Security Penetration Test of
HIE Portal for A CUSTOMER IMPLEMENTION

Services provided to:
[LOGO(s) of company providing service to]

Version –V1.0

V1 – February 13th, 2014

Prepared By:
Denis Calderone
TBG Security

Presented To:
Justin Case
ABC Heath

TBG SECURITY

CONFIDENTIALITY

In no event shall TBG Security be liable to anyone for special, incidental, collateral or consequential damages arising out of the use of this information.

This document contains information, which is confidential and proprietary to TBG Security and ABC Health. Extreme care should be exercised before distributing copies of this document, or the extracted contents of this document. TBG Security is authorizing our point of contact at ABC Health. to view and disseminate this document as he/she sees fit in accordance with ABC Health data handling policy and procedures. This document should be marked “CONFIDENCIAL” and therefore we suggest that this document be disseminated on a “need to know” basis.

Address questions regarding the proper and legitimate use of this document to:

TBG Security

31 Hayward Rd
Franklin, MA 02038

Attention: Contracts Manager

DISCLAIMERS

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and as such it is possible that something in the environment could have changed since the tests reflected in this report were run. Also, it is possible that new vulnerabilities may have been discovered since the tests were run. For this reason, this report should be considered a guide, not a 100% representation of the risk threatening your systems, networks and applications.

TBG SECURITY

1 Contents

TBG SECURITY

2 Purpose

ABC Health has asked TBG Security to perform a detailed security examination of their Health Information Exchange for one of their customers, *[A custom implementation]* (HIE Portal). This web based portal was in production at the time of the testing, and we were provided access to a test / staging system.

This testing effort took place in January and February of 2014, and concluded on February 12th 2014. Some preliminary findings were provided under separate cover, and this report is being presented to show the full results of our testing efforts and to make recommendations where appropriate.

3 Scope

The scope of this review was limited to a single Internet facing web application portal. This is an HIE application and the specific instantiation of the portal we were asked to test was for the STATENAME Health Network. The application is Internet facing and requires standard username and password identity elements for secure access. The landing page to the application under review was at the following addresses:

Application	Authentication Landing Page
ACUSTOMER HIE Portal	https://staging.ACustomerhie.org/concerto/Login.htm

Our testing included both unauthenticated as well as authenticated testing. For the purpose of our testing we were provided with 4 unique accounts for the HIE Portal. These accounts were used to test the application's internal security controls. These accounts are explained in the table below.

Account name	Access Level	Group Membership
penlevel1	Level 1 View	Users, Non eRX User
penlevel2	Level 2 View	Users, HIE Users, Non eRX User
penlevel3	Level 3 View	Users, HIE Users, Non eRX User
penlevel4	Level 4 View	Users, HIE Users, Non eRX User

Important Note: Some features of the application were unavailable in the staging system we were testing. For example, the search function, which is used for user lookups as part of the messaging application, was not able to connect to its database. This can be seen in the provided screenshot. The error is produced when submitting a POST to the /foo/sqlSearch/SQLResults.htm page.

For future assessments, it is recommended that the application be configured completely and all functionality is proven to be in working order.

Figure 1 - Database error during user lookup

4 Summary of Findings

In performing a detailed application penetration study against ABC Health's HIE Portal application, TBG security identified several issues of concern, but overall found the application to be built around a solid security model. Throughout this report we provide brief descriptions of each testing category and provide more detailed where our findings were negative.

The below table shows a breakdown of the vulnerabilities identified based on category and severity of risk. This table is followed by a detailed breakdown outlining each category. In the table below, a vulnerability listed under 'Pending' has been reported, where a vulnerability listed under 'Fixed', is a vulnerability that has been satisfactorily mitigated.

Figure 2 - Findings Matrix

Vulnerabilities tallied by Risk rating						
Testing Category	High		Medium		Low	
	Fixed	Pending	Fixed	Pending	Fixed	Pending
Web Site Pilfering						
Files Guessing attacks						
Modifying inputs and Parameter Tampering		1				
Bypassing client side validation		1				1
Hidden field identification and tampering						
Cookie Abuse						
Session Hijacking						
URL Jumping						
Cross Site Scripting						
Directory browsing						
SQL Injection						
Functional Design Issues				1		
System & Software vulnerabilities				2		

4.1 Web Site Pilfering

Often, attackers will gain much information simply by what is stored in the content of the web site files that are transferred to the client's browser. We spidered the HIE Portal application to make certain we understood the layout of the application before we started any actual attacks. We used regular expressions to search through the body of the html and java script to identify any information that might be useful to an attacker.

