VERACODE

Veracode Detailed Report

Application Security Report
As of 6 Feb 2014

Prepared for:	Radio Free Asia
Prepared on:	March 14, 2014
Application:	GlobaLeaks
Industry:	Not Specified
Business Criticality:	BC4 (High)
Required Analysis:	Static
Type(s) of Analysis Conducted:	Manual

Inside This Report

Executive Summary	1
Summary of Flaws by Severity	1
Action Items	1
Flaw Types by Category	3
Policy Summary	5
Findings & Recommendations	6
Manual Flaw Details	
Methodology	

65 Network Drive, Burlington, MA 01803

Tel.+1.339.674.2500 **Fax.**+1.339.674.2502 **URL:**http://www.veracode.com

Veracode Detailed Report

Application Security Report As of 6 Feb 2014

Veracode Level: VL2

Rated: Feb 6, 2014

Application:	GlobaLeaks	Business Criticality:	High
Target Level:	VL4	Published Rating:	A

Scans Included in Report

Static Scan	Dynamic Scan	Manual Scan
Not Included in Report	Not Included in Report	Jan 2014 Manual Results Score: 92 Completed: 2/6/14

Executive Summary

This report contains a summary of the security flaws identified in the application using automated static, automated dynamic and/or manual security analysis techniques. This is useful for understanding the overall security quality of an individual application or for comparisons between applications.

Application Business Criticality: BC4 (High)

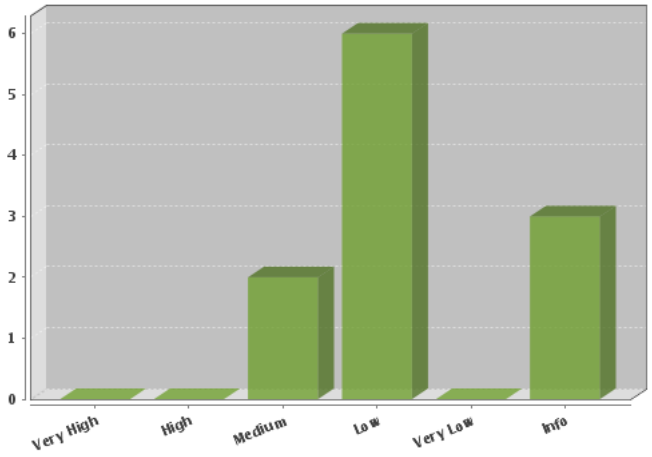
Impacts:Operational Risk (Medium), Financial Loss (Medium)

An application's business criticality is determined by business risk factors such as: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations. The Veracode Level and required assessment techniques are selected based on the policy assigned to the application.

Analyses Performed vs. Required

	Any	Static	Dynamic	Manual
Performed:		○	○	●
Required:	○	●	○	○

Summary of Flaws Found by Severity



Action Items:

Veracode recommends the following approaches ranging from the most basic to the strong security measures that a vendor can undertake to increase the overall security level of the application.

Required Analysis

→ Your policy requires Static Scan but it has not been performed. Please submit your application for Static Scan and remediate the required detected flaws to conform to your assigned policy.

Flaws To Fix By Expires Date

A grace period is specified for any flaw that violates the rules contained in your policy. These include CWE, Rollup Category, Issue Severity, Industry Standards as well as any flaws the prevent an application from achieving a minimum Veracode Level and/or score. To maintain policy compliance you must fix these flaws and resubmit your application for scanning before the grace period expires. The detailed flaw listing will badge the flaws that must be fixed and show the fix by date as well.

→ The grace period has expired [2/6/14] for 2 flaws that were found in your Manual Scan.

Longer Timeframe (6 - 12 months)

- Certify that software engineers have been trained on application security principles and practices.

65 Network Drive, Burlington, MA 01803

Tel.+1.339.674.2500 **Fax.**+1.339.674.2502 **URL:**http://www.veracode.com

Flaw Types by Severity and Category

	Manual Scan Security Quality Score = 92		
Very High	**0**		
High	**0**		
Medium	**2**		
Other	2		
Low	**6**		
Cryptographic Issues	5		
Information Leakage	1		
Very Low	**0**		
Informational	**3**		
Insecure Dependencies	3		
Total	**11**		

65 Network Drive, Burlington, MA 01803

Tel.+1.339.674.2500 **Fax.**+1.339.674.2502 **URL:**http://www.veracode.com

Manual Penetration Test Summary Findings

Below is a summary of vulnerabilities found as categorized by CAPEC. A score of 100 indicates that no vulnerabilities were discovered under that attack category.

Attack Category	Score (out of 100)
Abuse of Functionality Exploitation of business logic errors or misappropriation of programmatic resources. Application functions are developed to specifications with particular intentions, and these types of attacks serve to undermine those intentions.	59
Spoofing Impersonation of entities or trusted resources. A successful attack will present itself to a verifying entity with an acceptable level of authenticity.	100
Probabilistic Techniques Using predictive capabilities or exhaustive search techniques in order to derive or manipulate sensitive information. Attacks capitalize on the availability of computing resources or the lack of entropy within targeted components.	100
Exploitation of Authentication Circumventing authentication requirements to access protected resources. Design or implementation flaws may allow authentication checks to be ignored, delegated, or bypassed.	100
Resource Depletion Affecting the availability of application components or resources through symmetric or asymmetric consumption. Unrestricted access to computationally expensive functions or implementation flaws that affect the stability of the application can be targeted by an attacker in order to cause denial of service conditions.	100
Exploitation of Privilege/Trust Undermining the application's trust model in order to gain access to protected resources or gain additional levels of access as defined by the application. Application that implicitly extend trust to resources or entities outside of their direct control are susceptible to attack.	100
Injection (Injecting Control Plane content through the Data Plane) Inserting unexpected inputs to manipulate control flow or alter normal business processing. Applications must contain sufficient data validation checks in order to sanitize tainted data and prevent malicious, external control over internal processing.	100
Data Structure Attacks Supplying unexpected or excessive data that results in more data being written to a buffer than it is capable of holding. Successful attacks of this class can result in arbitrary command execution or denial of service conditions.	100
Data Leakage Attacks Recovering information exposed by the application that may itself be confidential or may be useful to an attacker in discovering or exploiting other weaknesses. A successful attack may be conducted by passive observation or active interception methods. This attack patter often manifests itself in the form of applications that expose sensitive information within error messages.	86
Resource Manipulation Manipulating application dependencies or accessed resources in order to undermine security controls and gain unauthorized access to protected resources. Applications may use tainted data when constructing paths to local resources or when constructing processing environments.	100
Time and State Attacks Undermining state condition assumptions made by the application or capitalizing on time delays between security checks and performed operations. An application that does not enforce a required processing sequence or does not handle concurrency adequately will be susceptible to these attack patterns.	100

Policy Evaluation

Policy Name: Veracode Recommended High

Revision: 1

Policy Status: Did Not Pass

Description

Veracode provides default policies to make it easier for organizations to begin measuring their applications against policies. Veracode Recommended Policies are available for customers as an option when they are ready to move beyond the initial bar set by the Veracode Transitional Policies. The policies are based on the Veracode Level definitions.

Rules

Rule type	Requirement	Findings	Status
Minimum Veracode Level	VL4	VL2	Did not pass
(VL4) Min Analysis Score	80	92	Passed
(VL4) Max Severity	Medium	Flaws found: 2	Did not pass

Scan Requirements

Scan Type	Frequency	Last performed	Status
Static	Quarterly	Never	Passed (until 4/13/14)

Remediation

Flaw Severity	Grace Period	Flaws Exceeding	Status
Very High	0 days	0	Passed
High	0 days	0	Passed
Medium	0 days	2	Did not pass
Low	0 days	0	Passed
Very Low	0 days	0	Passed
Informational	0 days	0	Passed

Type	Grace Period	Exceeding	Status
Min Analysis Score	0 days	0	Passed

Findings & Recommendations

Detailed Flaws by Severity

Very High (0 flaws)

No flaws of this type were found

High (0 flaws)

No flaws of this type were found

Medium (2 flaws)

Fix Required by Policy

Other(2 flaws)

Description

These flaws do not fit into one of Veracode's existing categories.

