Threat Modeling Report

Created on 10/25/2020 1:32:45 PM
Threat Model Name:
Owner:
Reviewer:
Contributors:
Description:
Assumptions:
External Dependencies:

Threat Model Summary:

Not Started	0
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	23
Total	23
Total Migrated	0

Diagram: Diagram 1

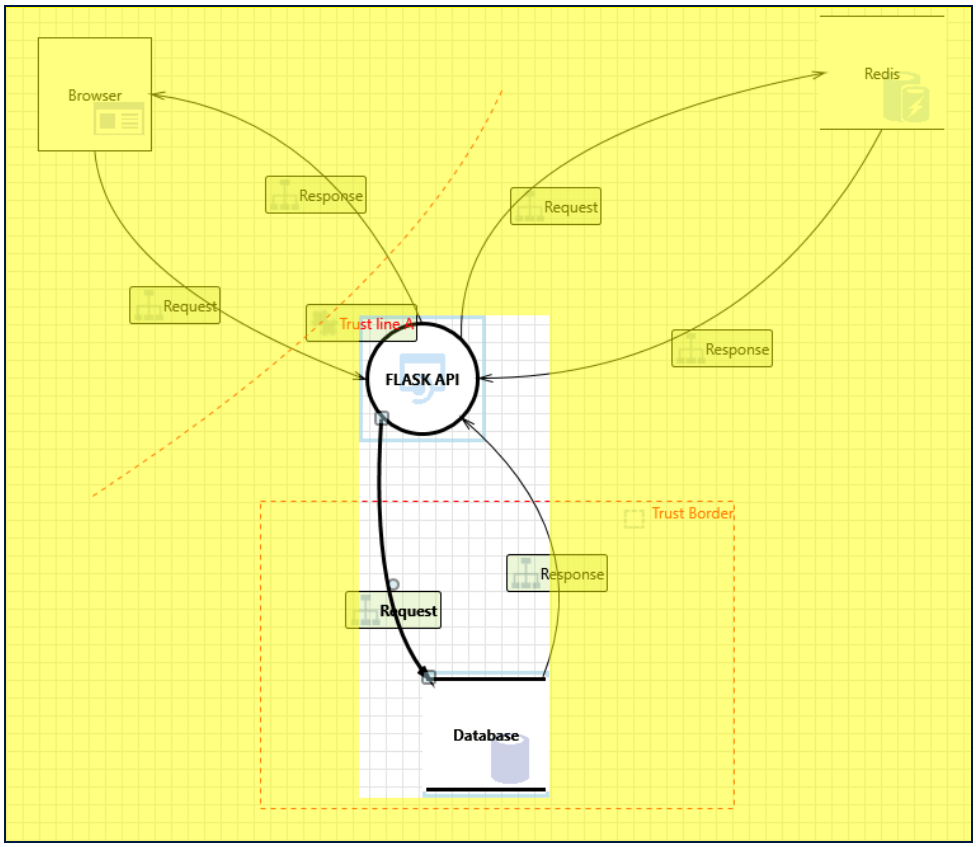
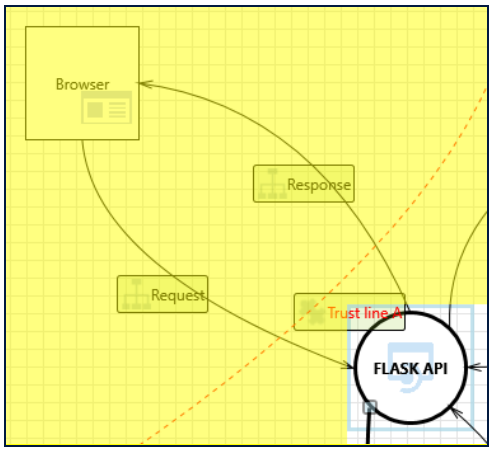


Diagram 1 Diagram Summary:

Not Started	0
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	23
Total	23
Total Migrated	0

Interaction: Request



1. An adversary may gain unauthorized access to Web API due to poor access control checks [State: Mitigation Implemented] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary may gain unauthorized access to Web API due to poor access control checks
Justification:	Strict access control, and multiple checks inplace. (2FA, existance in redis, strong passwords) Integrated login manager
Short Description:	A user subject gains increased capability or privilege by taking advantage of an implementation bug
Possible Mitigation(s):	Implement proper authorization mechanism in ASP.NET Web API. Refer: https://aka.ms/tmtauthz#authz-aspnet
SDL Phase:	Implementation

