PREPARED FOR:
VALIDUS HOLDINGS (BERMUDA) LTD

WEB APPLICATION SECURITY TESTING REPORT

Prepared by: MTI Technology Ltd

For: Validus Holdings (Bermuda) Limited

Ref No: 07808

VERSION CONTROL

Version	Author	Date	Revision Detail
0.1	Dave Daly	7/7/2018	Created
0.2	QA	12/7/2018	Initial QA
1.0	Internal	13/7/2018	Final Version

COPYRIGHT

CONSULTANT

Name	Relevant Credentials	Contact Number
Dave Daly	CCT-INF	01483 520200

TABLE OF CONTENTS

1.0 SCOPE OF TESTING

MTI Technology were commissioned by Validus Holdings (Bermuda) Limited ("Validus") to conduct a Web Application test against a single Internet facing application. The aim of the test is to ensure that the Internet facing application is secure enough to be exposed to external users and is free from known vulnerabilities and threats.

Specifically the following application was tested:

- https://online.talbotuw.com/products/political-violence/malaysia

The following user accounts were supplied for use during the penetration test.

Agency	User	Role	Username
MTI RI Penn Testing	MTI ExtRI	External Agent RI Lines	mti.extri
MTI RI Venture Penn Testing	MTI ExtRI2	External Agent RI Lines	mti.extri2
MTI UW	MTI UW	Underwriter	mti.uw

1.1. CHANGES TO SCOPE

- The forgotten password functionality was not fully functional during testing; it was therefore not possible to fully test this feature.

2.0 EXECUTIVE SUMMARY

Overall, the web application appeared to be developed with effective access control in mind. Permissions applied to user roles couldn't be bypassed in the time permitted, and the use of authorisation checking for any actions that involved accessing or changing data were effective.

There were, however, several potentially significant issues that affected the ability of the web application and web server to withstand indirect attacks against users, and to guarantee integrity of data. Any of these could lead to reputational damage or financial loss if exploited.

3.0 TECHNICAL SUMMARY

While direct compromise of the web application and the web server were not achieved successfully, several potential vectors for indirect attacks against users were identified. These are discussed below.

Firstly, the web server does not have any visible protection against malware embedded in uploaded spreadsheet files, for example anti-virus scanning that looked at the quote template files before the values were added to new quote forms. It was possible to upload a benign malware signature, the EICAR string, without it being rejected as potentially malicious. In addition, it was possible to embed executable code into quote values, which was then included in downloaded reports and ran when the reports were opened. Either of these flaws could lead to users of the application having their computers compromised, and subsequently taking legal action against Validus.

Secondly, the web application doesn't restrict access to scripts, data or pages from other domains. This means that web pages can be embedded in other websites, or scripts can be used to extract data from logged in users, provided that the users can tricked into visiting malicious websites while they are logged into the web application.

Finally, the website doesn't force web browsers to enable security options that are designed to protect users against indirect attacks that utilise the trust between the browser and the web application that the user is currently interacting with. These include enforced use of encryption, blocking of unexpected scripts, and filtering of known malicious content.

4.0 DEFINITION OF VULNERABILITIES

4.1. HIGH, MEDIUM & LOW RISK

Throughout this report certain symbols are used to help give an indication of the severity of the issues that are found.

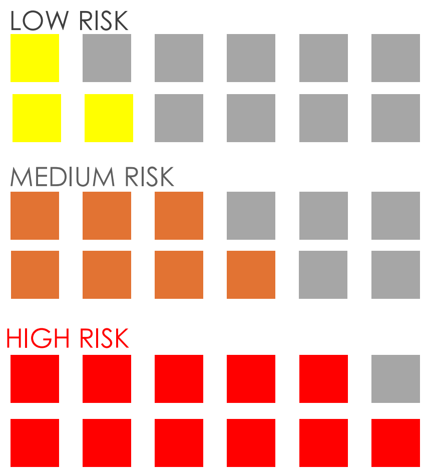
This severity level takes into consideration the following factors:

Ease of exploitation
Likelihood of vulnerability being exploited
Resulting level of access to the host and the network
Any mitigating factors in place

Yellow indicates a Low Risk issue that does not by itself allow malicious access to a host but does serve to either unnecessarily increase the attack surface, leak information or allow an attacker to learn information about the host that could help in other attacks; for example a host responding to ping probes or a web server version being revealed in the web server responses.

Orange indicates a Medium Risk issue that that with further action may allow an attacker to compromise a host or an issue that would be rated high but some mitigating factors are in place; for example an FTP service being exposed to the Internet that is vulnerable to automated password guessing attacks or an SQL injection issue that is masked with a Web Application Firewall.

Red indicates a High Risk issue that either has, or would, allow full access to the host in a malicious manner or an issue that MTI feel should be addressed as a top priority; for example a missing Operating System patch that allows a user to exploit a vulnerability and gain full access to the host or an SQL injection issue that allows a user to access the database and all data within it.



EASE OF EXPLOITATION

The number of dots indicates the Ease of Exploitation.

If an issue can be trivially exploited by a user of low skill or freely available tools are in circulation that will exploit the issue then more dots will be given; however, if the attack is very hard to execute or requires certain criteria to be in place then less dots will be allocated.

ISSUE CATEGORISATION

Please note that MTI rate issues based on our understanding of the risk at the time of testing. Any mitigating factors unknown to MTI that serve to reduce the risk cannot be taken in to consideration unless we are made aware of them in good time and can check that they are working correctly, for example; an IDS is present that would usually mask a vulnerability but has been turned off to facilitate testing cannot be used as justification to reduce the severity of an issue unless it can be turned on and MTI can check the issue is mitigated.

Further to this, issue ratings can be increased or decreased based on other issues present within the environment, so it is possible the same issue found on multiple hosts can have different ratings; for example, an FTP server running on a host may be rated as critical if a web server is running on the same host as this may provide a method to upload files and execute them, yet if only an FTP server is present with no other services available it may be rated as Medium.

5.0 ISSUE MATRIX

The following matrix is a collection of all issues located during the testing ranked in order from High to Low. It serves as a simple Vulnerability Register; however, a fully features Vulnerability Register can be produced upon request:

Ref	Location	Severity	Issue	Recommendation	Fixed	Date
APP-02	Application	🟧🟧🟧◼◼◼	The forgotten password form has no protection against automated attacks.	Implement a captcha mechanism or some other system of rate-limiting.		
APP-03	Application	🟧🟧🟧◼◼◼	A single user can have multiple valid access tokens.	Expire previous tokens whenever a new token is issued.		
APP-09	Application	🟧🟧🟧🟧◼◼	Interaction with content in other domains is not restricted.	Implement and enforce a whitelist-based CORS policy.		
APP-15	Application	🟧🟧🟧🟧◼◼	Uploaded documents are not visibly scanned for malware.	Configure on-access scanning for all uploaded files.		
APP-16	Application	🟧🟧🟧◼◼◼	Browser protections are not enforced by the web server.	Implement the missing headers.		
APP-22	Application	🟧🟧🟧🟧◼◼	Visits to the website are not forced to use HTTPS.	Implement and enforce HSTS.		
APP-23	Application	🟧🟧🟧◼◼◼	A weak encryption protocol is in use.	Disable TLS version 1.0.		