Recommendations

Please see individual flaw descriptions for remediation guidance.

Associated Flaws by CWE ID:

Improper Restriction of Excessive Authentication Attempts (CWE ID 307)(1 flaw)

Description

The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Recommendations

Implement a CAPTCHA solution to limit the number of requests to the Abuse and Feedback functionality.

Vulnerabilities found through Manual Penetration Testing

See "Detailed Flaws By Severity: Manual Flaws" for full information regarding these flaws.

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
1	Simple	5	1	Excessive Abuse and Feedback Submissions Allowed	Abuse of Functionality	2/6/14
	↳ *Reviewer Notes:*	CVSS = (AV:N/AC:L/Au:N/C:N/I:N/A:P)				

Protection Mechanism Failure (CWE ID 693)(1 flaw)

Description

The product does not use a protection mechanism that provides sufficient defense against directed attacks against the product.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Recommendations

Using iframes to load the hidden service is believed to be the only solution that will not allow the hidden service to modify the banner.

Vulnerabilities found through Manual Penetration Testing

See "Detailed Flaws By Severity: Manual Flaws" for full information regarding these flaws.

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
2	Average	4.3	1	Warning Banner Escapeable	Abuse of Functionality	2/6/14
	↳ *Reviewer Notes:*	CVSS = (AV:N/AC:M/Au:N/C:N/I:P/A:N)				

Low (6 flaws)

Cryptographic Issues(5 flaws)

Description

Applications commonly use cryptography to implement authentication mechanisms and to ensure the confidentiality and integrity of sensitive data, both in transit and at rest. The proper and accurate implementation of cryptography is extremely critical to its efficacy. Configuration or coding mistakes as well as incorrect assumptions may negate a large degree of the protection it affords, leaving the crypto implementation vulnerable to attack.

Common cryptographic mistakes include, but are not limited to, selecting weak keys or weak cipher modes, unintentionally exposing sensitive cryptographic data, using predictable entropy sources, and mismanaging or hard-coding keys.

Developers often make the dangerous assumption that they can improve security by designing their own cryptographic algorithm; however, one of the basic tenets of cryptography is that any cipher whose effectiveness is reliant on the secrecy of the algorithm is fundamentally flawed.

Recommendations

Select the appropriate type of cryptography for the intended purpose. Avoid proprietary encryption algorithms as they typically rely on "security through obscurity" rather than sound mathematics. Select key sizes appropriate for the data being protected; for high assurance applications, 256-bit symmetric keys and 2048-bit asymmetric keys are sufficient. Follow best practices for key storage, and ensure that plaintext data and key material are not inadvertently exposed.

Associated Flaws by CWE ID:

Inadequate Encryption Strength (CWE ID 326)(5 flaws)

Description

Insufficiently strong encryption schemes may not adequately secure secret data from attackers. This can result from poor cipher selection, insufficient key size, or weak key selection.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Recommendations

Use a cryptographic algorithm that has been subject to public scrutiny. Follow security best practices when selecting key sizes and when generating key material.

Vulnerabilities found through Manual Penetration Testing

See "Detailed Flaws By Severity: Manual Flaws" for full information regarding these flaws.

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
3	Average	3.2	4	HTTP Compression is Enabled	Abuse of Functionality	
	↳ *Reviewer Notes:*	CVSS: (AV:A/AC:H/Au:N/C:P/I:P/A:N) = 3.2				
4	Average	3.2	1	TLS Compression is Enabled	Abuse of Functionality	
	↳ *Reviewer Notes:*	CVSS: (AV:A/AC:H/Au:N/C:P/I:P/A:N) = 3.2				

Information Leakage(1 flaw)

Description

An information leak is the intentional or unintentional disclosure of information that is either regarded as sensitive within the product's own functionality or provides information about the product or its environment that could be useful in an attack. Information leakage issues are commonly overlooked because they cannot be used to directly exploit the application. However, information leaks should be viewed as building blocks that an attacker uses to carry out other, more complicated attacks.

There are many different types of problems that involve information leaks, with severities that can range widely depending on the type of information leaked and the context of the information with respect to the application. Common sources of information leakage include, but are not limited to:

* Source code disclosure
* Browsable directories
* Log files or backup files in web-accessible directories
* Unfiltered backend error messages
* Exception stack traces
* Server version information
* Transmission of uninitialized memory containing sensitive data

Recommendations

Configure applications and servers to return generic error messages and to suppress stack traces from being displayed to end users. Ensure that errors generated by the application do not provide insight into specific backend issues.

Remove all backup files, binary archives, alternate versions of files, and test files from web-accessible directories of production servers. The only files that should be present in the application's web document root are files required by the application. Ensure that deployment procedures include the removal of these file types by an administrator. Keep web and application servers fully patched to minimize exposure to publicly-disclosed information leakage vulnerabilities.

Associated Flaws by CWE ID:

Exposure of System Data to an Unauthorized Control Sphere (CWE ID 497)(1 flaw)

Description

The application reveals system data or other debugging information. While not directly exploitable, information leaks often facilitate other attacks against the application.

Effort to Fix: 1 - Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

Recommendations

Avoid exposing server-side system information such as environment variables to end users. Sanitize all messages to remove any sensitive information that is not absolutely necessary.

Vulnerabilities found through Manual Penetration Testing

See "Detailed Flaws By Severity: Manual Flaws" for full information regarding these flaws.

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
5	Simple	3	1	Version Information Leakage	Data Leakage Attacks	
	↳ *Reviewer Notes:* CVSS = (AV:N/AC:H/Au:N/C:P/I:N/A:N/E:F/RL:O/RC:C)					

Very Low (0 flaws)

No flaws of this type were found

Info (3 flaws)

Insecure Dependencies(3 flaws)

Description

When an application is dependent on vulnerable components such as certain libraries or runtimes being present, it inherits the risk of those components. Relying on vulnerable components can also place end users at risk.

Recommendations

Discontinue the use of known vulnerable dependencies.

Associated Flaws by CWE ID:

Inclusion of Functionality from Untrusted Control Sphere (CWE ID 829)(3 flaws)

Description

The application contains a Java applet. The Java Runtime Environment (JRE) has repeatedly and consistently been a source of high impact, reliably exploitable vulnerabilities, and it will likely continue to be a high-profile target. Using applets presents no risk to the server, but requiring end users to have the JRE installed puts them at undue risk.

Effort to Fix: 3 - Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.

Recommendations

Work toward removing the application's dependence on Java applets.

Vulnerabilities found through Manual Penetration Testing

See "Detailed Flaws By Severity: Manual Flaws" for full information regarding these flaws.

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
6	Average	0	1	Hidden Services can Exploit Cross-Domain Access	Abuse of Functionality	
7	Average	0	1	Hidden Services can Exploit Subdomain Access	Abuse of Functionality	
8	Average	0	1	Hidden Services Can Exploit Same Domain Access (x.tor2web.org)	Abuse of Functionality	

Detailed Flaws By Severity: Manual Flaws

Medium (2 flaws)

Fix Required by Policy

Other(2 flaws)

Associated Flaws by CWE ID:

Improper Restriction of Excessive Authentication Attempts (CWE ID 307)(1 flaw)

Vulnerabilities found through Manual Penetration Testing

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
1	Simple	5	1	Excessive Abuse and Feedback Submissions Allowed	Abuse of Functionality	2/6/14
	Reviewer Notes:	CVSS = (AV:N/AC:L/Au:N/C:N/I:N/A:P)				

Effort to Fix: 3

Description

The application allows for excessive submission of Abuse and Feedback notifications. When submitting an Abuse or Feedback notification, the application sends an email to a mailing list that requires manual review from Tor2Web node maintainers.