2. An adversary can gain access to sensitive information from an API through error messages [State: Mitigation Implemented] [Priority: High]

Category:	Information Disclosure
Description:	An adversary can gain access to sensitive data such as the following, through verbose error messages - Server names - Connection strings - Usernames - Passwords - SQL procedures - Details of dynamic SQL failures - Stack trace and lines of code - Variables stored in memory - Drive and folder locations - Application install points - Host configuration settings - Other internal application details
Justification:	Implemeted generic error messages where this could occur.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Ensure that proper exception handling is done in ASP.NET Web API. Refer: https://aka.ms/tmtxmgmt#exception
SDL Phase:	Implementation

3. An adversary may retrieve sensitive data (e.g, auth tokens) persisted in browser storage [State: Mitigation Implemented] [Priority: High]

Category:	Information Disclosure
Description:	An adversary may retrieve sensitive data (e.g, auth tokens) persisted in browser storage
Justification:	This is mitigated, after logout its no longer possible to view cache.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Ensure that sensitive data relevant to Web API is not stored in browser's storage. Refer: https://aka.ms/tmtdata#api-browser
SDL Phase:	Implementation

4. An adversary can gain access to sensitive data by sniffing traffic to Web API [State: Mitigation Implemented] [Priority: High]

Category:	Information Disclosure
Description:	An adversary can gain access to sensitive data by sniffing traffic to Web API
Justification:	HTTPS
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Force all traffic to Web APIs over HTTPS connection. Refer: https://aka.ms/tmtcommsec#webapi-https
SDL Phase:	Implementation

5. An adversary can gain access to sensitive data stored in Web API's config files [State: Mitigation Implemented] [Priority: Medium]

Category:	Information Disclosure
Description:	An adversary can gain access to the config files. and if sensitive data is stored in it, it would be compromised.
Justification:	Mitigated through use of well known and tested API.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Encrypt sections of Web API's configuration files that contain sensitive data. Refer: https://aka.ms/tmtconfigmgmt#config-sensitive

SDL Phase:	Implementation

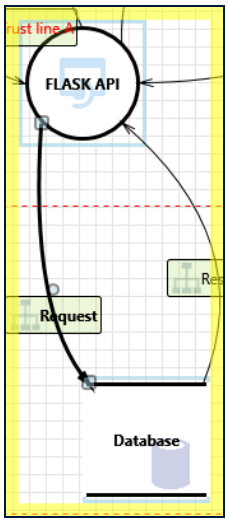
6. An adversary may inject malicious inputs into an API and affect downstream processes [State: Mitigation Implemented] [Priority: High]

Category:	Tampering
Description:	An adversary may inject malicious inputs into an API and affect downstream processes
Justification:	User input sanitization, checking each char in inpt string. Only allowing highly restricted set of characters.
Short Description:	Tampering is the act of altering the bits. Tampering with a process involves changing bits in the running process. Similarly, Tampering with a data flow involves changing bits on the wire or between two running processes
Possible Mitigation(s):	Ensure that model validation is done on Web API methods. Refer: https://aka.ms/tmtinputval#validation-api Implement input validation on all string type parameters accepted by Web API methods. Refer: https://aka.ms/tmtinputval#string-api
SDL Phase:	Implementation

7. An adversary can gain access to sensitive data by performing SQL injection through Web API [State: Mitigation Implemented] [Priority: High]

Category:	Tampering
Description:	SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed.
Justification:	all queries are done through flask, with fucntion query.filter_by() that sanitizes by default. We do use raw sql commands in trasaction history, however its mitigated with illegalChars() function
Short Description:	Tampering is the act of altering the bits. Tampering with a process involves changing bits in the running process. Similarly, Tampering with a data flow involves changing bits on the wire or between two running processes
Possible Mitigation(s):	Ensure that type-safe parameters are used in Web API for data access. Refer: https://aka.ms/tmtinputval#typesafe-api
SDL Phase:	Implementation

Interaction: Request



8. An adversary can gain unauthorized access to database due to lack of network access protection [State: Mitigation Implemented] [Priority: High]