We searched for many common issues including:

- Unnecessary and revealing programmer comments (none found)
- IP addresses (none found)
- Email addresses (none found)
- Raw SQL queries (none found)
- Database connection strings (none found)
- Hidden Fields (none found)

Conclusion:
We performed full text searches of crawl results looking for sensitive information within the HTML code. These tests did not reveal anything that would be of use to an attacker.

4.2 File Guessing attacks

It is sometimes possible to find interesting content on a web site simply by "snooping" around. Sometimes there are backup files of older versions of live code, or perhaps vulnerable sample application pages left on the web site. When accessing sensitive patient data, this application relies on dynamic tokens that change with each request. This behavior makes fuzzing for patient data an impractical test case, although we did still test for common file names using tools such as Burb, DirBuster and Acunetix.

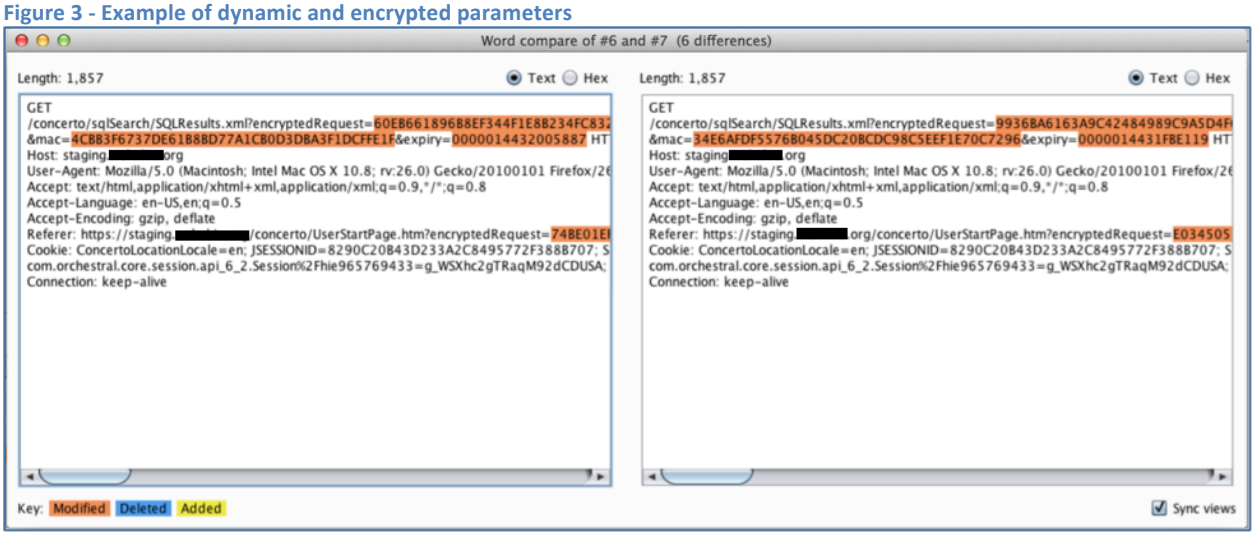
Conclusion:
We attempted various URL brute-force testing for common file names but none were successful in identifying any hidden or otherwise undisclosed files. We ran Burp, DirBuster and Acunetix scans in search of useful files, but did not succeed in identifying anything, which would aid an attacker.

4.3 Modifying input choices and Parameter Tampering

Web applications often pre-populate variables for users either based on the user's identity, pre populated values in hidden fields, or as a result of user selection from a list. The assumption is that these values will be presented to the server in a controlled state; however, it is possible to intercept the client initiated GET or POST and change these values. Since there is an assumption of trust in this process, developers sometimes treat this client provided input with less scrutiny then input directly typed by the user. We therefore are very interested in input generated on the client side of the connection, and spend time tampering with those inputs to see if we can trick the application into bypassing certain authorization controls. This attack method is commonly referred to as Parameter Tampering.

The HIE Portal application allows pages to send 'encrypted' requests via a series of different pages that take a 'encryptedRequest', 'expiry', and 'mac' parameter. The contents of the parameters are string representations of hexadecimal values and are generated on the server side, then passed through the browser via a HTTP 302 redirect, and passed to the requested page. The mechanism appears to be a means of mitigating against injecting arbitrary parameters into requests sent to several pages within the application. The encrypted strings are dynamic in that they are unique each time a page is generated. An example of this can be seen in the screen shot below where the differences in the parameters are being highlighted. Both requests are for the SQLResults.xml page, and each requests was for the same patient record, and each were generated using the same login session:

Figure 3 - Example of dynamic and encrypted parameters



While the encryption mechanism was not uncovered during the assessment, a motivated attacker may focus on discovering a means of decrypting the strings and expanding the potential attack surface against the application. The encrypted requests are never decrypted in the browser and as a result, the browser never modifies the contents of the requests. The contents are generated and processed exclusively on the server side, simply using the browser as a means of passing the strings from one page to another. Further evaluation of the application would be necessary to determine the strength of the

encryption mechanism.