Severity Description

By flooding the Tor2Web node maintainer's Abuse and Feedback email queue with excessive requests, an attacker may be able to prevent legitimate abuse requests from being fulfilled.

Exploitability

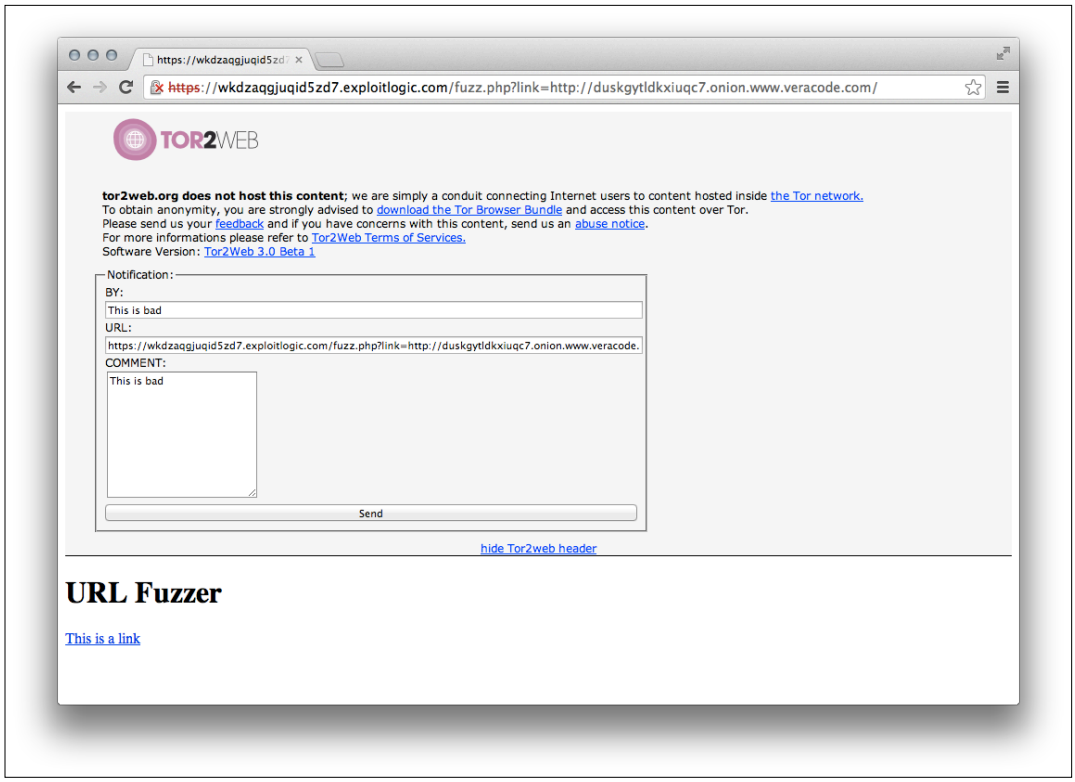
An attacker can automate sending excessive requests to flood the Tor2Web node maintainers Feedback and Abuse queue.

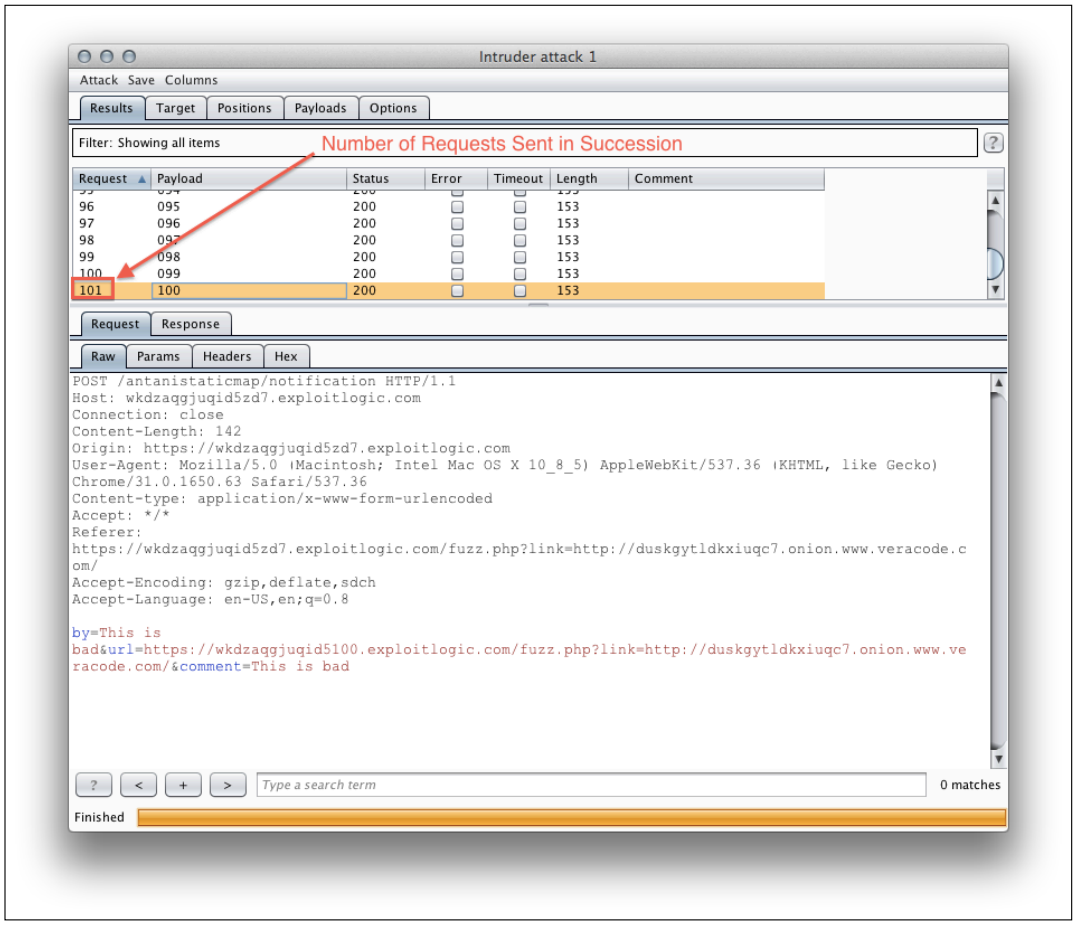
Location

https://hiddenservice.tor2web.org/antanistaticmap/notification

Exhibits

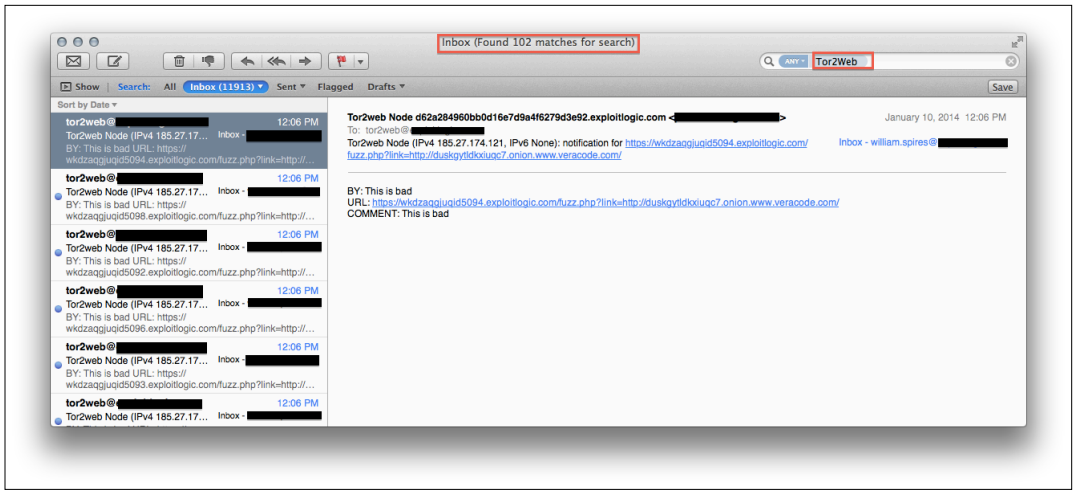
An attacker can automate sending repeated requests to the Abuse and Feedback functionality that results in excessive submissions that Tor2Web node maintainers will need to review.





