacks in which the attacker would capture a legitimate request and re-send it to the REST endpoint, potentially bypassing any authentication. |
| xyz | Business Logic & Use Cases/Scenarios | * If models are not created to validate incoming data types in the API, then a person could send a string instead of an integer, which could cause unexpected behavior in the logic app, i.e. {num: “1”} vs {num: 1}, and an addition on this, in some languages would concatenate vs add. * ‘Warm-Starts’ to serverless logic applications could have residual information from previous runs, which could potentially disclose sensitive information or cause errors. |
| xyz | Ecosystem and Interactions | * The developer documentation could potentially display sensitive privileged endpoints as the web service publishes these automatically. The expectations are to show them what they need to do their job, but this portion is not behind authentication/authorization to show tailored endpoints. (the premium service allows this to be authenticated) * A query may be developed in the API payload to force the ‘Logic Apps’ to talk to an unintended system, i.e. the webservices instead of the backend systems. |
| xyz | Security "Expectations" of Backend System | * Publishing new interfaces to the developer portal could accidentally uncover ‘test’ endpoints or secure endpoints to the developer documentation. * Information is sent over HTTP, which is an encrypted transport and could be seen by an attacker. |
| xyz | **Operational  Security** | Software Build Systems | * There are no encryption standards for data in transit or at rest, and if an attacker was able to successfully access these systems, they could obtain sensitive data. * There is not a logging functionality shown for non-repudiation of service actions. |
| xyz | Development Team Practices/Hygiene | * The system does not have an integrated pipeline to validate security best practices or sync white-listed apps/endpoints. * The system does not have the concept of dev->prod deployments. All code is seemingly deployed to production, which could temporarily introduce vulnerabilities. |
| xyz | **Network & Infrastructure** | Operating Systems | * Logic Apps are run on a serverless platform, which means the OS is managed by the platform provider (Azure). If they fail to patch a critical vulnerability on a container (or very long running container), then the system could be exposed to an OS vulnerability (i.e. buffer overflows) which could yield access to the underlying container. * Logic Apps can run on several disparate OS’s causing an expanded attack surface and vulnerabilities to locate. For example, the defaults of a linux OS may allow an attacker to see a wider permission set of files compared with a windows system—allowing a path traversal exploit into core system config files. |
| xyz | Private Network Boundary | * These [backend] services are not on a private network, therefore each node is publicly accessible and attackable. Attacks can come in from any point in the system, and this can make it difficult for logging/capturing/stopping malicious payloads, especially unauthenticated requests. * The logic applications are also not in a private network. Which means an attacker could invoke these services directly to bypass any auth/model validation. A malicious payload could be uploaded to the logic application (or modified), and then subsequently automatically published to the developer portal, which would then introduce the malicious payload to all application creators once they visit the portal (i.e. XSS). |