4.3.1 Issue 1: Message Disclosure Vulnerability

Risk:	**HIGH** Successful attack could result in ePHI disclosure
Complexity:	**HIGH** Attack requires authenticated access and access to valid encrypted MessageIDs

Summary:

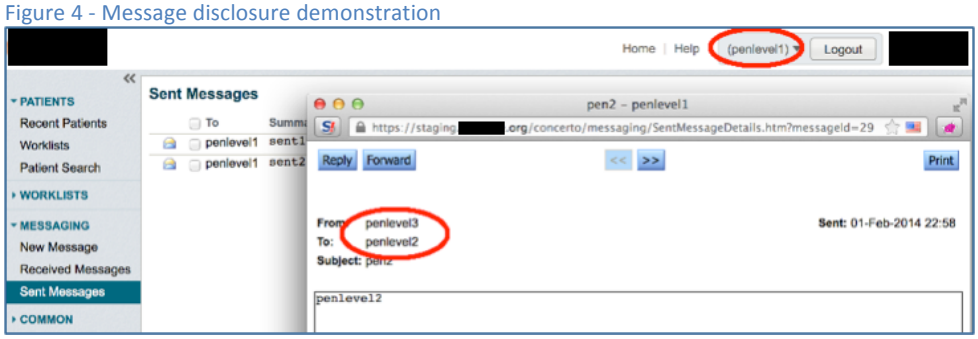
The application provides a user to user messaging mechanism which allows users of the application to send messages to other users. The messages are a basic form of email, taking a destination username, importance level, subject, and message.

The messaging implementation contains a vulnerability that allows users of any privilege level to view any message sent through the messaging mechanism, regardless of whether the user was an intended recipient or sender of the message.

We can demonstrate this by performing the following steps:

1. Log in with valid credentials to the application, using the 'penlevel1' credentials
2. Click on the 'Messaging' menu option
3. Click on the 'Sent Messages' option
4. Follow the following link:
    a. https://staging.ACustomerhie.org/concerto/messaging/SentMessageDetails.htm?messageId=397f5fdc-1b14-4808-bead-70accb6acdab
5. Note that neither username in the 'From:' and 'To:' fields are the current user. The screen shot below demonstrates this nicely.

Figure 4 - Message disclosure demonstration



Details:

The messaging implementation allows users to send messages to other users of the application and to view received and sent messages from the current user account. The 'Sent Messages' menu option displays a list of messages sent from the current user account and allows users to view the entire message detail, reply to the message, forward the message, or print the message.

When viewing the details of a sent message, via the 'Sent Messages' menu option, the application directs the browser to the 'SentMessageDetails.htm' page, with 'messageId' as a parameter. The 'messageId' parameter contains a unique identifier, which is a 128-bit hexadecimal value that

identifies each unique message.

When the 'SentMessageDetails.htm' page is opened with a valid 'messageId' entry, the entire contents of the message is returned and can be viewed by the current user. For example, the following link will open a message from 'penlevel3' to 'penlevel4':

/concerto/messaging/SentMessageDetails.htm?messageId=5cf8173a-47f5-419b-b243-504e7101c4ba

When the application loads the 'SentMessageDetails.htm' page, the application fails to validate that the current user has the correct privileges to view the message identifier requested. As a result, a user with access to the application can view any message on the system, regardless of whether they have the correct permissions or were included in the 'To:' field of the message.

In order to exploit the vulnerability, an attacker must have a valid 'messageId'. The 'messageId' entries are fairly long and random, making brute forcing of the identifiers impractical, but possible. The more practical attack would be to discover an easy means of disclosing message identifiers, either for a specific user or the entire system. While no vulnerabilities allowing for message disclosure were discovered in this test, a clever attacker may leverage other methods to disclose the seemingly innocuous message identifiers.

Recommended Resolution:
TBG Security recommends that, when accessing messages, the application validate that the current logged in user has proper permissions to view the message, is the sender, or is an intended recipient of the message.

Conclusion:

Most client requests to the server are done using parameters, which are encrypted and dynamic in nature. The crypto algorithm used was never revealed to us. An exception to this norm can be found in the Messaging application and this we have flagged as a vulnerability that we feel should be addressed.