65 Network Drive, Burlington, MA 01803

Tel.+1.339.674.2500 **Fax.**+1.339.674.2502 **URL:**http://www.veracode.com



Recommendations

Implement a CAPTCHA solution to limit the number of requests to the Abuse and Feedback functionality.

→ Protection Mechanism Failure (CWE ID 693)(1 flaw)

Vulnerabilities found through Manual Penetration Testing

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
2	Average	4.3	1	Warning Banner Escapeable	Abuse of Functionality	2/6/14
	↳ *Reviewer Notes:*	CVSS = (AV:N/AC:M/Au:N/C:N/I:P/A:N)				

Effort to Fix: 3

Description

The tor2web warning banner can be escaped or hidden by the site.

Severity Description

An hidden service attacker can hide the warning banner from the user.

Exploitability

HTML or JavaScript can modify the site in various ways to hide the warning banner.

Location

https://anysite.tor2web.org

Attack Vectors

hidden services

Exhibits

In the example below, an attacker can force the banner to load outside of the viewable screen.


```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Banner Escape Test</title>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link rel="stylesheet" href="my.css">
  </head>
  <body>
    <script type="text/javascript" src="/antanistaticmap/tor2web.js"></script>
    <style type="text/css">
      @import url(/antanistaticmap/tor2web.css);
    </style>
    <div id="tor2web-header">
      <div id="tor2web-visible">
        <div id="tor2web_logo">
          <a href="https://www.tor2web.org"><img src="/antanistaticmap/tor2web-small.png" alt="tor2web logo" /></a>
        </div>
        <div id="tor2web_disclaimer">
          <div><b>tor2web.org does not host this content</b>; we are simply a conduit connecting Internet users to
content hosted inside <a href="https://www.torproject.org/docs/hidden-services.html.en">the Tor network.</a></div>
          <div>To obtain anonymity, you are strongly advised to <a
href="https://www.torproject.org/download/">download the Tor Browser Bundle</a> and access this content over
Tor.</div>
          <div>Please send us your <a href="javascript:show_hide_notification_form()">feedback</a> and if you have
concerns with this content, send us an <a href="javascript:show_hide_notification_form()">abuse notice</a>.</div>
```

```
            <div>For more informations please refer to <a href="/antanistaticmap/tos.html">Tor2Web Terms of
Services.</a></div>
            <div>
              <t:transparent t:render="mirror" />
            </div>
            <div>
              Software Version:
              <a href="https://github.com/globaleaks/Tor2web-3.0">
                <t:transparent t:render="t2wvar-version" />
              </a>
            </div>
          </div>
          <div id="tor2web_notification_form">
            <fieldset>
              <legend>Notification:</legend>
              BY:
              <div><input type="text" id="by" name="by" /></div>
              URL:
              <div><input type="text" id="url" name="url" /></div>
              COMMENT:
              <div><textarea type="text" id="comment" name="comment" rows="10" cols="20"></textarea></div>
              <div><input type="button" value="Send" onclick="sendNotification()" /></div>
            </fieldset>
          </div>
          <div style="clear:both"></div>
          <div class="tor2web_showhide">
            <a href="javascript:show_hide_tor2web_header(true)">hide Tor2web header</a>
          </div>
        </div>
        <div id="tor2web-hidden">
          <div class="tor2web_showhide">
            <a href="javascript:show_hide_tor2web_header(false)">show Tor2web header</a>
          </div>
        </div>
      </div>

      <div class="navbar-fixed-top" >this is now on top<br>this is now on top<br>this is now on top<br>this is now on
top<br>this is now on top<br>this is now on top<br>this is now on top<br>this is now on top</div>


    </body>
```

Recommendations

Using iframes to load the hidden service is believed to be the only solution that will not allow the hidden service to modify the banner.

Low (6 flaws)

Cryptographic Issues(5 flaws)

Associated Flaws by CWE ID:

Inadequate Encryption Strength (CWE ID 326)(5 flaws)

Vulnerabilities found through Manual Penetration Testing

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
3	Average	3.2	4	HTTP Compression is Enabled	Abuse of Functionality	
	↳ *Reviewer Notes:*	CVSS: (AV:A/AC:H/Au:N/C:P/I:P/A:N) = 3.2				

Effort to Fix: 3

Description
The server responds with HTTP compressed responses, allowing HTTP compression attacks compression attacks (BREACH/TIME) which bypass the protections SSL provides to HTTP responses.

Severity Description
This attack would only effect anonymity and sensitive user data (which is not the goal of tor2web), so it is not as important for tor2web sites as it would be for more sensitive websites.

An attacker would be able to recover sensitive information in responses. Examples of this would be account numbers, account names, CSRF tokens, and any other values that the attacker may want to target in the response.

Exploitability
An attacker would need to man-in-the-middle traffic to exploit BREACH (which is more reliable than TIME). The attacker would force the user to make authenticated requests to sensitive content which contains the attacker's input in the response. The attacker would observe the response size of those responses to determine if their controlled input was repeated elsewhere in the response. This technique would be executed one character at at time until the entire secret value is disclosed.

Location
All servers

http://tor2web.org & https://hiddenservice.tor2web.org (194.150.168.70)
http://www.tor2web.org (195.85.254.203)
https://hiddenservice.tor2web.org (65.112.221.20)
https://hiddenservice.tor2web.org (38.229.70.4)

Attack Vectors
HTTP response sizes

Recommendations
Disable HTTP compression on all responses that contain sensitive content, which includes authentication and anti-CSRF tokens.

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
4	Average	3.2	1	TLS Compression is Enabled	Abuse of Functionality	
	↳ *Reviewer Notes:*	CVSS: (AV:A/AC:H/Au:N/C:P/I:P/A:N) = 3.2				

Effort to Fix: 3

Description
The server allows TLS compressed connections, allowing TLS compression attacks (CRIME) which bypasses some of the protections SSL provides to communications.

Severity Description
This attack would only effect anonymity and sensitive user data (which is not the goal of tor2web), so it is not as important for tor2web sites as it would be for more sensitive websites.

An attacker would be able to recover sensitive information in responses. Examples of this would be account numbers, account names, CSRF tokens, and any other values that the attacker may want to target in the response.

Exploitability

An attacker would need to man-in-the-middle traffic to exploit CRIME. The attacker would force the user to make authenticated requests to sensitive content which contains the attacker's input in the response. The attacker would observe the response size of those responses to determine if their controlled input was repeated elsewhere in the response. This technique would be executed one character at at time until the entire secret value is disclosed.

Location
https://hiddenservice.tor2web.org (38.229.70.4)

Attack Vectors
HTTP response sizes

Recommendations
Disable TLS compression on all communications that may contain sensitive content, such as session cookies, that would be of value to an attacker.

→ Information Leakage(1 flaw)

Associated Flaws by CWE ID:

→ Exposure of System Data to an Unauthorized Control Sphere (CWE ID 497)(1 flaw)

Vulnerabilities found through Manual Penetration Testing

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
5	Simple	3	1	Version Information Leakage	Data Leakage Attacks	
	↳ *Reviewer Notes:*	CVSS = (AV:N/AC:H/Au:N/C:P/I:N/A:N/E:F/RL:O/RC:C)				

Effort to Fix: 1

Description
The tor2web banner and the HTTP headers contains versioning information of the environment.