Category:	Elevation of Privileges
Description:	If there is no restriction at network or host firewall level, to access the database then anyone can attempt to connect to the database from an unauthorized location
Justification:	Integrated database into flask API. Technically possible to retrive database by connecting to server and downloading it. Mitigated with server access control and firewalls.
Short Description:	A user subject gains increased capability or privilege by taking advantage of an implementation bug
Possible Mitigation(s):	Configure a Windows Firewall for Database Engine Access. Refer: https://aka.ms/tmtconfigmgmt#firewall-db
SDL Phase:	Implementation

9. An adversary can gain unauthorized access to database due to loose authorization rules [State: Mitigation Implemented] [Priority: High]

Category:	Elevation of Privileges
Description:	Database access should be configured with roles and privilege based on least privilege and need to know principle.
Justification:	Integrated database. Random secret key.

Short Description:	A user subject gains increased capability or privilege by taking advantage of an implementation bug
Possible Mitigation(s):	Ensure that least-privileged accounts are used to connect to Database server. Refer: https://aka.ms/tmtauthz#privileged-server Implement Row Level Security RLS to prevent tenants from accessing each others data. Refer: https://aka.ms/tmtauthz#rls-tenants Sysadmin role should only have valid necessary users . Refer: https://aka.ms/tmtauthz#sysadmin-users
SDL Phase:	Implementation

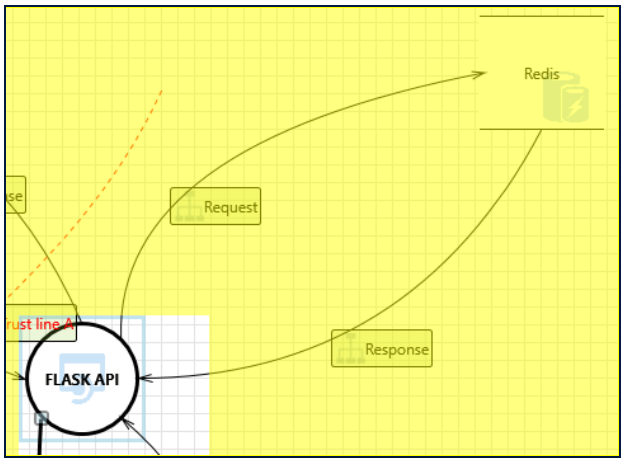
10. An adversary can gain access to sensitive data by performing SQL injection [State: Mitigation Implemented] [Priority: High]

Category:	Information Disclosure
Description:	SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed.
Justification:	If the flask API is compromised then this is possible, mitigtaed through using a well known API
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Ensure that login auditing is enabled on SQL Server. Refer: https://aka.ms/tmtauditlog#identify-sensitive-entities Ensure that least-privileged accounts are used to connect to Database server. Refer: https://aka.ms/tmtauthz#privileged-server Enable Threat detection on Azure SQL database. Refer: https://aka.ms/tmtauditlog#threat-detection Do not use dynamic queries in stored procedures. Refer: https://aka.ms/tmtinputval#stored-proc
SDL Phase:	Implementation

11. An adversary can gain access to sensitive PII or HBI data in database [State: Mitigation Implemented] [Priority: High]

Category:	Information Disclosure
Description:	Additional controls like Transparent Data Encryption, Column Level Encryption, EKM etc. provide additional protection mechanism to high value PII or HBI data.
Justification:	Personal data is encrypted and secured with user password. Ownership lookup requires password and is not possible to do without the password either for developers or users.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Use strong encryption algorithms to encrypt data in the database. Refer: https://aka.ms/tmtcrypto#strong-db Ensure that sensitive data in database columns is encrypted. Refer: https://aka.ms/tmtdata#db-encrypted Ensure that database-level encryption (TDE) is enabled. Refer: https://aka.ms/tmtdata#tde-enabled Ensure that database backups are encrypted. Refer: https://aka.ms/tmtdata#backup Use SQL server EKM to protect encryption keys. Refer: https://aka.ms/tmtcrypto#ekm-keys Use AlwaysEncrypted feature if encryption keys should not be revealed to Database engine. Refer: https://aka.ms/tmtcrypto#keys-engine
SDL Phase:	Implementation