APP-24	Application	🟧🟧🟧⬜⬜⬜	Encryption ciphers which can be intercepted are in use.	Disable weak hashing algorithms.		
APP-26	Application	🟧🟧🟧🟧⬜⬜	Executable code can be embedded in uploaded files.	Apply a whitelist filter to all fields.		
APP-28	Application	🟧🟧🟧⬜⬜⬜	Unnecessary information about the application is exposed.	Disable debugging and set a default error message.		
APP-29	Application	🟧🟧🟧⬜⬜⬜	Web server installed on default Windows Partition	It is recommended to install the web server in a non-standard Windows partition		
APP-01	Application	🟨🟨⬜⬜⬜⬜	Content returned from the database in JSON format is not encoded.	Encode content before it is used to generate web pages, unless it is guaranteed to be safe.		
APP-05	Application	🟨🟨⬜⬜⬜⬜	Session cookies may be sent over unencrypted HTTP.	Mark session cookies with the “secure” flag to force them to be sent over HTTPS.		
APP-06	Application	🟨🟨⬜⬜⬜⬜	A user account can be used in multiple locations at once.	Consider preventing simultaneous logins, or enabling session tracking for users.		
APP-07	Application	🟨⬜⬜⬜⬜⬜	Access tokens expose information unnecessary for user sessions.	Configure access tokens to hold only the essential information for session management.		

APP-08	Application	🟨■■■■■	Template files can be accessed without logging in.	Implement access control consistently, allowing anonymous access only where specifically required.		
APP-11	Application	🟨🟨■■■■	Website content can be embedded in frames on other sites.	Enable the clickjacking protection HTTP header.		
APP-12	Application	🟨🟨■■■■	The version of jQuery used by the website is out of date.	Update jQuery, and consider use of a CDN.		
APP-13	Application	🟨🟨■■■■	The version of AngularJS used by the website is out of date.	Update Angular, and consider the use of a CDN.		
APP-17	Application	🟨🟨■■■■	The internal IP address of the web server is exposed.	Apply the URL Rewrite rule to block HTTP/1.0 requests.		
APP-18	Application	🟨🟨■■■■	Unnecessary default content is on the web server.	Remove unnecessary or default content.		
APP-20	Application	🟨🟨■■■■	Unnecessary information about internal systems is exposed.	Review published documents and remove unnecessary or sensitive content.		
APP-21	Application	🟨■■■■■	Unnecessary system information is exposed.	Implement URL Rewrite rules to remove this content.		

APP-25	Application	🟨🟨⬜⬜⬜⬜	The root certificate can be impersonated.	Liaise with the certificate authority to obtain a certificate that uses a stronger hashing algorithm.		
APP-27	Application	🟨🟨⬜⬜⬜⬜	Information entered in forms is not sufficiently validated.	Apply a whitelist filter to all fields.		

6.0 APPLICATION TESTING

Validus provided the following URL to be assessed as part of the testing engagement:

- https://online.talbotuw.com/products/political-violence/malaysia

The following sections are broken down in to test and vulnerability types, with the severity cell indicating the criticality of the issue and the Ease of Exploitation.

6.1. TESTING METHODOLOGY

Web Applications are often the first place a malicious person will visit when targeting a company or organisation for attack. Due to the large volumes of traffic directed towards most web applications, malicious attacks can often go unnoticed until they are successful and adverse effects are experienced. As such, it is imperative to design and secure web applications to a very high standard and monitor all audit logs in order to reduce the risk of compromise.

Many automated scanners and tools are designed to target common vulnerabilities inherent in web based applications; often being run by inexperienced attackers with little or no regard to any damage that may occur to the application as a result of these automated attacks.

The OWASP Top 10 is considered by many to be the most complete and up-to-date resource for current trends in web application attacks; therefore MTI have developed our application testing methodology based around these top ten threats. The table below outlines the core testing that MTI will perform (which also forms the headings for the relevant sections in this report), with their OWASP Top 10 counterparts being adjacent to them in the table:

MTI Report Headings	OWASP Top 10
Injection Attacks (SQL, XML, LDAP)	Injection
Cross Site Scripting	Cross-Site Scripting (XSS)
Authentication and Session Analysis	Authentication and Session Management
Additional Application Manipulation	Insecure Direct Object References
Cross-Site Request Forgery	Cross-Site Request Forgery (CSRF)
Web Server Infrastructure	Security Misconfiguration
Encryption	Insecure Cryptographic Storage
Input validation	Failure to Restrict URL Access
Encryption	Insufficient Transport Layer Protection
Error Handling	N/A
Web Services	N/A
Common Attacks	Non-validated Redirects and Forwards

Please note that the OWASP Top 10 list is designed to be a guide to managing the most common risks to an application and MTI's testing extends well beyond these top ten issues to cover **all** areas of risk to the application; however, time capped tests may not have all issues tested, likewise not all applications are vulnerable to all of the issues so some headings may not appear within the report.

6.2. A WORD ABOUT DATA INPUT VALIDATION

Almost all critical rated issues that affect web applications stem from the application and backend processes not correctly validating user input before it is processed. Over the years MTI have found that this is the area that our clients struggle with the most to get right so we have attempted to explain this issue in detail here.

All input that is sent to an application should be treated as potentially dangerous and be robustly validated before any part of the application acts upon it. Correct validation makes the submission of invalid and malicious data much more difficult for an attacker and helps to prevent casual attackers or automated attack programs from easily launching attacks such as SQL injection or Cross Site Scripting style attacks against the application.

This malicious data will often be encoded or obfuscated to attempt to bypass many filtering and validation routines; therefore the validation routines in use need to check for all possible encodings of this malicious input. For example:

< can be URL encoded as %3C
> can be URL encoded as %3E

Therefore if the validation routines in use are only checking for standard <> characters and are either not decoding the data before it is validated or checking for the encoded version of the characters too, it would be possible to bypass the checks and execute attacks within the application or backend databases.

As a general rule, where possible all non-standard characters such as <>"()'@-- etc should not be permitted in form fields or other URL parameters, as these often allow for injection based attacks against the application and backend infrastructure if they are acted upon directly by any part of the application or database. These checks also need to include any hardcoded values present within the application, such as drop down menus and check boxes that at first glance may not appear to be user editable but are trivial for a causal attacker to alter.

If this data needs to be permitted for legitimate purposes then the application should first detect all non-standard input and then either reject, encode or escape it in a safe manner before it is processed, this can be implemented on a per form field, per parameter or a per application basis but, as mentioned, the checks need robust enough to detect all of the different encodings for all of this unwanted data.

The current best security practises for validation routines recommend that a list of allowed characters is created and all input is referenced against this and handled accordingly, as opposed to using a list of characters to be blocked; also known as whitelisting and blacklisting.

It is important to check the validation routines after any changes are made to the application, especially if new pages or form fields are added, to ensure that the user data submitted via the new areas is still passing through the validation routines.

Finally, many vendors' hardware and software will ship with some default in-built validation mechanisms present; however, these typically only prevent very common and obvious attacks from occurring as the vendor will have no knowledge of your application, so these validation routines should not be relied upon as the only point of validation for the application. As these are default validation routines available to anyone who uses the vendor's software, they are also a very popular target for security researchers who attempt to find ways to bypass them.

1.0 TESTING RESULTS

The following issues all pertain to the https://online.talbotuw.com/products/political-violence/malaysia web application.

1.1. CROSS SITE SCRIPTING

Cross Site Scripting (XSS) in number two on the OWASP Top Ten and usually occurs when user supplied input is not sanitised and stripped from a request or otherwise safely encoded before it is processed.

Usually JavaScript is used in an XSS attack as there is a lot more functionality and attack vectors available to a malicious user when causing a user's browser to execute malicious JavaScript code as opposed to other code such as HTML.

Cross Site Scripting generally falls under two categories: Reflected XSS and Stored XSS.

Reflected XSS is when a specially crafted URL that contains JavaScript is clicked on by a user and the application does not correctly identify that malicious input is contained within the URL, so will render the code in the users browser. This is a 'one time only' attack, meaning that each user must click the URL to be effected by the code; however, only one click may be all that is required to take over a user's web browser and attack the user.

Common attack points to trick a user into executing a reflected XSS attack are to send URL's in an email, upload them to a web forum or to upload / message a user via social media web sites such as Facebook and twitter.