Severity Description
In the future if an exploit is identified, services that are unpatched could be easily identified by attackers.

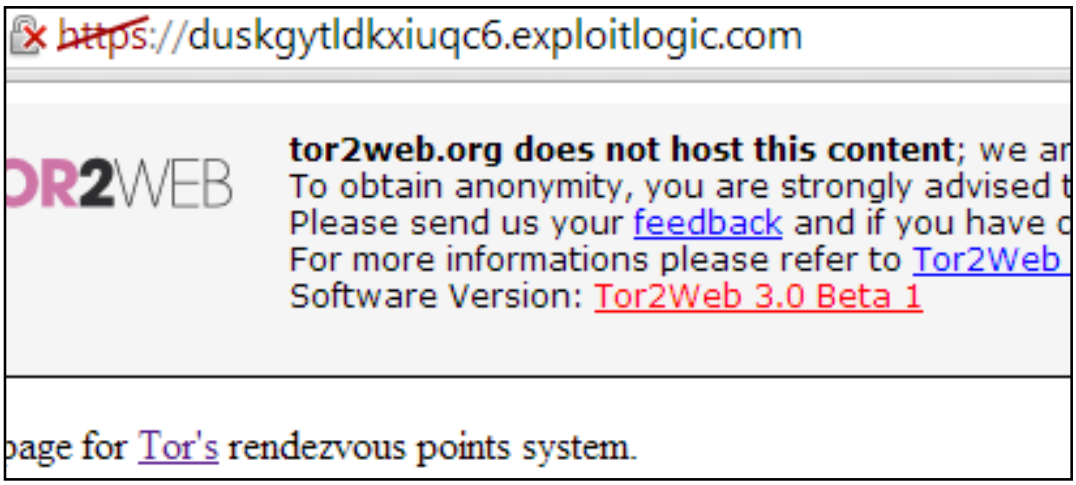
Exploitability
An attacker gathers the information on the version of server and components running with exploits that could be found in the future.

Location
https://hiddenservice.tor2web.org/ & http://tor2web.org/

Attack Vectors
Tor2Web Banner & HTTP Response Headers

Exhibits
The server header shows the version of Apache that is running.

The Tor2Web banner shows the version of Tor2Web running.




```
HTTP/1.1 200 OK
Date: Fri, 10 Jan 2014 02:24:58 GMT
Server: Apache/2.2.14
Last-Modified: Sun, 14 Jul 2013 18:46:04 GMT
ETag: "3801ad-ee2-4e17d27fd11ca"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Length: 3810
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Recommendations

Do not show the version information unless it is necessary.

Info (3 flaws)

Insecure Dependencies(3 flaws)

Associated Flaws by CWE ID:

Inclusion of Functionality from Untrusted Control Sphere (CWE ID 829)(3 flaws)

Vulnerabilities found through Manual Penetration Testing

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
6	Average	0	1	Hidden Services can Exploit Cross-Domain Access	Abuse of Functionality	

Effort to Fix: 3

Description

Hidden services can use weaknesses in unsecured browsers to identify and track users. Among more technical and well known tactics for tracking TOR users, hidden services can track users by forcing them to make cross-domain requests, such as loading a tracking image from another non-tor2web domain.

Severity Description

Viewers and document publishers to a hidden service may have their identity revealed.

Exploitability

A malicious hidden service an embed an offsite resource, such as an image, to track viewers to the service.

Location

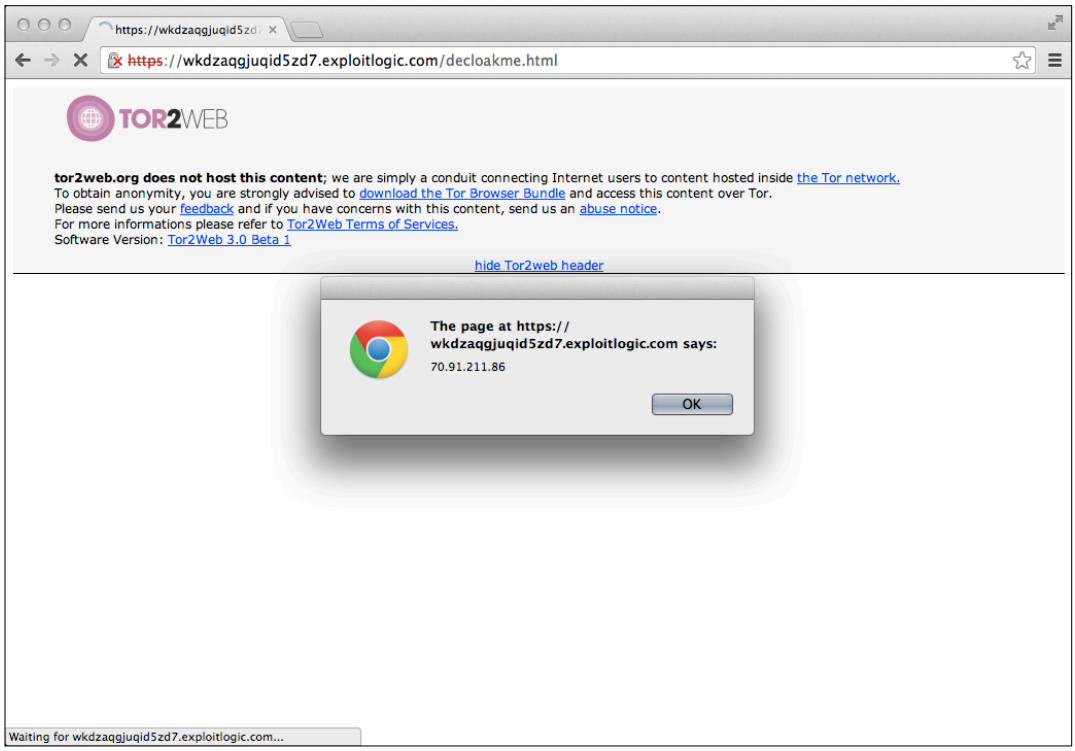
https://hiddenservice.tor2web.org/

Attack Vectors

Any malicious hidden services

Exhibits

The proof-of-concept hidden service content below sources an offsite JavaScript resource that reveals the client's IP address.