4.4 Bypassing client side validation

Validating user input is an important security control which must be performed or a web application will be susceptible a large range of injection based attacks and potentially authentication or authorization bypass attacks. There are 2 methods of validation checks that a developer can employ. The first is to check input within the client via the HTML source or scripting language source loaded into the browser. These checks will be performed when a user clicks a submit button on the page, or when focus moves from an input field. The other form of validation check is a server side check. This type of check is performed within the receiving application on the server. These checks are typically performed after the page is submitted to the server, but before the input choices are submitted to lower level processes. The major difference between these 2 methods is that a malicious user can alter client side checks but would have no control over server side checks.

4.4.1 Issue 2: DOB validation on patient search can be bypassed

Risk:	**HIGH** Successful attack could result in ePHI disclosure
Complexity:	**LOW** This attack only requires the ability to intercept POST parameters being sent to a server. This attack is trivial to carry out.

Summary:

The application provides a Patient Search feature that allows users to retrieve patient records. The search tool requires a user to provide an 'EMR Identifier' along with 'Facility Name' OR the 'Last Name' and 'Date of Birth' for the patient. Attempting to enter only 1 input produces a client side generated error informing the user of the need for both inputs.

The Patient Search application contains a vulnerability that allows the user to bypass the DOB requirement and search for patient records based on just the last name.

We can demonstrate this by performing the following steps:

1. Setup an intercept proxy[1] to alter information submitted from the client to the server
2. Navigate to the Patient Search page
3. Enter a Last Name and any Date of Birth
4. Submit the form, but intercept it with the proxy
5. Remove the submitted date from the DOB field
6. Allow the altered HTTP POST to continue to the server as normal

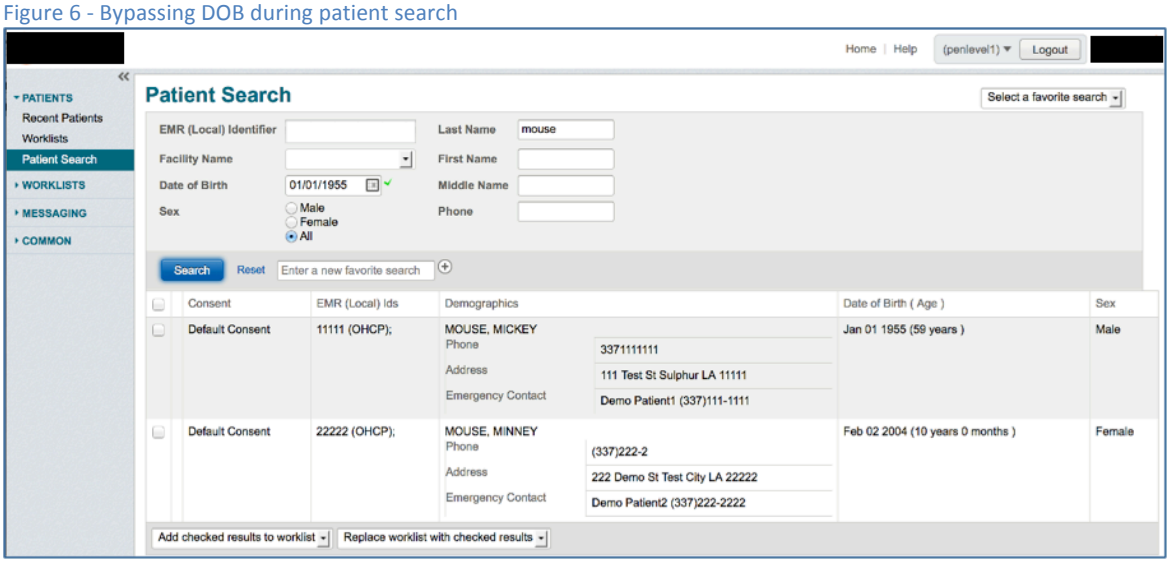
The screen shot below shows an example of the HTTP POST parameters and the DOB field is circled. What we did was to remove the data from this field before sending it to the server.

Figure 5 - Intercepting a POST request to SQLResults.htm

[1] An intercept proxy (sometimes called a client side proxy or attack proxy) is a common pen-testing tool for performing man-in-the-middle type attacks against web applications.

The results of removing the DOB field from the POST request was that we were returned a list of all patients with the last name 'Mouse'.

Figure 6 - Bypassing DOB during patient search



Details:

When the application sends the POST request to the SQLResults.htm page, client side code is doing a good job to make certain the user provides the information that is required for the authorized search to be conducted. The problem lies in that this same check is not being performed on the server side.

We attempted other variations of this attack, such as providing just the DOB and no name, but these attacks all resulted in no records being returned. The issue seems to be isolated to the DOB field.

To exploit this vulnerability, the user needs only to intercept the POST as it is being submitted to the server, such as we did in our example above, or intercept and alter the search form before it is rendered in the browser to remove the script code performing the validation check. In our case we used a tool called Burp Proxy, but this is only one of many available proxy tools. There are even Firefox and Chrome extensions that can accomplish what we did here.