Stored XSS, sometimes called Persistent XSS is when a malicious user manages to store malicious code in a backend data store, such as a database. When a user loads up the stored record, the XSS attack will be executed, often without them knowing it. An example of this would be a user's address on a web site that requires them to make a profile. Each time another user or a web administrator / database administrator views the address, the XSS attack would execute within the users browser, often without their knowledge. This provides a persistent attack vector that once the malicious user has setup, requires no more intervention from them.

If a user is attacked via a successful XSS attack then a malicious user could take over their web browser, capture key strokes, access internal networked hosts, execute client side exploits and potentially access local files on their work stations.

The root cause of XSS attack is the same as data injection attacks; incorrect or poor validation of user supplied input before it is processed by the application.

1.1.1. *INSUFFICIENT OUTPUT ENCODING*

Ref	Location	Severity	Issue	Recommendations
APP-01	Application	🟨🟨⬜⬜⬜⬜	Content returned from the database in JSON format is not encoded.	Encode content before it is used to generate web pages, unless it is guaranteed to be safe.

DESCRIPTION

The web application doesn't apply sufficient filtering or encoding of user-supplied input on the server-side. This enables arbitrary content, such as HTML tags, to be submitted via form fields, and then returned later as data. Typically, this would lead to a common flaw, stored XSS, however, as the data is returned in JSON structures, and processed by AngularJS script before being placed in the final rendered page, no reliable mechanism for exploiting a web browser or website user could be identified.

The following shows how HTML tags are returned in the JSON content for policy 1012:

https://online.talbotuw.com/products/political-violence/malaysia/vsu-terrorism-server/v2/reinsurance/policies/1012

```
{[...]"cedantName":"<i>MTI RI Penn Testing</i>",[...]}
```

RECOMMENDATION

Ensure that any client-supplied input is appropriately filtered and sanitised before it can be sent to a client user's web browser. Ideally, a whitelist approach, such as that already taken on numerical fields, would be used. In cases where this isn't practical, for example where forms must permit arbitrary text to be entered, utilise encoding to supress potentially dangerous characters such as angle-brackets and quote marks.

In the .Net framework, this can be done with the "`HttpServerUtility.HtmlEncode()`" function.

1.2. AUTHENTICATION AND SESSION MANAGEMENT

Authentication and Session management attacks are currently number three on the OWASP Top Ten.

Usually when attacking an application and trying to access sensitive data, an attacker will first look for SQL Injection issues and if this fails they will then examine the authentication and user segregation within the application, as these two areas provide the most likely attack vectors that will result in unauthorised data access.

If authentication can be bypassed then obviously a malicious user will have access to functionality and possibly data they should not have access to; having access to more functionality will allow the malicious user to attempt to locate more attack vectors such as SQL injection issues that would not be accessible from an unauthenticated user perspective.

Typical authentication bypass issues are poor password policies, poor lockout policies, incorrect file / directory permissions or bugs within the code that handles authentication.

Session Management attacks will usually focus on the session cookies that a user will be set once correctly authenticated to the application. Generally a user authenticates once and then is set a cookie value that will be sent with every request the user makes to prove they have previously authenticated. This prevents them from having to enter the username / password each time they view a web page.

If these session tokens can be stolen, predicted, replayed, intercepted or fixed then it is usually possible for a malicious user to access that users account without the need to know valid credentials.

Some session management issues would require other attack vectors to be present, such as Cross Site Scripting to be able to obtain a session cookie, therefore issues may be highlighted within this section that rely on the application being vulnerable in other areas for the attack to work, regardless of if those issues are currently present tor not.

1.2.1. FORGOTTEN PASSWORD FORM DENIAL OF SERVICE

Ref	Location	Severity	Issue	Recommendations
APP-02	Application	🟧🟧🟧⬜⬜⬜	The forgotten password form has no protection against automated attacks.	Implement a captcha mechanism or some other system of rate-limiting.

DESCRIPTION

The website allows users to trigger a password reset process by entering their email address into the following form:

https://online.talbotuw.com/products/political-violence/malaysia/#!/account/forgotpassword

This form doesn't implement any visible form of protection against automated submissions, such as a "reCaptcha" form or rate-limit, and as such could be used by attackers to create a denial of service.

The extent to which a denial of service would impact on Validus is difficult to ascertain without more detailed understanding of the forgotten password process, but at the very least, this page could be targeted to cause a high level of resource use on the web server, affecting its availability for other users.

RECOMMENDATION

Implement a reCaptcha mechanism or alternative protection against automated attacks on this page. Please note that there is already an effective reCaptcha mechanism in place on the login page.

1.2.2. JWT VALIDITY WINDOWS OVERLAP

Ref	Location	Severity	Issue	Recommendations
APP-03	Application	🟧🟧🟧⬜⬜⬜	A single user can have multiple valid access tokens.	Expire previous tokens whenever a new token is issued.

DESCRIPTION

The web application relies on JSON Web Token (JWT) values to manage sessions and to provide a stateless communication between the web application and client browsers. The JWT values are stored by the web browser in the "accessToken" session cookie, and validated whenever the browser requests new resources from the server.

While the JWT value is refreshed with each request, previous values are still considered valid for a period of twenty minutes after first issue. This means that someone obtaining an old or previously used token value, for example through interception, could authenticate to the website as a targeted user at any point up until the value expires. Note that each valid interaction would create a new token value for the attacker to use, and so only the initial interaction needs to occur within the twenty minute window.

This is demonstrated by the web application's support for concurrent sessions, as discussed in APP-06.

RECOMMENDATION

There is no standard mechanism for preventing overlapping sessions when JWT are used for session management, however, the potential for overlapping JWT values to be used for attacks against the website can be minimised by either:

- tracking previously issued tokens and invalidating them after their first use, which is likely to cause a significant increase in server load; or
- implementing a very short expiry time for JWT values that makes interception and replay impractical for all but specifically targeted attacks. A careful balance between usability and security would need to be struck to prevent reducing the user experience, and so an understanding of a standard session timeline may be required to fine-tune this value.

1.2.3. SESSION COOKIES NOT MARKED SECURE

Ref	Location	Severity	Issue	Recommendations
APP-05	Application	🟨🟨◼◼◼◼	Session cookies may be sent over unencrypted HTTP.	Mark session cookies with the "secure" flag to force them to be sent over HTTPS.

DESCRIPTION

The web application stores JSON Web Token (JWT) values in the "accessToken" client-side cookie. These values are used to manage user sessions on the web application, and remove the need for them to re-authenticate during normal or active use.

The cookie is not marked as secure, which means that it will be sent over unencrypted HTTP connections to the server, as well as encrypted HTTPS connections. This is the case even if the server itself doesn't respond to HTTP. As such, the cookie could be intercepted by an attacker on a shared network connection.

RECOMMENDATION

Mark the cookie as secure, and ensure that the website itself is configured to only support encrypted HTTPS communication by enforcing HSTS; as discussed in APP-22.

1.2.4. CONCURRENT SESSIONS PERMITTED

Ref	Location	Severity	Issue	Recommendations
APP-06	Application	🟨🟨◼◼◼◼	A user account can be used in multiple locations at once.	Consider preventing simultaneous logins, or enabling session tracking for users.

DESCRIPTION

It is possible to authenticate to the web application as the same user from two different browsers, and potentially even from two different locations although this could not be confirmed during testing. The two sessions were fully functional, and could be used independently of one another.

This means that an authorised user could have their account compromised without their knowledge.

RECOMMENDATION

Consider implementing a process by which existing user sessions are logged out when they login to a new session, or in which a user is notified that they already have existing sessions elsewhere. Consider also implementing a process of detecting when a single user is logged in from multiple IP addresses, as this could indicate that their account has been compromised.