```
<html>
<body>
<script>
var myip;
function ip_callback(o) {
    myip = o.host;
}
</script>
<script src="https://smart-ip.net/geoip-json?callback=ip_callback"></script>
<script>alert(myip);</script>
</body>
</html>
```

Recommendations

Reword text on main page of site to clarify that tor2web protects publishers of hidden services, not just "publishers". This will make it clear that a person who publishes a document to another hidden service may not be protected.

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
7	Average	0	1	Hidden Services can Exploit Subdomain Access	Abuse of Functionality	

Effort to Fix: 3

Description

Sites can set cookies scoped to *.tor2web.org that may supersede other cookies set by other sister domains.

Severity Description

Hidden services can potentially compromise a user's session, or track users.

Exploitability

An attacker can set cookies that are scoped to tor2web.org that will be sent with all future requests. This means that one hidden service can set cookies that are sent to all other hidden services. It can also overwrite cookies that have been previously set by a hidden service.

Location

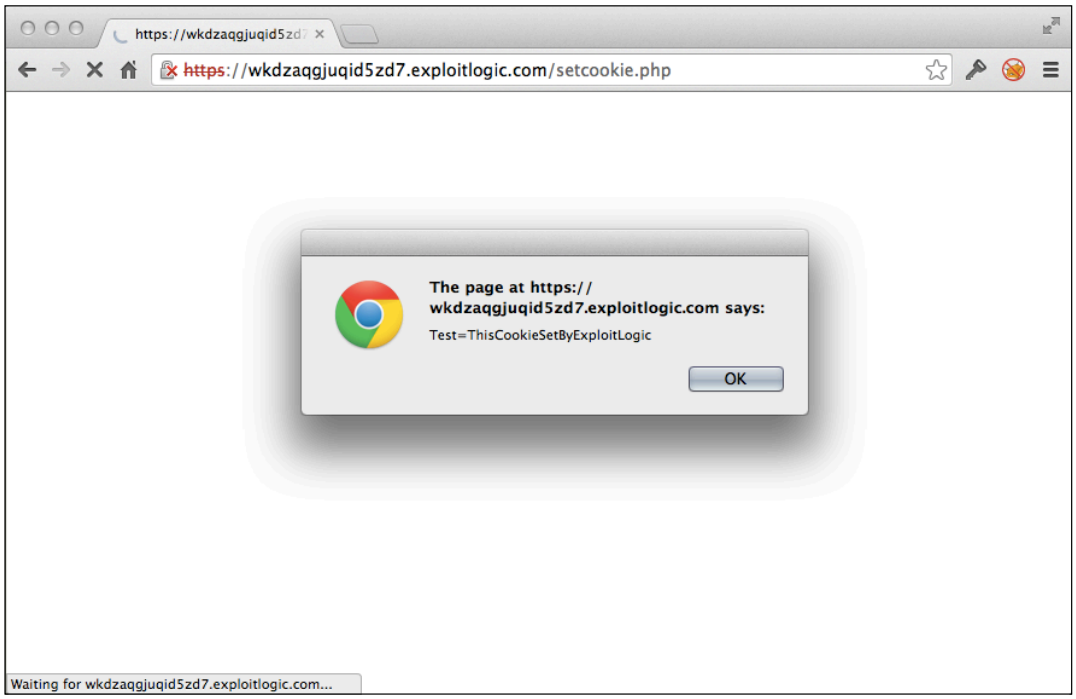
https://wkdzaqgjuqid5zd7.exploitlogic.com/setcookie.php will set a cookie that will then be sent to all hidden services.

Attack Vectors

All malicious hidden service URLs

Exhibits

An arbitrary cookie has been set below that is scoped to the parent domain.




```
HTTP/1.1 200 OK
Content-Length: 58
X-Powered-By: PHP/5.3.10-1ubuntu3.9
Set-Cookie: Test=ThisCookieSetByExploitLogic;domain=.exploitlogic.com
Strict-Transport-Security: max-age=10
Vary: Accept-Encoding
Server: Apache/2.2.22 (Ubuntu)
Date: Fri, 10 Jan 2014 21:40:30 GMT
Content-Type: text/html