Interaction: Request

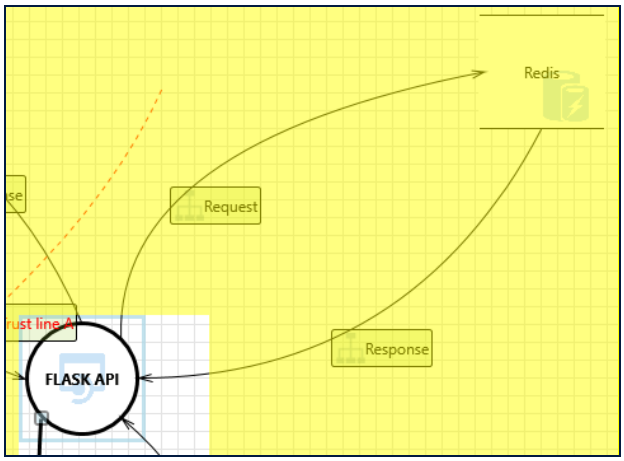


12. An adversary can read sensitive data by sniffing traffic to Redis [State: Mitigation Implemented] [Priority: High]

Category:	Information Disclosure
Description:	An adversary can read sensitive data by sniffing traffic to Redis

Justification:	Redis runs locally, but possible. Again there are more serious issues if this is the case.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Ensure that communication to Redis is over SSL/TLS. Configure Redis such that only connections over SSL/TLS are permitted. Also ensure that connection string(s) used by clients have the ssl flag set to true (I.e. ssl=true). Refer: https://aka.ms/tmt-th14.
SDL Phase:	Implementation

Interaction: Response



13. An adversary can gain access to sensitive data by performing SQL injection through Web API [State: Mitigation Implemented] [Priority: High]

Category:	Tampering
Description:	SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed.
Justification:	No raw user input is used to generate queries to redis.
Short Description:	Tampering is the act of altering the bits. Tampering with a process involves changing bits in the running process. Similarly, Tampering with a data flow involves changing bits on the wire or between two running processes
Possible Mitigation(s):	Ensure that type-safe parameters are used in Web API for data access. Refer: https://aka.ms/tmtinputval#typesafe-api
SDL Phase:	Implementation

14. An adversary may inject malicious inputs into an API and affect downstream processes [State: Mitigation Implemented] [Priority: High]

Category:	Tampering
Description:	An adversary may inject malicious inputs into an API and affect downstream processes
Justification:	requests are generated by Web API and user input is sanitized.
Short Description:	Tampering is the act of altering the bits. Tampering with a process involves changing bits in the running process. Similarly, Tampering with a data flow involves changing bits on the wire or between two running processes
Possible Mitigation(s):	Ensure that model validation is done on Web API methods. Refer: https://aka.ms/tmtinputval#validation-api Implement input validation on all string type parameters accepted by Web API methods. Refer: https://aka.ms/tmtinputval#string-api
SDL Phase:	Implementation

15. An adversary can gain access to sensitive data stored in Web API's config files [State: Mitigation Implemented] [Priority: Medium]

Category:	Information Disclosure
Description:	An adversary can gain access to the config files. and if sensitive data is stored in it, it would be compromised.
Justification:	Technically true, but redis config editing requries root privileges.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Encrypt sections of Web API's configuration files that contain sensitive data. Refer: https://aka.ms/tmtconfigmgmt#config-sensitive
SDL Phase:	Implementation

16. An adversary can gain access to sensitive information from an API through error messages [State: Mitigation Implemented] [Priority: High]

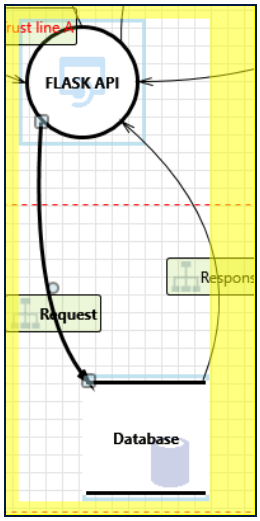
Category:	Information Disclosure
Description:	An adversary can gain access to sensitive data such as the following, through verbose error messages - Server names - Connection strings - Usernames - Passwords - SQL procedures - Details of dynamic SQL failures - Stack trace and lines of code - Variables stored in memory - Drive and folder locations - Application install points - Host configuration settings - Other internal application details

Justification:	redis errors are never displayed to user.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Ensure that proper exception handling is done in ASP.NET Web API. Refer: https://aka.ms/tmtxmgmt#exception
SDL Phase:	Implementation