1.2.5. *JWT CONTAINS POTENTIALLY UNNECESSARY INFORMATION*

Ref	Location	Severity	Issue	Recommendations
APP-07	Application	🟨 ■ ■ ■ ■ ■	Access tokens expose information unnecessary for user sessions.	Configure access tokens to hold only the essential information for session management.

DESCRIPTION

The web application stores JSON Web Token (JWT) values in the “accessToken” client-side cookie. These values are used to manage user sessions on the web application, and remove the need for them to re-authenticate during normal or active use. Typically, these token values should contain only the minimum amount of information required for an authenticated user to interact with the application; all other information needed for business logic should be stored in server-side database or profile record.

The following example shows the values stored in a JWT value for the user “mti.uw”, which had the “underwriter” role on the web application:

Encoded JWT:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJuYW1laWQiOiI3ZGU4NmMyOC1hYjFmLTFiY2EtNGQ1MC0zOW
U3NTM2ZjIzODciLCJ1bmlxdWVfbmFtZSI6Im10aS51dyIsImh0dHA6Ly9zY2hlbWFzLm1pY3Jvc29mdC5jb20vY
WNjZXNzY29udHJvbHNlcnZpY2UvMjAxMC8wNy9jbGFpbXMvaWRlbnRpdHlwcm92aWRlciI6IkFTUC5ORVQgSWRl
bnRpdHkiLCJBc3BOZXQuSWRlbnRpdHkuU2VjdXJpdHlTdGFtcCI6IjQwNjUzMjQ5LTRiN2EtNDc5ZS1hYTdiLTI
yYmI2N2EwMTgxYyIsInJvbGUiOiJVbmRlcndyaXRlciIsImVtYWlsIjoiYmVuLmtlcm5AdmFsaWR1c2hvbGRpbm
dzLmNvbSIsImlzcyI6InZhbGlkdXMiLCJhdWQiOiJBbnkiLCJleHAiOjE1MzA2MDgyNzYsIm5iZiI6MTUzMDYwN
zA3Nn0.s7rgAi5NtO0A29PuslSiIfIWwUpI5IoZ0nS3--J6THk
```

Decoded JWT:

```
Headers = {"typ":"JWT","alg":"HS256"},
Payload = {"nameid":"7de86c28-ab1f-1bca-4d50-
39e7536f2387","unique_name":"mti.uw","http://schemas.microsoft.com/accesscontrolservice
/2010/07/claims/identityprovider":"ASP.NET
Identity","AspNet.Identity.SecurityStamp":"40653249-4b7a-479e-aa7b-
22bb67a0181c","role":"Underwriter","email":"ben.kern@validusholdings.com","iss":"validu
s","aud":"Any","exp":1530608276,"nbf":1530607076}
Signature = "s7rgAi5NtO0A29Pus1SiIfIWwUpI5IoZ0nS3"
```

As shown above, the token includes the connecting user’s email address, their role, and the identity provider in use by the application. Typically, none of these are normally required for a user to interact with the web application, as all three will be values already stored on the web server.

RECOMMENDATION

Consider sanitising the JWT value so it stores only the minimum amount of information required for the web application to function. Typically, this will include a JWT header, a session identifier, the expiry and validity timestamps, and the JWT signature.

1.2.6. *TEMPLATE FILES DON'T REQUIRE AUTHENTICATION*

Ref	Location	Severity	Issue	Recommendations
APP-08	Application	🟨◼◼◼◼◼	Template files can be accessed without logging in.	Implement access control consistently, allowing anonymous access only where specifically required.

DESCRIPTION

The web application is based on the AngularJS framework, which uses HTML templates, combined with JSON data structures, and rendered by client-side JavaScript, to deliver an interactive web application to end users.

Access to web application functionality and data is dependent on the users' roles within the web application, and no mechanism for bypassing this was identified during testing. The base templates, however, could be accessed without logging in, provided that the URLs were known.

While this did not allow access to data, or anonymous interaction with the application in any meaningful sense, it did provide a large amount of information about business logic and the manner in which the web application worked for authorised users. This amount of information is unnecessary for anonymous users, and could assist a targeting attacker with formulation of an effective attack strategy.

RECOMMENDATION

Review the current access control processes, and consider enforcing them across all web application content, not just the interactive components. Ideally, access should only be granted to resources where specifically required.

1.3. ADDITIONAL APPLICATION MANIPULATION

As all applications are designed in different ways and built using different technologies it is not possible to group all attacks into specific categories.

Additional Application Manipulation encompasses a wide range of attacks that do not specifically fit in to any of the attack categories listed within this report. Generally any issues in this section will allow a malicious user some level of access to the application that they should not have or would contribute to a different type of attack.

1.3.1. *INSECURE CORS POLICY*

Ref	Location	Severity	Issue	Recommendations
APP-09	Application	🟧🟧🟧🟧⬜⬜	Interaction with content in other domains is not restricted.	Implement and enforce a whitelist-based CORS policy.

DESCRIPTION

The web application has a Cross-Origin Resource Sharing (CORS) policy that allows two-way asynchronous JavaScript communications from any other domain, including for authenticated sessions. This means that an attacker can create a website with script intended to access and interact with resources on the target web application. These scripts could potentially be used to obtain session cookies or cause an authenticated user's browser to perform actions on the website without the user's knowledge or consent.

RECOMMENDATION

Enforce a CORS policy which whitelists specifically authorised domains. Additionally, disable the use of credentials for cross-domain interaction, to prevent communications from other domains being used to perform privileged activities on the web application.

This can be done by implementing a server-side validation of the "Origin" header as per the following guide:

https://www.w3.org/TR/cors/#access-control-allow-origin-response-hea

1.3.2. *FRAMEABLE RESPONSE ALLOWS CLICKJACKING*

Ref	Location	Severity	Issue	Recommendations
APP-11	Application	🟨🟨⬜⬜⬜⬜	Website content can be embedded in frames on other sites.	Enable the clickjacking protection HTTP header.

DESCRIPTION

The web application permits generated pages to be rendered in an iframe as it doesn't enforce the "X-Frame-Options" HTTP security header. This means that, if an attacker successfully tricks an authorised user into accessing a website under the attacker's control, the user's browser could be manipulated into performing actions on the web application without the user's knowledge or consent.

Please note that the web application was seen to make use of "framebusting" to detect when the page has been rendered in an iframe, however, this can be bypassed by using "double-framing", in which the page is rendered in an iframe on a page which is itself rendered in an iframe on another page.

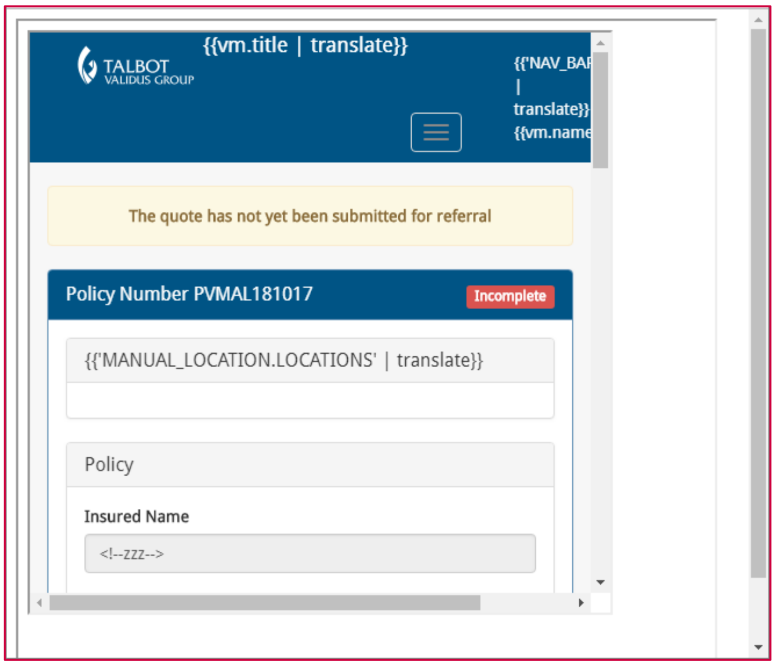
The following shows how this can be done. Please note that the example below breaks some of the interactive functionality on the page, however, a more targeted attack would make use of corrective script to mask the framing, or hide the rendered page entirely from view.

cj2.html:

```
<html><head><title></title></head><body><iframe src="cj.html" width="600px"
height="600px"></iframe></body></html>
```

cj.html:

```
<html><head><title></title></head><body><iframe
src="https://online.talbotuw.com/products/political-
violence/malaysia/#!/reinsuarnce/policy/1017/referral" width="500px"
height="500px"></iframe></body></html>
```



The quote has not yet been submitted for referral

Policy Number PVMAL181017 Incomplete

{{'MANUAL_LOCATION.LOCATIONS' | translate}}

Policy

Insured Name

<!--zzz-->

RECOMMENDATION

Implement clickjacking protection by configuring the "X-Frame-Options" header on the web server for every response as shown below:

```
X-Frame-Options: SAMEORIGIN
```

1.3.3. OUTDATED JQUERY LIBRARY IN USE

Ref	Location	Severity	Issue	Recommendations
APP-12	Application	🟨🟨◼◼◼◼	The version of jQuery used by the website is out of date.	Update jQuery, and consider use of a CDN.

DESCRIPTION

The web application makes use of version 3.2.1 of the jQuery JavaScript library. This is outdated; the current version at the time of writing being 3.3.1.

While there are no specifically exploitable vulnerabilities reported for version 3.2.1, there have been several deprecations and enhancements made in later versions, and so upgrading is generally recommended where not prevented by compatibility issues.

RECOMMENDATION

Consider upgrading to the latest version of jQuery, and consider using a Content Delivery Network (CDN) to always use the latest stable version, rather than hosting jQuery directly on the web server.

Please note that upgrading jQuery may impact on compatibility with AngularJS (APP-13).

1.3.4. OUTDATED ANGULARJS LIBRARY IN USE

Ref	Location	Severity	Issue	Recommendations
APP-13	Application	🟨🟨◼◼◼◼	The version of AngularJS used by the website is out of date.	Update Angular, and consider the use of a CDN.

DESCRIPTION

The web application makes use of version 1.6.8 of the AngularJS JavaScript framework. This is outdated; the current version at the time of writing being 1.7.2.

Versions earlier than 1.7.2 have been reported as vulnerable to cross-site scripting, however, the functions reported as being affected did not appear to be in use by the web application. Additionally, there have been several deprecations and enhancements made in later versions, and so upgrading is generally recommended where not prevented by compatibility issues.

RECOMMENDATION

Consider upgrading to the latest version of Angular (note AngularJS is a legacy framework and has been superseded by Angular), and consider using a Content Delivery Network (CDN) to always use the latest stable version, rather than hosting AngularJS directly on the web server.

Please note that upgrading AngularJS may impact on compatibility with other JavaScript libraries such as jQuery (APP-12).

1.4. WEB SERVER INFRASTRUCTURE

Web servers provide a hosting platform for web applications; therefore if any vulnerabilities are present within the web server they could be used to access the application and / or database, regardless of if the application is secure or not.

Infrastructure attacks against web server can also provide a stepping stone into the internal network, often bypassing any firewall restrictions in place.

MTI will assess the web servers that the application is running on to ensure no common misconfigurations are present; however, this test may be omitted if the applications are running in a test environment as the results would not be relevant to the test.

1.4.1. POTENTIALLY INSUFFICIENT MALWARE PROTECTION

Ref	Location	Severity	Issue	Recommendations
APP-15	Application	🟧🟧🟧🟧⬜⬜	Uploaded documents are not visibly scanned for malware.	Configure on-access scanning for all uploaded files.

DESCRIPTION

There was no visible protection against the uploading of files containing malware through the web application's "new quote" section.

As a proof of concept example, a "new quote" template spreadsheet was uploaded with the "Property Name" field set to the EICAR (European Institute for Computer Antivirus Research) string. This string is completely benign, but most commercial anti-malware software will identify it. As such it is used diagnostically to determine whether anti-malware software is present and running.

As the file was accepted, processed, and the contents immediately used to populate the property details on the "New Quote" form, it is believed that on-access scanning of uploaded files is not performed. This means that spreadsheet files containing malware, for example in the form of macros, could also be uploaded to the server. Due to the potential for this to result in a denial of service, or a loss of integrity of the server, this was not attempted beyond the proof of concept.

RECOMMENDATION

Configure on-access scanning for all uploaded files on the web server, supported by existing filetype verification and data validation.

1.4.2. MISSING HTTP SECURITY HEADERS

Ref	Location	Severity	Issue	Recommendations
APP-16	Application	🟧🟧🟧⬜⬜⬜	Browser protections are not enforced by the web server.	Implement the missing headers.

DESCRIPTION

The web server is not configured to send HTTP security headers to connecting browsers as part of its responses. These headers, discussed in more detail below, help to protect client's web browsers against attacks targeted against them within the context of the web application. They also demonstrate that the web application and web server have been configured with good security practice in mind.

The key headers that were identified as missing, and are note discussed elsewhere, are as follows:

- Browser Cross-Site Scripting Protection ("**X-XSS-Protection: 1; mode=block**"):
  This header instructs the web browser to enable extra protection against cross-site scripting, although this can impact on compatibility with websites that rely heavily on JavaScript.
- Content Sniffing Prevention ("**X-Content-Type-Options: nosniff**"):
  This header instructs the web browser to treat content strictly in accordance with the type of data the filetype or "**Content-Type**" header indicates, and so not to parse HTML or JavaScript contained in files such as text or JSON files. This provides additional protection against vulnerabilities such as cross-site scripting.
- Content Security Policy ("**Content-Security-Policy**"):
  This header instructs web browsers in how to handle specific resources by listing authorised sources of elements such as scripts, styles, images, and embedded content. If content is included in a web page that is not authorised by the content security policy, it will not be rendered. By default, this includes inline scripts, such as those which are commonly targeted in cross-site scripting attacks.

More information on HTTP security headers can be obtained from https://securityheaders.io, which also provides free scanning and tutorials relating to web server security headers.

Please note that some of these headers are present when accessing the web server over unencrypted HTTP.

RECOMMENDATION

Implement the HTTP security headers on the web server as listed above. Please note that content security policy headers in particular require care, testing and ongoing maintenance to ensure that they do not impact on legitimate web application functionality, particularly whenever AngularJS or jQuery libraries are updated.

1.4.3. *WEB SERVER EXPOSES INTERNAL IP ADDRESS*

Ref	Location	Severity	Issue	Recommendations
APP-17	Application	🟨🟨◼◼◼◼	The internal IP address of the web server is exposed.	Apply the URL Rewrite rule to block HTTP/1.0 requests.

DESCRIPTION

The web server responds to requests using HTTP version 1.0 which are sent over TCP port 80 with the web server's internal IP address, as shown below.

Request:

```
GET / HTTP/1.0
```

Response:

```
[...]Location: https://10.200.151.72/[...]
```

This information is potentially of use to an attacker attempting to exploit weaknesses in CORS policies (see APP-09), attempting to spoof connections to the server to gain access to private administrative functionality, or as part of a larger attack strategy against the corporate network.

RECOMMENDATION

This is a well-known issue on Microsoft Internet Information Services (IIS) web servers, and the primary mechanism for addressing it is by installing the URL Rewrite extension and then implementing the following rule to reject HTTP/1.0 connection requests:

```
<system.webServer><rewrite><rules>
<rule name="RequestBlockingRule1" patternSyntax="Wildcard" stopProcessing="true">
<match url="*" />
<conditions><add input="{SERVER_PROTOCOL}" pattern="HTTP/1.0" /></conditions>
<action type="AbortRequest" />
</rule></rules></rewrite></system.webServer>
```

Additionally, block access to port 80 (HTTP) on this web server and implement HSTS as discussed in APP-22.

1.4.4. DEFAULT ANGULARJS FILE PRESENT ON SERVER

Ref	Location	Severity	Issue	Recommendations
APP-18	Application	🟨🟨⬜⬜⬜⬜	Unnecessary default content is on the web server.	Remove unnecessary or default content.

DESCRIPTION

The installation of AngularJS on the web server contains the default "mock" file, and may contain other AngularJS files that are not required for the web application to function.

This can suggest that the web application has not been completely hardened, and may include default or unused content that exposes exploitable vulnerabilities.

The content of the "mock" file is shown below:

https://online.talbotuw.com/products/political-violence/malaysia/node_modules/angular-mocks/angular-mocks.js

```
angular.mock.$Browser = function() {
var self = this;
this.isMock = true;
self.$$url = 'http://server/';
self.$$lastUrl = self.$$url; // used by url polling fn
self.pollFns = []; [...] }
```

RECOMMENDATION

Remove the mock file and any other content that is not directly used or required for the web application to function. Review change control and configuration management process to ensure that the removal of default content, and hardening of web applications, is included as a quality assurance step for all production web servers.

1.4.5. FILE AND DOCUMENT INFORMATION LEAKAGE

Ref	Location	Severity	Issue	Recommendations
APP-20	Application	🟨🟨⬜⬜⬜⬜	Unnecessary information about internal systems is exposed.	Review published documents and remove unnecessary or sensitive content.

DESCRIPTION

The following files contained information relating to internal systems, personnel, and software which could be of interest as part of a targeted attack against Validus. Please note that a comprehensive search for such information was not undertaken, and so other examples may be present that are not shown below.

https://online.talbotuw.com/products/political-violence/malaysia/assets/images/logo.svg

```
<!-- Generator: Adobe Illustrator 15.1.0, SVG Export Plug-In . SVG Version: 6.00 Build 0)  -->
```

https://online.talbotuw.com/products/political-violence/malaysia/assets/images/Logo_White.svg

```
<!-- Generator: Adobe Illustrator 22.0.1, SVG Export Plug-In . SVG Version: 6.00 Build 0)  -->
```

https://online.talbotuw.com/products/political-violence/malaysia/app/partials/reinsurance/ScheduleTemplateMalaysia.xlsx

```
Author: Hua Zheng
Default Printer: \\CAWATDC01\CAWATMAIN
```

As shown in the extracts above, the information exposed includes, specific software versions, employee names and internal network hostnames pertaining to Validus.

RECOMMENDATION

Remove potentially sensitive information from any files or documents that are likely to be accessed from outside of the organisation. In the case of SVG image files, the information can be removed by reviewing the file in an XML editor. For Office files, such as XLSX spreadsheets, use the "Inspect Document" function to find and remove potentially sensitive content before publishing the file.

1.4.6. HTTP RESPONSE HEADERS EXPOSE SYSTEM INFORMATION

Ref	Location	Severity	Issue	Recommendations
APP-21	Application	🟨 ■ ■ ■ ■ ■	Unnecessary system information is exposed.	Implement URL Rewrite rules to remove this content.

DESCRIPTION

The web server returns unnecessary system information in HTTP response header, which indicates the version of the operating system running as Windows Server 2012 R2. The headers are shown below:

```
Server: Microsoft-IIS/8.5
X-Powered-By: ASP.NET
```

RECOMMENDATION

Implement a URL Rewrite rule on the web server to remove these headers, as follows. Please note that these require the IIS URL Rewrite Extension to be installed.

```
<system.webServer>
<httpProtocol><customHeaders>
<remove name="X-Powered-By" />
</customHeaders></httpProtocol>
<rewrite><outboundRules>
<rule name="Modify Server header value" patternSyntax="Wildcard">
<match serverVariable="RESPONSE_SERVER" pattern="*" />
<action type="Rewrite" value="Server" />
</rule></outboundRules></rewrite>
</system.webServer>
```

1.5. ENCRYPTION

Insufficient Transport Layer encryption is currently number eight on the OWASP Top Ten list of common web application vulnerabilities.

One of the main methods of securing application related data is to ensure that robust and correct encryption is in use. Usually Secure Sockets Layer (SSL) or Transport Layer Security (TLS) are used to encrypt data between a client and server; however, these protocols rely on strong encryption ciphers being chosen when the secure tunnel is established to ensure the data is robustly encrypted. If weak SSL or TLS ciphers are enabled and a HTTPS connection opts to use them, then any data intercepted could be decrypted to reveal the plain text payload.

In addition to ensuring strong ciphers are used, valid certificates should be installed that are in-date, issued by a trusted Certificate Authority and issued to the correct server. If web server certificates are not correctly issued and installed then users will encounter certificate warning messages when browsing to the web site; therefore, over time they will become accustomed to these warnings and not be able to realise when a man-in-the-middle attack is occurring, negating the security benefits offered by certificates.

1.5.1. HSTS NOT ENABLED

Ref	Location	Severity	Issue	Recommendations
APP-22	Application	🟧🟧🟧🟧⬜⬜	Visits to the website are not forced to use HTTPS.	Implement and enforce HSTS.

DESCRIPTION

The web application doesn't enforce the use of encrypted HTTPS as it doesn't include the HTTP Strict Transport Security (HSTS) header in responses. The HSTS header instructs browsers that all connections to the web application must occur over encrypted HTTPS, and so, providing an initial HTTPS connection had been made, browser will refuse to connect over HTTP for the period specified in the "max-age" parameter of the HSTS header.

By not enforcing this header, an authorised user of the website could be tricked into attempting to visit the web application over clear-text HTTP, by for example through clicking on a link, and their web browser would attempt to do so. This will work even though the web application itself is only served over HTTPS; using a 302 redirect for any connections made over HTTP.

RECOMMENDATION

Implement the HSTS header as shown below. The value specified for the "max-age" parameter is equivalent to a period of six months, however, a shorter period may be preferred while testing this implementation.

```
Strict-Transport-Security: max-age=1576800
```

1.5.2. TLS VERSION 1.0 SUPPORTED

Ref	Location	Severity	Issue	Recommendations
APP-23	Application	🟧🟧🟧⬜⬜⬜	A weak encryption protocol is in use.	Disable TLS version 1.0.

DESCRIPTION

The web server supporting the web application permits connections made using version 1 of the Transport Layer Security (TLSv1) encryption protocol. TLSv1 has been shown to be vulnerable to interception and decryption attacks which could lead to exposure of sensitive information or theft of session cookies.

RECOMMENDATION

Disable support for TLSv1.

1.5.3. TLS CIPHER SUITES SUPPORT WEAK HASHING ALGORITHMS

Ref	Location	Severity	Issue	Recommendations
APP-24	Application	🟧🟧🟧⬜⬜⬜	Encryption ciphers which can be intercepted are in use.	Disable weak hashing algorithms.

DESCRIPTION

The web server supporting the web application permits connections to be made which use version 1 of the Secure Hashing Algorithm (SHA1). This algorithm has been shown to be vulnerable to collision attacks which allow the production of spoofed signatures and messages. This could permit connections to be intercepted and information to be leaked.

Additionally, SHA1 is considered to be a deprecated hashing algorithm, and as such many software vendors are removing support for it, including in web browsers.

The cipher suites supported by the web server are shown below.