<html>
<script>alert(document.cookie);</script>
</html>
```

Recommendations

Advise users with the header that it's not completely safe to use, especially for any login/authentication. Alternatively, tor2web could strip all cookies and scripts (or screenshot) webpages for complete. Recommend implementing Google Caja (or something similar) as the default mode to make webpages safe.

Flaw Id	Exploit Difficulty	CVSS	Prevalence	Name	Attack Category (CAPEC)	Fix By
8	Average	0	1	Hidden Services Can Exploit Same Domain Access (x.tor2web.org)	Abuse of Functionality	

Effort to Fix: 3

Description

Sites accessed using the same domain (x.tor2web.org) can access each other's data. For example, cookies set by x.tor2web.org/documentportal/ can be accessed by x.tor2web.org/attacker/. Additionally, same-domain requests can be made and the response can be viewed.

There are many browser security protections in place which are built around trusting the same domain. Additionally, the sites are not developed with sharing domain access with another site in mind.

Severity Description

When the "x" feature is enabled, a tor2web user could unknowingly have one site be compromised by another malicious site.

In conversations with Tor2Web staff it was determined that this was a deprecated feature.

Exploitability

The attacks that could be used include a large variety of attack types. Typical web application attacks could be used, but just logging unscoped cookies and

Location

https://x.tor2web.org/*

Attack Vectors

All URLs

Exhibits

This feature did not appear to be enabled in the development or production releases, however it is still documented at:

http://tor2web.org/security/

Recommendations

If the "x" functionality is enabled, display a banner stating that x.tor2web.org users are not protected from cross-site attacks on this domain, and that users should not authenticate to sites on this domain and are susceptible to tracking.

Trying to correctly scope cookies and prevent all types of data transfer would be nearly impossible, especially since content security policy does not allow restrictions to be put in place per path.

Alternatively, unique domains could be dynamically re-written for each user, although this would be difficult since an access cookie would need to be set for only this user to have continual access, while fulfilling the privacy goal of using the "x" domain feature.

About Veracode's Methodology

The Veracode platform uses static and dynamic analysis (for web applications) to inspect executables and identify security flaws in your applications. Using both static and dynamic analysis helps reduce false negatives and detect a broader range of security flaws. The static binary analysis engine models the binary executable into an intermediate representation, which is then verified for security flaws using a set of automated security scans. Dynamic analysis uses an automated penetration testing technique to detect security flaws at runtime. Once the automated process is complete, a security technician verifies the output to ensure the lowest false positive rates in the industry. The end result is an accurate list of security flaws for the classes of automated scans applied to the application.

Veracode Rating System Using Multiple Analysis Techniques

Higher assurance applications require more comprehensive analysis to accurately score their security quality. Because each analysis technique (automated static, automated dynamic, manual penetration testing or manual review) has differing false negative (FN) rates for different types of security flaws, any single analysis technique or even combination of techniques is bound to produce a certain level of false negatives. Some false negatives are acceptable for lower business critical applications, so a less expensive analysis using only one or two analysis techniques is acceptable. At higher business criticality the FN rate should be close to zero, so multiple analysis techniques are recommended.

Application Security Policies

The Veracode platform allows an organization to define and enforce a uniform application security policy across all applications in its portfolio. The elements of an application security policy include the target Veracode Level for the application; types of flaws that should not be in the application (which may be defined by flaw severity, flaw category, CWE, or a common standard including OWASP, CWE/SANS Top 25, or PCI); minimum Veracode security score; required scan types and frequencies; and grace period within which any policy-relevant flaws should be fixed.

Policy constraints

Policies have three main constraints that can be applied: rules, required scans, and remediation grace periods.

Evaluating applications against a policy

When an application is evaluated against a policy, it can receive one of four assessments:

Not assessed The application has not yet had a scan published
Passed The application has passed all the aspects of the policy, including rules, required scans, and grace period.
Did not pass The application has not completed all required scans; has not achieved the target Veracode Level; or has one or more policy relevant flaws that have exceeded the grace period to fix.
Conditional pass The application has one or more policy relevant flaws that have not yet exceeded the grace period to fix.

Understand Veracode Levels

The Veracode Level (VL) achieved by an application is determined by type of testing performed on the application, and the severity and types of flaws detected. A minimum security score (defined below) is also required for each level.

There are five Veracode Levels denoted as VL1, VL2, VL3, VL4, and VL5. VL1 is the lowest level and is achieved by demonstrating that security testing, automated static or dynamic, is utilized during the SDLC. VL5 is the highest level and is achieved by performing automated and manual testing and removing all significant flaws. The Veracode Levels VL2, VL3, and VL4 form a continuum of increasing software assurance between VL1 and VL5.

For IT staff operating applications, Veracode Levels can be used to set application security policies. For deployment scenarios of different business criticality, differing VLs should be made requirements. For example, the policy for applications that handle credit card transactions, and therefore have PCI compliance requirements, should be VL5. A medium business criticality internal application could have a policy requiring VL3.

Software developers can decide which VL they want to achieve based on the requirements of their customers. Developers of software that is mission critical to most of their customers will want to achieve VL5. Developers of general purpose business software may want to achieve VL3 or VL4. Once the software has achieved a Veracode Level it can be communicated to customers through a Veracode Report or through the Veracode Directory on the Veracode web site.

Criteria for achieving Veracode Levels

The following table defines the details to achieve each Veracode Level. The criteria for all columns: Flaw Severities Not Allowed, Flaw Categories not Allowed, Testing Required, and Minimum Score.

*Dynamic is only an option for web applications.

Veracode Level	Flaw Severities Not Allowed	Testing Required*	Minimum Score
VL5	V.High, High, Medium	Static AND Manual	90
VL4	V.High, High, Medium	Static	80
VL3	V.High, High	Static	70
VL2	V.High	Static OR Dynamic OR Manual	60
VL1		Static OR Dynamic OR Manual	

When multiple testing techniques are used it is likely that not all testing will be performed on the exact same build. If that is the case the latest test results from a particular technique will be used to calculate the current Veracode Level. After 6 months test results will be deemed out of date and will no longer be used to calculate the current Veracode Level.

Business Criticality

The foundation of the Veracode rating system is the concept that more critical applications require higher security quality scores to be acceptable risks. Less business critical applications can tolerate lower security quality. The business criticality is dictated by the typical deployed environment and the value of data used by the application. Factors that determine business criticality are: reputation damage, financial loss, operational risk, sensitive information disclosure, personal safety, and legal violations.

US. Govt. OMB Memorandum M-04-04; NIST FIPS Pub. 199

Business Criticality Description

Very High	Mission critical for business/safety of life and limb on the line
High	Exploitation causes serious brand damage and financial loss with long term business impact
Medium	Applications connected to the internet that process financial or private customer information
Low	Typically internal applications with non-critical business impact
Very Low	Applications with no material business impact

Business Criticality Definitions

Very High (BC5) This is typically an application where the safety of life or limb is dependent on the system; it is mission critical the application maintain 100% availability for the long term viability of the project or business. Examples are control software for industrial, transportation or medical equipment or critical business systems such as financial trading systems.

High (BC4) This is typically an important multi-user business application reachable from the internet and is critical that the application maintain high availability to accomplish its mission. Exploitation of high criticality applications cause serious brand damage and business/financial loss and could lead to long term business impact.
Medium (BC3) This is typically a multi-user application connected to the internet or any system that processes financial or private customer information. Exploitation of medium criticality applications typically result in material business impact resulting in some financial loss, brand damage or business liability. An example is a financial services company's internal 401K management system.
Low (BC2) This is typically an internal only application that requires low levels of application security such as authentication to protect access to non-critical business information and prevent IT disruptions. Exploitation of low criticality applications may lead to minor levels of inconvenience, distress or IT disruption. An example internal system is a conference room reservation or business card order system.
Very Low (BC1) Applications that have no material business impact should its confidentiality, data integrity and availability be affected. Code security analysis is not required for applications at this business criticality, and security spending should be directed to other higher criticality applications.

Scoring Methodology

The Veracode scoring system, Security Quality Score, is built on the foundation of two industry standards, the Common Weakness Enumeration (CWE) and Common Vulnerability Scoring System (CVSS). CWE provides the dictionary of security flaws and CVSS provides the foundation for computing severity, based on the potential Confidentiality, Integrity and Availability impact of a flaw if exploited.

The Security Quality Score is a single score from 0 to 100, where 0 is the most insecure application and 100 is an application with no detectable security flaws. The score calculation includes non-linear factors so that, for instance, a single Severity 5 flaw is weighted more heavily than five Severity 1 flaws, and so that each additional flaw at a given severity contributes progressively less to the score.

Veracode assigns a severity level to each flaw type based on three foundational application security requirements — Confidentiality, Integrity and Availability. Each of the severity levels reflects the potential business impact if a security breach occurs across one or more of these security dimensions.

Confidentiality Impact

According to CVSS, this metric measures the impact on confidentiality if a exploit should occur using the vulnerability on the target system. At the weakness level, the scope of the Confidentiality in this model is within an application and is measured at three levels of impact -None, Partial and Complete.

Integrity Impact

This metric measures the potential impact on integrity of the application being analyzed. Integrity refers to the trustworthiness and guaranteed veracity of information within the application. Integrity measures are meant to protect data from unauthorized modification. When the integrity of a system is sound, it is fully proof from unauthorized modification of its contents.

Availability Impact

This metric measures the potential impact on availability if a successful exploit of the vulnerability is carried out on a target application. Availability refers to the accessibility of information resources. Almost exclusive to this domain are denial-of-service vulnerabilities. Attacks that compromise authentication and authorization for application access, application memory, and administrative privileges are examples of impact on the availability of an application.

Security Quality Score Calculation

The overall Security Quality Score is computed by aggregating impact levels of all weaknesses within an application and representing the score on a 100 point scale. This score does not predict vulnerability potential as much as it enumerates the security weaknesses and their impact levels within the application code.

The Raw Score formula puts weights on each flaw based on its impact level. These weights are exponential and determined by empirical analysis by Veracode's application security experts with validation from industry experts. The score is normalized to a scale of 0 to 100, where a score of 100 is an application with 0 detected flaws using the analysis technique for the application's business criticality.

