Deepscan

SECURITY ASSESSMENT REPORT

November 10, 2019

http://83.212.174.87

EXECUTIVE SUMMARY

OVERVIEW OF EFFORT

Deepscan performed a Security Assessment of **http://83.212.174.87** ("the application") using a combination of manual review analysis as well as semi-automated penetration testing. Testing activities took place November 09 – 10, 2019. The purpose of this report is to present a summary of the findings and their impact. Several of the tests performed resulted in the discovery of a security finding. A major focus of testing was SQL Injection of the Login Form.

In all, 3142 attack vectors were examined finding 5 types of vulnerabilities related to the application: 1 High risk, 1 Medium risk, and 3 Low risk.

The application was found to be relatively secure from most external attacks. With exception to a few Cross-Site Scripting and ClickJacking vulnerabilities, the application responded appropriately to malicious traffic. It is recommended that all High & Medium risk items are addressed with priority to prevent potentially serious security breaches. A re-test is recommended once the vulnerabilities have been addressed to ensure no new risks have been introduced.

HOW TO READ OUR FINDINGS

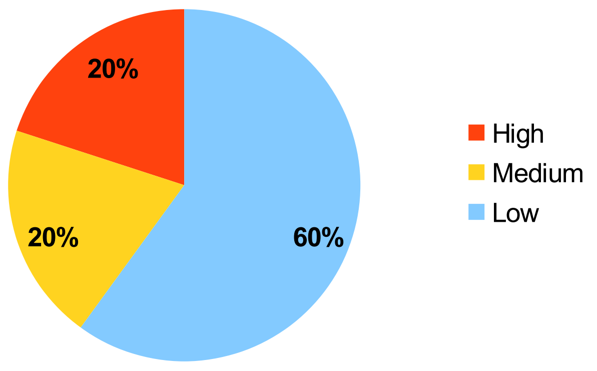
For each security critical area, a series of *Findings*, or security relevant discoveries is included. Each finding is detailed in a similar and easy-to-digest pattern. For each finding, the following is detailed:

- The location of the security-relevant finding, whether it be a URL or a source code file
- A description of the risk – to include the likelihood and impact
- A recommended solution for the particular finding
- Any notable references

Each finding has an associated severity calculation. The severity calculation will aid the software development team in prioritizing what order the issues should be addressed.

SUMMARY OF FINDINGS

Risk Level	Number of Findings
High	**1**
Medium	**1**
Low	**3**



A summary of the findings for the application Security Assessment

FINDING OVERVIEW

An overview of every risk type identified by the security assessment is described along with its severity and impact to the system or business.

Risk Type	Severity	Impact
Cross Site Scripting (Reflected)	High	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site.
ClickJacking	Medium	Potential to trick a user into clicking on a button or link on another page when they were intending to click on the the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both. May be used to steal personal information, execute unintended actions, etc.
Absence of Anti-CSRF Tokens	Low	Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.
XSS Protection Not Enabled	Low	Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other

		sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.
X-Content-Type-Options Header Missing	Low	MIME type sniffing is a standard functionality in browsers to find an appropriate way to render data where the HTTP headers sent by the server are either inconclusive or missing. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the intended content type. The problem arises once a website allows users to upload content which is then published on the web server. If an attacker can carry out XSS (Cross-site Scripting) attack by manipulating the content in a way to be accepted by the web application and rendered as HTML by the browser, it is possible to inject code in e.g. an image file and make the victim execute it by viewing the image.

RISK DETERMINATION

A standard approach set forth by OWASP (Open Web Application Security Project) is used in determining the risk associated with the findings. The overall severity of each finding is an evaluation of likelihood and system/business impact, which is then used to calculate the severity level as follows:

SEVERITY = LIKELIHOOD * IMPACT

LIKELIHOOD

The likelihood of a finding is the probability that the finding will be discovered and exploited by a threat agent.

THREAT AGENT FACTORS

- **SKILL LEVEL** – What is the technical level required exploiting the vulnerability?
- **MOTIVE** - How motivated is this group of attackers to find and exploit this vulnerability?
- **OPPORTUNITY** - What resources and opportunities are required for this group of attackers to find and exploit this vulnerability?
- **SIZE** - How large is this group of attackers?

VULNERABILITY FACTORS

- **EASE OF DISCOVERABILITY** - How easy is it for this group of attackers to discover this vulnerability?
- **EASE OF EXPLOIT** - How easy is it for this group of attackers to actually exploit this vulnerability?
- **AWARENESS** - How well known is this vulnerability to this group of attackers?
- **INTRUSION DETECTION** - How likely is an exploit to be detected?

IMPACT

The impact of a finding is the assessment of negative consequence on the system and the business.

SYSTEM IMPACT FACTORS

- **LOSS OF CONFIDENTIALITY** - How much data could be disclosed and how sensitive is it?
- **LOSS OF INTEGRITY** - How much data could be corrupted and how damaged is it?
- **LOSS OF AVAILABILITY** - How much service could be lost and how vital is it?
- **LOSS OF ACCOUNTABILITY** - Are the attackers' actions traceable to an individual?

BUSINESS IMPACT FACTORS

- **FINANCIAL DAMAGE** - How much financial damage will result from an exploit?
- **REPUTATION DAMAGE** - Would an exploit result in reputation damage that would harm the business?
- **NON-COMPLIANCE** - How much exposure does non-compliance introduce?
- **PRIVACY VIOLATION** - How much personally identifiable information could be disclosed?