17. An adversary may gain unauthorized access to Web API due to poor access control checks [State: Mitigation Implemented] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary may gain unauthorized access to Web API due to poor access control checks
Justification:	Mitigated with server password, redis server only runs locally.
Short Description:	A user subject gains increased capability or privilege by taking advantage of an implementation bug
Possible Mitigation(s):	Implement proper authorization mechanism in ASP.NET Web API. Refer: https://aka.ms/tmtauthz#authz-aspnet
SDL Phase:	Implementation

Interaction: Response



18. An adversary can gain access to sensitive data by performing SQL injection through Web API [State: Mitigation Implemented] [Priority: High]

Category:	Tampering
Description:	SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. The primary form of SQL injection consists of direct insertion of code into user-input variables that are concatenated with SQL commands and executed. A less direct attack injects malicious code into strings that are destined for storage in a table or as metadata. When the stored strings are subsequently concatenated into a dynamic SQL command, the malicious code is executed.
Justification:	given that the web API is compromised this is possible, but mitigated by using a well tested API.
Short Description:	Tampering is the act of altering the bits. Tampering with a process involves changing bits in the running process. Similarly, Tampering with a data flow involves changing bits on the wire or between two running processes
Possible Mitigation(s):	Ensure that type-safe parameters are used in Web API for data access. Refer: https://aka.ms/tmtinputval#typesafe-api
SDL Phase:	Implementation

19. An adversary may inject malicious inputs into an API and affect downstream processes [State: Mitigation Implemented] [Priority: High]

Category:	Tampering
Description:	An adversary may inject malicious inputs into an API and affect downstream processes
Justification:	Possible, but we strictly enforce input sanitization which mitigates this threat.
Short Description:	Tampering is the act of altering the bits. Tampering with a process involves changing bits in the running process. Similarly, Tampering with a data flow involves changing bits on the wire or between two running processes
Possible Mitigation(s):	Ensure that model validation is done on Web API methods. Refer: https://aka.ms/tmtinputval#validation-api Implement input validation on all string type parameters accepted by Web API methods. Refer: https://aka.ms/tmtinputval#string-api
SDL Phase:	Implementation

20. An adversary can gain access to sensitive data stored in Web API's config files [State: Mitigation Implemented] [Priority: Medium]

Category:	Information Disclosure
Description:	An adversary can gain access to the config files. and if sensitive data is stored in it, it would be compromised.
Justification:	Yes, this does require server login, and thus mitigated thorugh server security implementations.

Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Encrypt sections of Web API's configuration files that contain sensitive data. Refer: https://aka.ms/tmtconfigmgmt#config-sensitive
SDL Phase:	Implementation

21. An adversary can gain access to sensitive data by sniffing traffic to Web API [State: Mitigation Implemented] [Priority: High]

Category:	Information Disclosure
Description:	An adversary can gain access to sensitive data by sniffing traffic to Web API
Justification:	Yes, but this is running locally on server so you would need to actually be logged in for this to occur. By then we have larger issues.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Force all traffic to Web APIs over HTTPS connection. Refer: https://aka.ms/tmtcommsec#webapi-https
SDL Phase:	Implementation

22. An adversary can gain access to sensitive information from an API through error messages [State: Mitigation Implemented] [Priority: High]

Category:	Information Disclosure
Description:	An adversary can gain access to sensitive data such as the following, through verbose error messages - Server names - Connection strings - Usernames - Passwords - SQL procedures - Details of dynamic SQL failures - Stack trace and lines of code - Variables stored in memory - Drive and folder locations - Application install points - Host configuration settings - Other internal application details
Justification:	Mitigated, either no error is given or a generic one is given by the actual function that preforms the action. ciritcal functions like verify user are encapsulated in try: except: to catch all errors.
Short Description:	Information disclosure happens when the information can be read by an unauthorized party
Possible Mitigation(s):	Ensure that proper exception handling is done in ASP.NET Web API. Refer: https://aka.ms/tmtxmgmt#exception
SDL Phase:	Implementation

23. An adversary may gain unauthorized access to Web API due to poor access control checks [State: Mitigation Implemented] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary may gain unauthorized access to Web API due to poor access control checks
Justification:	Web API can only be access through server, wich is secured with strong password and firewall.
Short Description:	A user subject gains increased capability or privilege by taking advantage of an implementation bug
Possible Mitigation(s):	Implement proper authorization mechanism in ASP.NET Web API. Refer: https://aka.ms/tmtauthz#authz-aspnet
SDL Phase:	Implementation