Understand Severity, Exploitability, and Remediation Effort

Severity and exploitability are two different measures of the seriousness of a flaw. Severity is defined in terms of the potential impact to confidentiality, integrity, and availability of the application as defined in the CVSS, and exploitability is defined in terms of the likelihood or ease with which a flaw can be exploited. A high severity flaw with a high likelihood of being exploited by an attacker is potentially more dangerous than a high severity flaw with a low likelihood of being exploited.

Remediation effort, also called Complexity of Fix, is a measure of the likely effort required to fix a flaw. Together with severity, the remediation effort is used to give Fix First guidance to the developer.

Veracode Flaw Severities

Veracode flaw severities are defined on a five point scale:

Severity	Name	Description
5	Very High	The offending line or lines of code is a very serious weakness and is an easy target for an attacker. The code should be modified immediately to avoid potential attacks.
4	High	The offending line or lines of code have significant weakness, and the code should be modified immediately to avoid potential attacks.
3	Medium	A weakness of average severity. These should be fixed in high assurance software. A fix for this weakness should be considered after fixing the very high and high for medium assurance software.
2	Low	This is a low priority weakness that will have a small impact on the security of the software. Fixing should be consideration for high assurance software. Medium and low assurance software can ignore these flaws.
1	Very Low	Minor problems that some high assurance software may want to be aware of. These flaws can be safely ignored in medium and low assurance software.
0	Informational	Issues that have no impact on the security quality of the application but which may be of interest to the reviewer.

Informational findings

Informational (Severity 0) Findings are items observed in the analysis of the application that have no impact on the security quality of the application but may be interesting to the reviewer for other reasons. These findings may include code quality issues, API usage, and other factors.

Informational Findings have no impact on the security quality score of the application and are not included in the summary tables of flaws for the application.

Exploitability

Each flaw instance in a static scan may receive an exploitability rating. The rating is an indication of the intrinsic likelihood that the flaw may be exploited by an attacker. Veracode recommends that the exploitability rating be used to prioritize flaw remediation within a particular group of flaws with the same severity and difficulty of fix classification.

The possible exploitability ratings include:

Exploitability	Description
V. Unlikely	Very unlikely to be exploited
Unlikely	Unlikely to be exploited
Neutral	Neither likely nor unlikely to be exploited.
Likely	Likely to be exploited
V. Likely	Very likely to be exploited

Note: All reported flaws found via dynamic scans are assumed to be exploitable, because the dynamic scan actually executes the attack in question and verifies that it is valid.

Effort/Complexity of Fix

Each flaw instance receives an effort/complexity of fix rating based on the classification of the flaw. The effort/complexity of fix rating is given on a scale of 1 to 5, as follows:

Effort/Complexity of Fix	Description
5	Complex design error. Requires significant redesign.
4	Simple design error. Requires redesign and up to 5 days to fix.
3	Complex implementation error. Fix is approx. 51-500 lines of code. Up to 5 days to fix.
2	Implementation error. Fix is approx. 6-50 lines of code. 1 day to fix.
1	Trivial implementation error. Fix is up to 5 lines of code. One hour or less to fix.

Flaw Types by Severity Level

The flaw types by severity level table provides a summary of flaws found in the application by Severity and Category. The table puts the Security Quality Score into context by showing the specific breakout of flaws by severity, used to compute the score as described above. If multiple analysis techniques are used, the table includes a breakout of all flaws by category and severity for each analysis type performed.

Flaws by Severity

The flaws by severity chart shows the distribution of flaws by severity. An application can get a mediocre security rating by having a few high risk flaws or many medium risk flaws.

Flaws in Common Modules

The flaws in common modules listing shows a summary of flaws in shared dependency modules in this application. A shared dependency is a dependency that is used by more than one analyzed module. Each module is listed with the number of executables that consume it as a dependency and a summary of the impact on the application's security score of the flaws found in the dependency.

The score impact represents the amount that the application score would increase if all the flaws in the shared dependency module were fixed. This information can be used to focus remediation efforts on common modules with a higher impact on the application security score.

Only common modules that were uploaded with debug information are included in the Flaws in Common Modules listing.

Action Items

The Action Items section of the report provides guidance on the steps required to bring the application to a state where it passes its assigned policy. These steps may include fixing or mitigating flaws or performing additional scans. The section also includes best practice recommendations to improve the security quality of the application.

Common Weakness Enumeration (CWE)

The Common Weakness Enumeration (CWE) is an industry standard classification of types of software weaknesses, or flaws, that can lead to security problems. CWE is widely used to provide a standard taxonomy of software errors. Every flaw in a Veracode report is classified according to a standard CWE identifier.

More guidance and background about the CWE is available at http://cwe.mitre.org/data/index.html.

About Manual Assessments

The Veracode platform can include the results from a manual assessment (usually a penetration test or code review) as part of a report. These results differ from the results of automated scans in several important ways, including objectives, attack vectors, and common attack patterns.

A manual penetration assessment is conducted to observe the application code in a run-time environment and to simulate real-world attack scenarios. Manual testing is able to identify design flaws, evaluate environmental conditions, compound multiple lower risk flaws into higher risk vulnerabilities, and determine if identified flaws affect the confidentiality, integrity, or availability of the application.

Objectives

The stated objectives of a manual penetration assessment are:

- Perform testing, using proprietary and/or public tools, to determine whether it is possible for an attacker to:
- Circumvent authentication and authorization mechanisms
- Escalate application user privileges
- Hijack accounts belonging to other users
- Violate access controls placed by the site administrator
- Alter data or data presentation
- Corrupt application and data integrity, functionality and performance
- Circumvent application business logic
- Circumvent application session management
- Break or analyze use of cryptography within user accessible components
- Determine possible extent access or impact to the system by attempting to exploit vulnerabilities
- Score vulnerabilities using the Common Vulnerability Scoring System (CVSS)
- Provide tactical recommendations to address security issues of immediate consequence
- Provide strategic recommendations to enhance security by leveraging industry best practices

Attack vectors

In order to achieve the stated objectives, the following tests are performed as part of the manual penetration assessment, when applicable to the platforms and technologies in use:

- Cross Site Scripting (XSS)
- SQL Injection
- Command Injection
- Cross Site Request Forgery (CSRF)

- Authentication/Authorization Bypass
- Session Management testing, e.g. token analysis, session expiration, and logout effectiveness
- Account Management testing, e.g. password strength, password reset, account lockout, etc.
- Directory Traversal
- Response Splitting
- Stack/Heap Overflows
- Format String Attacks
- Cookie Analysis
- Server Side Includes Injection
- Remote File Inclusion
- LDAP Injection
- XPATH Injection
- Internationalization attacks
- Denial of Service testing at the application layer only
- AJAX Endpoint Analysis
- Web Services Endpoint Analysis
- HTTP Method Analysis
- SSL Certificate and Cipher Strength Analysis
- Forced Browsing

CAPEC Attack Pattern Classification

The following attack pattern classifications are used to group similar application flaws discovered during manual penetration testing. Attack patterns describe the general methods employed to access and exploit the specific weaknesses that exist within an application. CAPEC (Common Attack Pattern Enumeration and Classification) is an effort led by Cigital, Inc. and is sponsored by the United States Department of Homeland Security's National Cyber Security Division.

Abuse of Functionality

Exploitation of business logic errors or misappropriation of programmatic resources. Application functions are developed to specifications with particular intentions, and these types of attacks serve to undermine those intentions.

Examples:

- Exploiting password recovery mechanisms
- Accessing unpublished or test APIs
- Cache poisoning

Spoofing

Impersonation of entities or trusted resources. A successful attack will present itself to a verifying entity with an acceptable level of authenticity.

Examples:

- Man in the middle attacks
- Checksum spoofing
- Phishing attacks

Probabilistic Techniques

Using predictive capabilities or exhaustive search techniques in order to derive or manipulate sensitive information. Attacks capitalize on the availability of computing resources or the lack of entropy within targeted components.

Examples:

- Password brute forcing
- Cryptanalysis
- Manipulation of authentication tokens

Exploitation of Authentication

Circumventing authentication requirements to access protected resources. Design or implementation flaws may allow authentication checks to be ignored, delegated, or bypassed.

Examples:

- Cross-site request forgery
- Reuse of session identifiers
- Flawed authentication protocol

Resource Depletion

Affecting the availability of application components or resources through symmetric or asymmetric consumption. Unrestricted access to computationally expensive functions or implementation flaws that affect the stability of the application can be targeted by an attacker in order to cause denial of service conditions.

Examples:

- Flooding attacks
- Unlimited file upload size
- Memory leaks

Exploitation of Privilege/Trust

Undermining the application's trust model in order to gain access to protected resources or gain additional levels of access as defined by the application. Applications that implicitly extend trust to resources or entities outside of their direct control are susceptible to attack.

Examples:

- Insufficient access control lists
- Circumvention of client side protections
- Manipulation of role identification information

Injection

Inserting unexpected inputs to manipulate control flow or alter normal business processing. Applications must contain sufficient data validation checks in order to sanitize tainted data and prevent malicious, external control over internal processing.

Examples:

- SQL Injection
- Cross-site scripting
- XML Injection

Data Structure Attacks

Supplying unexpected or excessive data that results in more data being written to a buffer than it is capable of holding. Successful attacks of this class can result in arbitrary command execution or denial of service conditions.

Examples:

- Buffer overflow
- Integer overflow
- Format string overflow

Data Leakage Attacks

Recovering information exposed by the application that may itself be confidential or may be useful to an attacker in discovering or exploiting other weaknesses. A successful attack may be conducted passive observation or active interception methods. This attack pattern often manifests itself in the form of applications that expose sensitive information within error messages.

Examples:

- Sniffing clear-text communication protocols
- Stack traces returned to end users
- Sensitive information in HTML comments

Resource Manipulation

Manipulating application dependencies or accessed resources in order to undermine security controls and gain unauthorized access to protected resources. Applications may use tainted data when constructing paths to local resources or when constructing processing environments.

Examples:

- Carriage Return Line Feed log file injection
- File retrieval via path manipulation
- User specification of configuration files

Time and State Attacks

Undermining state condition assumptions made by the application or capitalizing on time delays between security checks and performed operations. An application that does not enforce a required processing sequence or does not handle concurrency adequately will be susceptible to these attack patterns.

Examples:

- Bypassing intermediate form processing steps
- Time-of-check and time-of-use race conditions
- Deadlock triggering to cause a denial of service

Terms of Use

Use and distribution of this report are governed by the agreement between Veracode and its customer. In particular, this report and the results in the report cannot be used publicly in connection with Veracode's name without written permission.