Recommended Resolution:

TBG Security recommends that applicable validation checks be performed on the server side of the search operation to assure the client is providing the expected input before using the provided inputs in SQL queries, stored procedures or other lower level functions.

We also must point out that we had no visibility into how data is handled on the server side, and we cant be certain we exhausted all potential attack vectors when testing the SQLResults.htm form. Therefore we strongly recommend a detailed code review be performed to search for other potential abuses.

4.4.2 Issue 3: 30,000 character message limit can be bypassed	
Risk:	**LOW** There is a potential resource consumption or denial of service risk
Complexity:	**LOW** This attack only requires the ability to intercept POST parameters being sent to a server. This attack is trivial to carry out.

Summary:
The messaging mechanism restricts sent message sizes to 30,000 characters, however the validation of message size is checked in the browser via Javascript and not validated server side. It is possible for users to send messages of arbitrary sizes and have them stored by the application. While this is a low severity issue, it could present availability concerns if a malicious user crafts a number of large messages, consuming large amounts of hard drive space.

Details:
The steps to reproduce this issue are the same is explained in 4.4.1 above. The attack vector is exactly the same, as is the root cause. The application is not validating the user supplied content prior to processing it on the server side. Defeating the client side control is trivial.

Recommended Resolution:
TBG Security recommends that applicable validation checks be performed on the server side as well as on the client side. This practice should be applied as a standard throughout the application code base.

TBG Security strongly recommends reviewing the server side code for other examples of this flaw.

Conclusion:
TBG Security identified 2 specific areas where the application was failing to validate client supplied inputs on the server before those inputs were processed. This may be evidence that a more systemic failure exists, but it is not possible for us to determine this at this time. We do believe that a more holistic review of how client and server side validation is conducted across the entire application is certainly warranted.

4.5 Hidden field identification and tampering

Developer's sometimes hide information in hidden fields within the HTML of the web page. These hidden fields are not meant to be used as security control, and doing so constitutes attempting to provide "security through obscurity". Just because the field doesn't show up on the rendered page in the browser, doesn't mean it can't be seen by viewing the page source or even intercepted and tampered with.

Conclusion:
In no way was TBG able to leverage hidden fields to aid in a successful attack.

4.6 Cookie Abuse

Cookies are used to store static information on a per user/browser basis or on a per session basis. It is session cookies that control much of the security of modern web applications. Modification of cookies, often called poisoning, has been a common attack vector since their inception into web based applications. Using cookie poisoning attacks, an attacker could gain unauthorized information about another user or steal a user's identity.

several cookies are generated for the HIE Portal application which handle session state or host state. These include:

- JSESSIONID
- PlatformNodewec-stagefrontend
- PlatformNode-[############]
- com.orchestral.core.session.api_6_2.Session%2Fclinicalportal[############]
- PlatformNode[############]
- PlatformNodewec-stagebackend
- PlatformNode############.com.orchestral.core.session.api_6_2.Session%2Fhie[############]

The JSESSIONID cookie is reset upon a proper logout. We also observed that the following cookies are set with each logon session:

Figure 7 - Cookies being set after successful logon

```
Set-Cookie: JSESSIONID=ACF465DD53461A7172E574F11A2794C4; Path=/concerto; Secure; HttpOnly
Set-Cookie: persistentAuthenticationMethod=""; Expires=Thu, 01-Jan-1970 00:00:10 GMT; Path=/concerto
Set-Cookie: persistentUserId=""; Expires=Thu, 01-Jan-1970 00:00:10 GMT; Path=/concerto
Set-Cookie: persistentPassword=""; Expires=Thu, 01-Jan-1970 00:00:10 GMT; Path=/concerto
Set-Cookie: ConcertoLocationLocale=en; Expires=Mon, 16-Feb-2015 03:00:05 GMT; Path=/concerto
Set-Cookie: ConcertoLocale=en; Path=/
Set-Cookie: com.orchestral.core.session.api_6_2.Session%2Fclinicalportal308897665=QjpSoP54rE95pxpZQyabkg; Path=/;
```

Note, that the 2nd, 3rd, and 4th cookies set in the example above are set to NULL, and these cookies do not appear to be utilized by the client.

We attempted various types of cookie abuses such as:

- removing cookies post-authentication,
- swapping out cookies with those of other known and active sessions, and
- forcing out of sequence authentication submittals (bypassing logout)

Tampering with the cookies would either cause a Session Expired message or some other error.

Conclusion:
Cookies that control session state and or authorization appear to be properly protected from tampering attacks. We were unsuccessful in finding any issues in the way that cookies were being implemented within the HIE Portal application.