SECURITY CRITICAL AREAS

For each security critical area, you will find a brief description of the area as well as the impact of findings under the area. The focus of the security assessment performed was on the following security critical areas as noted in the OWASP Top Ten 2017:

A1:2017-Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

A2:2017-Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

A3:2017-Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

A4:2017-XML External Entities (XXE)

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

A5:2017-Broken Access Control

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

A6:2017-Security Misconfiguration

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

A7:2017-Cross-Site Scripting (XSS)

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

A8:2017-Insecure Deserialization

Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

A9:2017-Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

A10:2017-Insufficient Logging & Monitoring

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

FINDINGS DETAIL

Each of the specific security findings found during the security assessment are detailed below. For each finding, we describe the risk severity, location of the vulnerability, recommended solution, and provide any notable references.

High (Medium)	**Cross Site Scripting (Reflected)**
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://83.212.174.87/login.php
Method	POST
Parameter	password
Attack	`<script>alert(1);</script>`
Evidence	`<script>alert(1);</script>`

Instances	1
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to

	be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Source ID	1

Medium (Medium)	**X-Frame-Options Header Not Set**
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://83.212.174.87/login.php
Method	GET
Parameter	X-Frame-Options
URL	http://83.212.174.87/login.php
Method	POST
Parameter	X-Frame-Options
URL	http://83.212.174.87/
Method	GET

Parameter	X-Frame-Options
Instances	3
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	**Web Browser XSS Protection Not Enabled**
Description	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
URL	http://83.212.174.87/robots.txt
Method	GET
Parameter	X-XSS-Protection
URL	http://83.212.174.87/login.php
Method	GET
Parameter	X-XSS-Protection
URL	http://83.212.174.87/
Method	GET
Parameter	X-XSS-Protection
URL	http://83.212.174.87/login.php
Method	POST
Parameter	X-XSS-Protection
URL	http://83.212.174.87/sitemap.xml
Method	GET
Parameter	X-XSS-Protection
URL	http://83.212.174.87/mal.sh;%60
Method	GET
Parameter	X-XSS-Protection
Instances	6
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Other information	The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it:

X-XSS-Protection: 1; mode=block

X-XSS-Protection: 1; report=http://www.example.com/xss

The following values would disable it:

X-XSS-Protection: 0

The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit).

Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length).

Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://www.veracode.com/blog/2014/03/guidelines-for-setting-security-headers/
CWE Id	933
WASC Id	14
Source ID	3

Low (Medium)	**X-Content-Type-Options Header Missing**
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://83.212.174.87/
Method	GET
Parameter	X-Content-Type-Options
URL	http://83.212.174.87/util.sh
Method	GET
Parameter	X-Content-Type-Options
URL	http://83.212.174.87/login.php
Method	POST
Parameter	X-Content-Type-Options
URL	http://83.212.174.87/login.php
Method	GET
Parameter	X-Content-Type-Options
Instances	4
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and

that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.

If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

Other information

This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.

At "High" threshold this scanner will not alert on client or server error responses.

Reference

http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx

https://www.owasp.org/index.php/List_of_useful_HTTP_headers

CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	**Absence of Anti-CSRF Tokens**

Description

No Anti-CSRF tokens were found in a HTML submission form.

A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.

CSRF attacks are effective in a number of situations, including:

* The victim has an active session on the target site.

* The victim is authenticated via HTTP auth on the target site.

* The victim is on the same local network as the target site.

CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.

URL	http://83.212.174.87/
Method	GET
Evidence	<form action="/login.php" method="POST">
Instances	1
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script. Phase: Architecture and Design Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change. Phase: Implementation Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.
Other information	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret] was found in the following HTML form: [Form 1: "password"].
Reference	http://projects.webappsec.org/Cross-Site-Request-Forgery

	http://cwe.mitre.org/data/definitions/352.html
CWE Id	352
WASC Id	9
Source ID	3

ABOUT US

Deepscan is a Senior Quality Professional with over 20 years experience in the fields of Quality Assurance, Ethical Hacking, and Web Programming. We provide customers with the information needed to be secure through Award Winning cutting-edge technology and industry expertise.

Deepscan is based in Canada and proudly serves customers around the world!

www.DeepscanSecure.com Info@DeepscanSecure.com

DEEPSCAN ADVANTAGE

Deepscan services are performed with customer success and satisfaction in mind. Every effort is made to ensure customers receive the most accurate and highest quality information.

We work closely with each customer from project inception through delivery; identifying project requirements and ensuring deliverables meet or *exceed* expectations.
The **Deepscan Advantage** includes:

- Over 3000 attack vectors including SQL Injection, Cross-Site Scripting (XSS), Remote Command Execution, Parameter Tampering and more!
- Deep Crawl & Analysis
- Information Gathering & Reconstruction
- High Detection Rate, Low False Positives
- Vulnerability Scanning & Management to prioritize security threats
- 5-star Customer Service

QUALIFICATIONS

NAIT (Northern Alberta Institute of Technology)
B.Sc. - Bachelor of Applied Science - Application Development
2001

American Society for Quality (ASQ.org)
Professional Member
2005 - current

Six Sigma Practitioner
Design for Six Sigma
2010

Certified Ethical Hacker (CEH)
EC-Council
2012