High-level requirements for DGP security:

1. The platform should, at a minimum, correct all 10 of the OWASP API Top 10 Security Issues.
2. The platform should also correct as many of the security problems listed in the CISQ 2022 report on the annual cost of poor-quality software systems as possible.
3. (optional) DGP software systems should be designed and built to be able to pass a PCI-DSS audit. An audit itself would not be necessary for most systems, but the preparation represents a testable and verifiable level of effective security.

Requirements

1. Quick patching of all system infrastructure

What: *The quick application of patches to all parts of the infrastructure of a system.*

Why: *Frequent and timely patching the infrastructure of a system is the single most effective way to improve the security of the system as a whole. This includes operating systems, app servers, firmware, etc.*

Testing: *Patches should be applied sequentially starting with the least critical environments (dev), then testing, and then QA to prove the patches cause no problems to the system before they are applied to the production environments.*

2. Data-driven role-based access control (RBAC) for account authentication, API method authorization based on role membership, and shared data access based on group membership, distinct per environment.

What: *A data-driven role-based access control system is used to first authenticate requests to the web service API's, and then create access control lists that are used to authorize each API method called by the authenticated account. Similarly, access to shared data is controlled by data group membership.*

Why: *An authenticated account should not be able to call every API method in a system. The ACL created by the RBAC is used to control which API methods each account is authorized to call. Admin UI's are used to maintain the security data.*

Testing: *Postman and test harness tests of accounts successfully calling authorized API methods, and failing to call API methods for which they have not been authorized.*

3. The RBAC security system should provide multiple optional forms of multi-tenancy.

What: *The security system should allow for both physical partitioning (separate databases per client) and optional logical portioning between clients using ID values within a shared database as needed.*

Why: *When a system is shared by multiple users, departments, organizations, etc. it is necessary to separate and protect the data of each segment of users from access by those that are not members of that segment.*

Testing: *The segregation of the data is implemented as security groups in a DGP system. The logic to restrict both reading and writing data by security group membership is built into each CRUD API method's data access logic (SQL syntax).*

4. Message-based RPC APIs

What: *APIs must be implemented as Remote Procedure Calls (RPC) message-based APIs executed by sending API request and response messages between the client and the server.*

Why: *Message-based API request and response messages can be used as a two-way communication mechanism between the client and the server. This flexible communication is able to transfer a lot more information than the standard method calls input parameters and return value, and that extra data is used for method authentication and authorization. It is important that the messages be treated as nothing more than structured text (XML fragments), and NOT as serialized objects. This is less of an issue for HTTP services, where the HTTP request and response body contains the messages, but is more important for SOAP web services.*

Testing: *The integrated test harness (or tools such as Postman) can be used to send test messages to the API controller and receive API response messages in return. However, the API Tester test harness has been designed to specifically work with the XML fragments, and has some extra functionality built in that the other tools do not support by default.*

5. Security account credentials included as part of each API message: depends on message-based RPC APIs

What: *Each API message must contain the security account token embedded as part of each message, which are mandatory for API request messages and optional for API response messages (used to validate the server to the client).*

Why: *Embedding the security token (HMAC hash) into each API message helps simplify the authentication/authorization process, and also works just as well if the messages are queued for deferred execution. API messages should be encrypted during transport to protect the content being sent to and returned by the API methods.*

Testing: *Postman or the integrated test harness can be used to send test messages to the API controller and receive API response messages in return.*

6. Minimum possible API attack surface: depends on message-based APIs

What: *The use of message-based APIs allow a single argument front controller pattern to be used for the web services.*

Why: *Each exposed endpoint is a chance for attackers to find an exploit of some kind to gain access to a system. The fewer endpoints exposed publicly the better. The minimum possible public attack surface consists of a single web service, which exposes a single front controller method, which accepts a single input parameter (an API requests message) and returns a single value (an API response message).*

Testing: *End-to-end tests of an API, especially when using the API Tester test harness, will highlight the number of endpoint URL's the test client must keep track of.*

7. API request TTL: depends on message-based APIs

What: *Each API request message must contain a timestamp that expires in a relatively short period of time.*

Why: *Setting a maximum time-to-live for request messages helps to prevent the replay of messages, which is one of the most common techniques used for DDOS attacks.*

Testing: *Automated tests run by the API Tester test harness call API methods successfully with current timestamps, and fail with expired timestamp values.*

8. Account rate limits: depends on message-based APIs

What: *The number of API methods called by an account in a given period of time should be tracked, and the API calls rejected if the number of method calls exceeds the maximum allowed.*