4.7 Session Hijacking

Session hijacking is the act of taking control of a user session after successfully obtaining or generating an authentication session ID. Session hijacking involves an attacker using captured, brute forced or reverse-engineered session IDs to seize control of a legitimate user's Web application session while that session is still in progress.

We performed an Entropy test, to test the randomness of a couple of cookies that are generated which each successful login attempt. The results were rated as "Excellent".

Figure 8 - Entropy test for JSESSIONID cookie

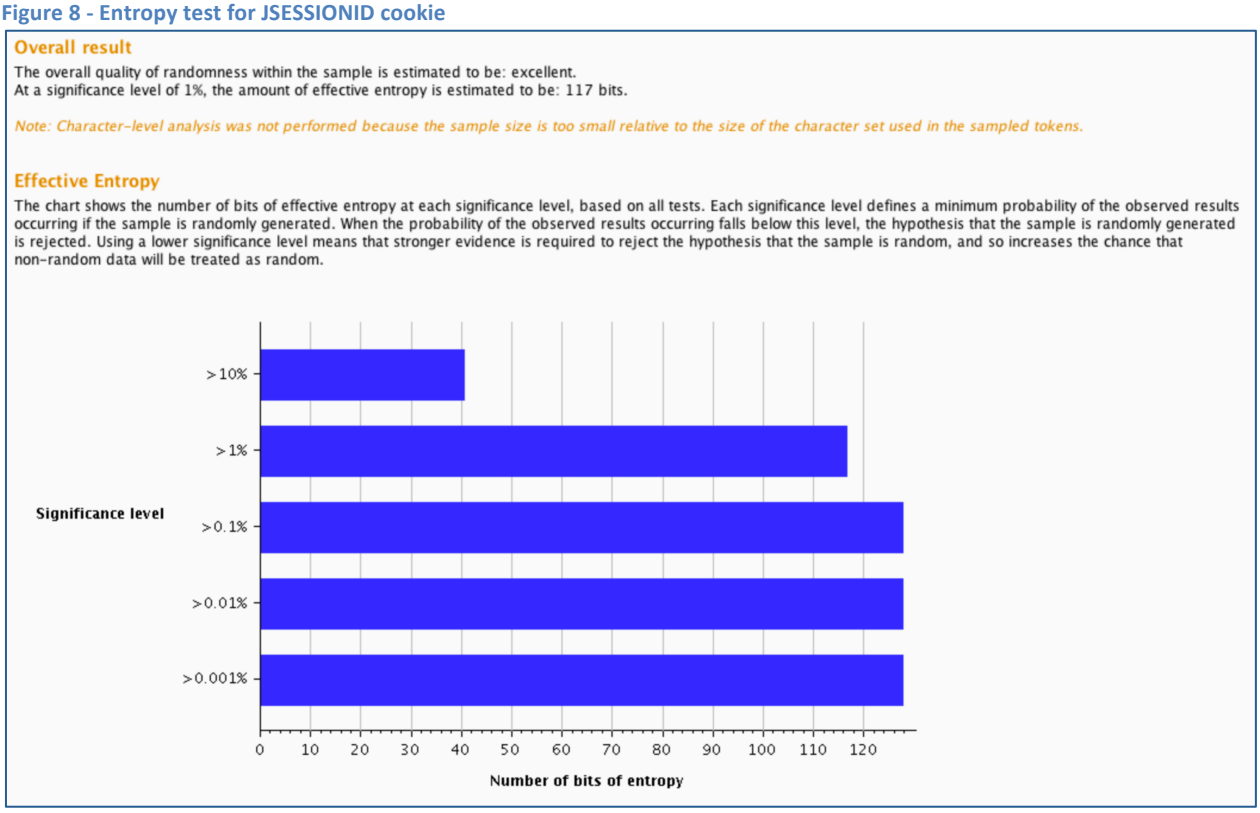
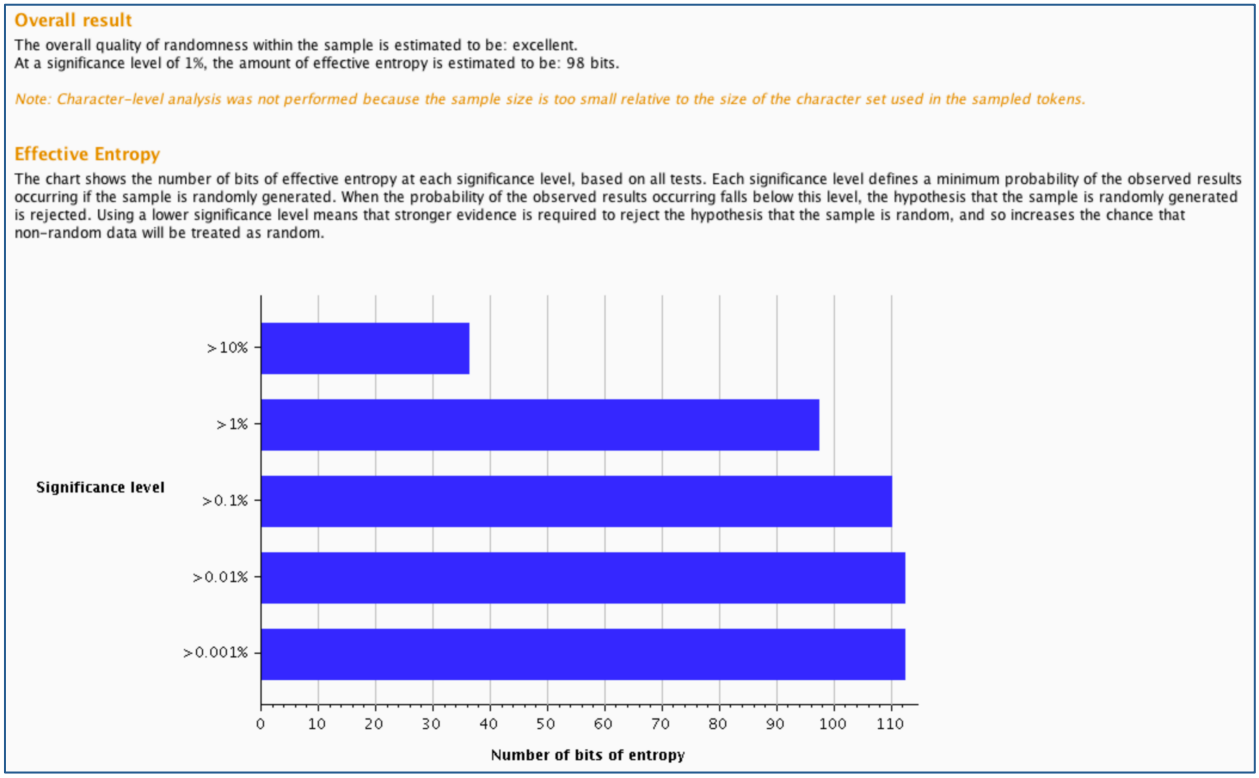