```
DHE-RSA-AES128-SHA      Kx=DH      Au=RSA     Enc=AES-CBC(128)   Mac=SHA1
DHE-RSA-AES256-SHA      Kx=DH      Au=RSA     Enc=AES-CBC(256)   Mac=SHA1
ECDHE-RSA-AES128-SHA    Kx=ECDH    Au=RSA     Enc=AES-CBC(128)   Mac=SHA1
ECDHE-RSA-AES256-SHA    Kx=ECDH    Au=RSA     Enc=AES-CBC(256)   Mac=SHA1
```

RECOMMENDATION

Remove support for cipher suites that use SHA1 as their hashing algorithm, instead using stronger cipher suites such as those using SHA256.

1.5.4. CA CERTIFICATE SIGNED WITH WEAK HASHING ALGORITHM

Ref	Location	Severity	Issue	Recommendations
APP-25	Application	🟨🟨⬜⬜⬜⬜	The root certificate can be impersonated.	Liaise with the certificate authority to obtain a certificate that uses a stronger hashing algorithm.

DESCRIPTION

The web application uses a digital certificate which has been issued by the DigiCert Certificate Authority. While the certificate itself is valid, the root certificate, which is at the top of the certificate tree is signed using a deprecated protocol. This protocol, Secure Hashing Algorithm version 1 (SHA1), has been shown to be vulnerable to collision attacks and spoofing, which could allow an attacker to issue a fake certificate which appears valid to browsers.

The affected certificate is shown below:

```
Subject: C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Global Root CA
             Signature Algorithm: SHA-1 With RSA Encryption
```

RECOMMENDATION

Liaise with the certificate authority to re-issue their root certificate with a stronger hashing algorithm. Please note that this will likely be resisted, as they would potentially need to reissue a large number of digital certificates across their customer base, at significant cost.

1.6. INPUT VALIDATION

Input Validation concerns the checking of all user supplied data before it is processed by the application.

Almost all serious web application attacks occur due to the lack of correct data validation, therefore it is beneficial to carry out these checks in several locations to decrease the scope of malicious data being processed.

The first logical place to validate user supplied input is within the users' web browser using client side scripts. This not only removes some of the processing load from the backend server but it helps prevent low-skilled attackers from sending malicious input and also legitimate users accidently sending unintended input to the application.

It should be noted that reasonably skilled attackers can disable client side validation scripts or alter the data with a proxy server, so client side validation alone should not be relied upon to validate data and instead it should supplement robust server side validation.

1.6.1. EXCEL FILE GENERATION PERMITS EXECUTABLE CODE

Ref	Location	Severity	Issue	Recommendations
APP-26	Application	🟧🟧🟧🟧⬜⬜	Executable code can be embedded in uploaded files.	Apply a whitelist filter to all fields.

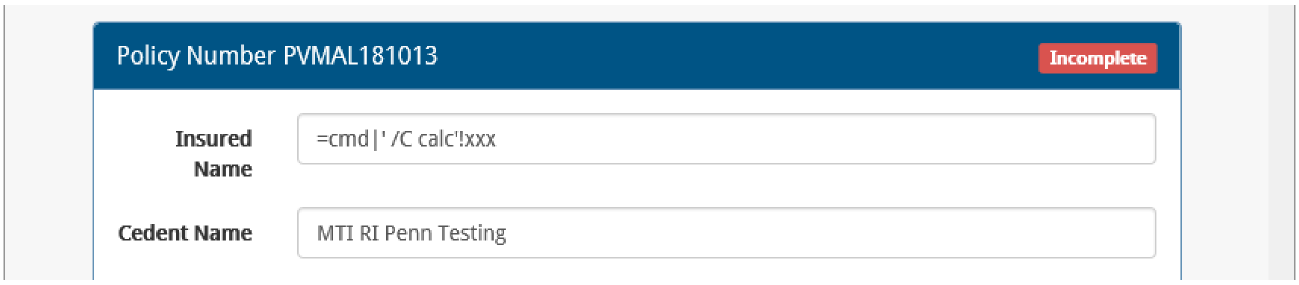
DESCRIPTION

The values entered into the "New Quote" forms support executable code which is echoed into the downloadable spreadsheets attached to the "Quote List" and "Quote Review" forms.

The following proof of concept shows how this can be exploited to load the Windows Calculator application when a downloadable report is opened by a user with the Underwriter role.

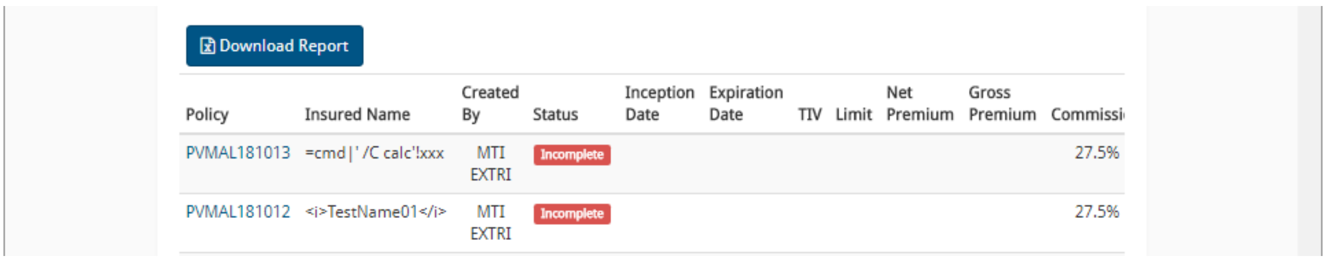
1) As an agent, start a new quote, setting the "Insured Name" to the following and entering default values for all other fields:

```
=cmd|' /C calc'!xxx
```


Policy Number PVMAL181013

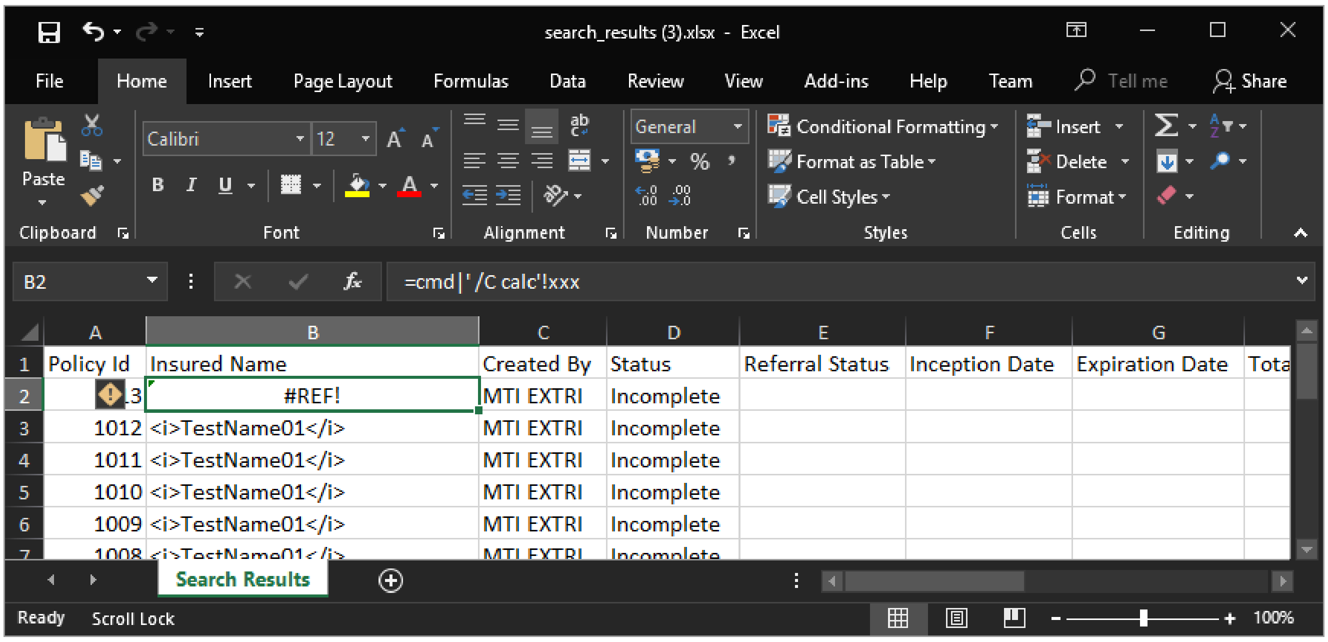
Insured Name	=cmd|' /C calc'!xxx
Cedent Name	MTI RI Penn Testing

2) Submit the quote for referral.
3) As an underwriter, browse to the quote list page:



Policy	Insured Name	Created By	Status	Inception Date	Expiration Date	TIV	Limit	Net Premium	Gross Premium	Commissi
PVMAL181013	=cmd|' /C calc'!xxx	MTI EXTRI	Incomplete							27.5%
PVMAL181012	<i>TestName01</i>	MTI EXTRI	Incomplete							27.5%

4) Click on "Download Report" and open the downloaded Excel file.



=cmd|' /C calc'!xxx

	A	B	C	D	E	F	G	
1	Policy Id	Insured Name	Created By	Status	Referral Status	Inception Date	Expiration Date	Tota
2	3	#REF!	MTI EXTRI	Incomplete				
3	1012	<i>TestName01</i>	MTI EXTRI	Incomplete				
4	1011	<i>TestName01</i>	MTI EXTRI	Incomplete				
5	1010	<i>TestName01</i>	MTI EXTRI	Incomplete				
6	1009	<i>TestName01</i>	MTI EXTRI	Incomplete				
7	1008	<i>TestName01</i>	MTI EXTRI	Incomplete				

5) The executable code will run and launch "`calc`", which is the Windows Calculator application.

Please note that, by default, later versions of Microsoft Office and Microsoft Windows will warn the user before executing code embedded in this manner, but it is possible to bypass these warnings in some instances.

RECOMMENDATION

Apply a whitelist-based filter to all form fields and all quote data, allowing only expected characters, and removing or encoding any others, based on the data type. It is unusual, for example, for names to contain characters such as the following, and so these can generally be treated as suspicious:

```
= | ! / < > { } [ ] ( )
```

1.6.2. *INSUFFICIENT INPUT VALIDATION IN QUOTE FORMS*

Ref	Location	Severity	Issue	Recommendations
APP-27	Application	🟨🟨⬜⬜⬜⬜	Information entered in forms is not sufficiently validated.	Apply a whitelist filter to all fields.

DESCRIPTION

It is possible to submit any character in any of the "text" fields used to populate quote details produced through the web application, by sending the values directly to the server as "`PUT`" requests; thus bypassing client-side JavaScript validation. These values are then echoed back in later responses within JSON data structures, and displayed, with encoding, in web pages elsewhere on the application.

The following example shows how this can be done, by entering an HTML comment in place of the insured person's name.

```
PUT /products/political-violence/malaysia/vsu-terrorism-
server/v2/reinsurance/policies/1017 HTTP/1.1
{[…]"clientName":"<!--zzz-->","clientStreetName":"<!--zzz-->"[…]}
```

This value is echoed back in returned JSON data structures for this quote:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
[…]
{[…]"clientName":"<!--zzz-->","clientStreetName":"<!--zzz-->",[…]}
```

In most circumstances, this would lead to cross-site scripting attacks, and would allow theft of session cookies or embedding of other malicious content, however, due to the inclusion of the HTTP response header "`Content-Type: application/json; charset=utf-8`", this attack would generally only be successful against older browsers that attempt to render the JSON data as HTML.

At a minimum, this issue does demonstrate a risk to the integrity of the data being processed.

RECOMMENDATION

Apply a whitelist-based filter to all form fields and all quote data, allowing only expected characters, and removing or encoding any others, based on the data type. It is unusual, for example, for names to contain characters such as the following, and so these can generally be treated as suspicious:

```
= | ! / < > { } [ ] ( )
```

1.7. ERROR CONTROL

Error Control does not form part of the OWASP Top Ten; however, it is MTI's experience that not handling error conditions correctly can often lead to applications being compromised in many ways, such as via SQL injection or software exploits, which do form part of the OWASP Top ten.

All significant attacks against an application will first look to cause an error condition as this will often indicate if any data validation is being conducted or not, if the application has been designed with security in mind and what technologies are in use. For example a prelude to most SQL Injection attacks is to enter a ' into form fields. If this request is directly processed within a database then it will usually cause an SQL related error that is displayed to the attacker. This would indicate that the data they are entering is being directly acted upon and the resultant error message can often include verbose information, such as the SQL requests being processed at the time of the error and the detailed reason as to why the request could not be processed.

It is important that all errors are recognised by the application and a generic custom error page is displayed to a user that does not reveal why the error occurred, where the error occurred or any other technical details such as software versions or source code.

1.7.1. DETAILED ERROR MESSAGES EXPOSE SYSTEM INFORMATION

Ref	Location	Severity	Issue	Recommendations
APP-28	Application	🟧🟧🟧⬜⬜⬜	Unnecessary information about the application is exposed.	Disable debugging and set a default error message.
APP-29	Application	🟧🟧🟧⬜⬜⬜	Web server installed on default Windows Partition	It is recommended to install the web server in a non-standard Windows partition

DESCRIPTION

The web application is configured to return debug messages in response to certain error conditions. These messages include detailed, and potentially sensitive information about the application and the web server, such as that shown in the example below which is caused by entering an invalid username and password.

Please note that the error message is not visible on the rendered web page, but requires interception of the returned JSON data structure, for example with a web proxy application.

Request:
```
POST /products/political-violence/malaysia/vsu-terrorism-server/v2/users/login HTTP/1.1
[…]
{"vrUsername":"test","password":"test","recaptchaResponse":"A"}
```

Response:
```
HTTP/1.1 500 Internal Server Error
[…]

{"message":"An error has occurred.","exceptionMessage":"invalid-input-
response","exceptionType":"System.Security.Authentication.AuthenticationException","sta
ckTrace":"   at
Terrorism.Web.Authentication.AuthenticationRequestHandler.<Handle>d__5.MoveNext() in
```

```
C:\\Users\\jason.meckley\\source\\repos\\terror-worldwide\\worldwide-application-
server\\Terrorism.Web\\Authentication\\AuthenticationRequestHandler.cs:line 35\r\n---
[…]
```

As is highlighted above, the error message also shows that the web server has also been installed in a standard windows partition. With this information, a malicious user would find it much easier to access and modify web page files if a vulnerability such as Directory Traversal was found and successfully exploited.

RECOMMENDATION

Disable debugging behaviour on the web application, prevent any detailed error message content from being sent to clients, including in the form of JSON data structures, and implement a default error message that is returned if an unexpected error occurs.

Also, it is recommended to re-install the web server on a non-standard Windows partition to reduce the risk of a malicious user gaining access to the System32 directory on the C:\ partition via a directory traversal attack.

Head Office
Riverview House
Weyside Park
Catteshall Lane
Godalming
GU7 1XE

Tel:	0845 888 6060
Fax:	0845 888 6061
Web:	www.mti.com

About MTI
MTI is a leading provider of data centre storage, virtualisation and security solutions and services, across both public and private cloud environments. With offices in the UK, Germany and France it services over 3000 customers across the world. MTI work with their customers to focus on their data – ensuring it is secure and always available whether in a public or private cloud.

MTI deliver a full consultancy service, ranging from Data Protection Act issues through to ISO 27001 Compliance, and are qualified to conduct CHECK and CREST level penetration and application testing services. MTI can also help clients achieve PCI DSS compliance through our team of PCI Qualified Security Assessors (QSA).