Why: *This helps to prevent accounts from flooding a system with API request messages (as in a DDOS attack) or monopolizing the resources of the system. Any API exposed publicly must have this type of rate limit functionality.*

Testing: *Automated test methods can be called at a high rate for accounts that have a low rate limit set in order to test the functionality that limits method calls when the maximum has been exceeded for a time period.*

9. Failed authentication count and user account lockout: depends on message-based APIs

What: *The number of consecutive authentication failures must have an allowed maximum, and any failures above that number must disable the account.*

Why: *This maximum failed sequential authentication count and lockout helps to prevent brute-force dictionary attacks used to discover user passwords. A successful authentication resets the count to zero.*

Testing: *Automated tests create a test account and then deliberately use incorrect credentials for multiple API method calls to test this lockout functionality.*

10. Encryption of API request and response messages during transport

What: *API request and response messages must be encrypted during transport.*

Why: *First, the request and response messages contain sensitive data, and so production systems should encrypt those request and response messages. Second, it is a necessary condition needed for PCI-DSS certification.*

Testing: *TLS cannot be tested by the web services themselves (the encryption is transparent to the API request and response messages). The built-in message encryption can be tested using the API Tester test harness.*

11. Simple and fast configuration data management plus encrypted system secrets.

What: *Configuration data and certain secret data such as ADO.NET connection strings, encryption keys, etc. are needed by the web services during their operation. This data must be encrypted and stored securely for the web services.*

Why: *Secret data contains account names, passwords, encryption keys, and so on – which must be encrypted for PCI compliance.*

Testing: *The secret data can be observed in storage to see if it is encrypted or in clear text.*

12. Application-level encryption

What: *Some sensitive data such as passwords, credit card data, etc. must be stored in encrypted form in the databases of a system.*

Why: *Zero-knowledge encryption separates the encryption keys from the encrypted data (on a different physical tier) from the encrypted data storage. It prevents DBA's and other admins from being able to see the data content in clear text when using admin tools, consoles, etc.*

Testing: *The data in the databases can be observed using various admin tools to verify that it has been encrypted and is not visible as clear text. This can be compared to the clear text values sent in the API messages as another mechanism to verify that the encryption and decryption of the values to and from the database tables are all working correctly.*

13. Endpoint protection.

What: *Anti-virus software must be run on all servers in a system to protect them from most common types of attacks.*

Why: *Endpoint (anti-virus) protection is an important part of the overall security of servers in production environments.*

Testing: *Periodic scans are run on the memory and storage of the servers looking for various types of malware.*

14. Executable white-list protection.

What: *Another part of endpoint protection is software that only allows executables contained in a whitelist to be allowed to run on each server.*

Why: *Preventing unauthorized executables from running on servers is one of the ways to protect against malware infections.*

Testing: *Attempts are made to run different types of unauthorized executables, which should all be blocked.*

15. Admin UI's to manage the RBAC data

What: *The platform requires administrative UI applications that make it easy to manage and maintain the RBAC data for a system.*

Why: *The fine-grained control over which accounts are authorized to access which API methods and data security groups is also the primary feature-toggle mechanism used for each environment, and also will control which accounts have access to which products/services…*

Testing: *Using the API Tester test harness to call both authorized and unauthorized API methods, which are successful for authorized methods and fail for unauthorized methods.*

16. The deliverables for the web app and web service tiers of the software system should be natively compiled binaries.

What: *The source code of web apps and web services should be compiled into native executables for deployment.*

Why: *Native binaries cannot be modified, unlike interpreted languages which deploy and run the source code. Avoiding dynamic programming languages protects systems against the multiple types of code injection attacks. Also, the performance of compiled executables is much better than interpreted code.*

Testing: *Interpreted source code can be modified after it is deployed, while natively compiled binaries are immutable.*

17. No .NET code generators

What: *DGP should be built with no automatically generated code, especially avoiding any 3rd party code generators.*

Why: *Code generators (added in .NET 5) are one of the newest system attack vectors. They are particularly problematic for public dependency ecosystems such as NuGet. The easiest solution to this problem is to avoid them entirely.*

Testing: *If open source libraries and/or tools are built into a DGP system, software tools that scan the entire tree of open source code dependencies must be used prior to every new release being deployed.*

18. No dependencies on open-source libraries.

What: *DGP should be built with little or no open-source libraries or tools (refer to the SBOM for a list of all dependencies).*

Why: *Open-source tools have become one of the best and most popular attack vectors used by hackers. Malicious code is injected into low level open-source libraries, which are then used by many open-source projects.*

Testing: *If open-source libraries and/or tools are built into a DGP system, software tools that scan the entire tree of open-source code dependencies must be used prior to every new release being deployed.*