Figure 9 - Entropy test for com.orchestral.core.session.api_6_2.Session-hie########## Cookie



Conclusion:
The session cookies used for these applications are quite strong and therefore resistant to tampering.

4.8 URL Jumping

URL jumping is the practice of avoiding authentication and authorization simply by pointing directly to a known link. For instance a user might be logged into a site and enter /admin at the end of the URL bar and be directed to the administrative page, even though the user is not an administrator. This sort of attack will work when security controls, such as session management, are not in play, and security is provided through obfuscation only.

Conclusion:
We did not uncover any URL jumping issues within the HIE Portal.

4.9 Cross Site Scripting

Cross site scripting is an attack vector that takes advantage of dynamically generated Web pages. In an XSS attack, a Web application is sent with a script that activates when it is read by an unsuspecting user's browser or by an application that has not protected itself against cross-site scripting. Because dynamic Web sites rely on user input, a malicious user can input malicious script into the page by hiding it within legitimate requests. Common exploitations include search engine boxes, online forums and public-accessed blogs. Once XSS has been launched, the attacker can change user settings, hijack

accounts, poison cookies with malicious code, expose SSL connections, access restricted sites and even launch false advertisements.

Conclusion:
We ran automated tests in search of cross site scripting issues in the HIE Portal web site, but found no evidence of this vulnerability.

4.10 Directory browsing

Directory browsing is an information gathering attack which leverages an administrative misconfiguration in a web server which allows listing of directory contents. This is a very bad practice as it provides a would-be attack far too much information. Most web servers are configured out-of-the-box with directory browsing turned on. As a result, this vulnerability is still often found in the wild.

Conclusion:
We identified no instances where directory browsing was allowed on the HIE Portal web site.

4.11 SQL Injection

SQL Injection is an attack method which allows an attacker to inject SQL code through some method of client supplied input. This injected code will be concatenated with valid code on the server side to change the SQL query to allow some form of unauthorized access, data mining, or code execution. SQL injection flaws can often be found on authentication pages and allow unauthorized access to a vulnerable site.

Conclusion:
We performed both automated and manual attacks, but did not find any SQL Injection flaws on the HIE Portal web application.

4.12 Logical Design Issues

Logical design issues are programmatic flaws within the application, which cause the application to operate in some way other than how the programmer originally planned, and that could pose a threat to the security of the site. This category of testing is somewhat of a catchall where any issues that don't fall neatly into one of the earlier categories can be presented.

4.12.1 Issue 4: Password not required when setting email

Risk:	**Medium** Successful attack could result in unauthorized access to a legitimate user's account
Complexity:	**HIGH** This attack requires the attack to highjack an active user session or gain access to a user's workstation while they are logged into the

	application.

Summary:
The 'My Details' page allows users to update preferences, including their email address. The application does not require the user to enter their password prior to updating the email address field in 'My Details', as is commonly accepted security practice. Should a user leave their session logged in or a malicious user gain momentary access to a browser session that is logged into the application, it is possible for the malicious user to update the user's email address and gain persistent access to their account.

Recommended Resolution:
TBG Security recommends that the application validate the user's password prior to changing the email address preference.

Conclusion:
Overall the application seems very soundly built. We did not uncover many logical flaws. We did point out the lack of a password when updating a user's email. Please note that this check is required when changing the security question, so it only makes sense that the same control be applied when changing the email.

4.13 System and software vulnerabilities

Web server software and host operating systems where web applications exist are sometimes misconfigured in ways that impose risk on the web application. We use various vulnerability scanning methods to test for issues within the application server.

4.13.1 Issue 5: The version of Yahoo! YUI is out of date and "end of life"

Risk:	**Medium** Potential impact could vary
Complexity:	**HIGH** An attack would likely need to conduct a successful XSS attack to leverage the vulnerability

Summary:
The version of Yahoo! YUI is out of date and contains vulnerabilities in several SWF files shipped with the library. While the test environment restricted access to the SWF files, there are known issues in the uploader.swf file (**http://www.securityfocus.com/bid/63660**).

Furthermore, Yahoo notes in their security bulletin (**http://yuilibrary.com/support/20131111-vulnerability/**) that this product is "end of life".

Recommended Resolution:
TBG Security recommends that any swf note explicitly necessary be removed if not needed. Further more, if YUI 2.x is still the standard implementation, TBG recommends this be discontinued, and YUI 3 be introduced in its place, as per Yahoo's recommendation from the security bulletin.

4.13.2 Issue 6: 'secure' attribute not set on some cookies

Risk:	**Medium** Impact could include disclosure of session cookies; however, the dynamically generated, per-session cookies were found to have the Secure Flag set appropriately
Complexity:	**HIGH** An attack would likely need to conduct a successful XSS attacker to leverage the vulnerability

Summary:
The application sets some cookies without the 'secure' attribute during an encrypted session. Since the cookie does not contain the 'secure' attribute, it might also be sent to the site during an unencrypted session. Any information such as cookies, session tokens or user credentials that are sent to the server as clear text may be stolen and used later for user impersonation.

Details
If the secure flag is set on a cookie, then browsers will not submit the cookie in any requests that use an unencrypted HTTP connection, thereby preventing the cookie from being trivially intercepted by an attacker. If the secure flag is not set, then the cookie will be transmitted in clear-text if the user visits any HTTP URLs within the cookie's scope. An attacker may be able to induce this event by feeding a user suitable links, either directly or via another web site. Even if the domain which issued the cookie does not host any content that is accessed over HTTP, an attacker may be able to use links of the form http://example.com:443/ to perform the same attack.

The following cookies should have the 'secure' attribute set:

- persistentUserId
- persistentPassword
- persistentAuthenticationMethod
- PlatformNodewec-stagefrontend
- PlatformNode-*.

Recommended Resolution:
TBG Security recommends that secure attribute be set on all application cookies, when feasible.

Conclusion:
The staging system we tested during this application penetration test was found to have only a few issues. These issues should be addressed as soon as possible.

5 Conclusion

Overall, we found the application under review to be well designed and to be utilizing many solid security practices. During testing we noted the obvious implementation of a solid security framework. The use of encryption for the delivery of client delivered parameters, and the general temporal nature of tokenized parameters make the application mostly unsusceptible to tampering style attacks, although we do note 1 example of an exception to this otherwise very good practice. It should be noted that we did not have access to the source code, as a source code review was out of scope for this project, but it was obvious that a good solid framework of security was being employed and we were unsuccessful in leveraging the most common attack methods plaguing Internet application today such as Cross Site Scripting and SQL Injection.

The following list is a summary of items requiring remediation:

	Description	Risk Rating	Page Ref
Issue 1	Message disclosure vulnerability	High	8
Issue 2	DOB validation bypass vulnerability	High	10
Issue 3	Message character limit can be bypassed	Low	12
Issue 4	Password validation not required when resetting user email	Medium	16
Issue 5	Yahoo! YUI is out of date	Medium	17
Issue 6	'secure' attribute not set on some cookies	Medium	18