Vulnerability Details Report (Sites)

Atlassian

Report As Of Wednesday, February 17, 2016

Prepared By security@atlassian.com

Report Description The Vulnerability Details report provides detailed descriptions of the vulnerabilities found on the sites selected for this report, grouped by vulnerability class. Each report section contains a full description of the vulnerability class, remediation instructions for that class, and a list of specific instances of that vulnerability on each site. Note that this report is available for Sentinel (dynamic testing) only, since it is based on an assessment of the production or pre-production site.
This report is intended for security team members, development managers and developers.

Notes Sites are assessed using dynamic analysis, and vulnerabilities are rated by their risk levels. For descriptions of dynamic analysis and risk levels, please see the Appendix.

Report Filtered By

Vulnerability Status	Open
Vulnerability Rating	Critical, High, Medium, Low, Note
Start Date	2001-01-01
End Date	2016-02-18

Assets Number of Sites 2

Selected Vulnerability Classes

Routing Detour	Brute Force	Insufficient Password Policy	URL Redirector Abuse
Query Language Injection	Autocomplete Attribute	Insufficient User Session Invalidation	Insufficient Session Invalidation
Weak Cipher Strength	Insufficient Transport Layer	Path Traversal	Session Fixation
Insufficient Process Validation	SSI Injection	Insufficient Authentication	HTTP Response Splitting
Denial of Service	Insufficient Authorization	Directory Traversal	Predictable Resource Location
OS Command Injection	Insufficient Session Expiration	Buffer Overflow	Information Leakage
LDAP Injection	OS Commanding	XPath Injection	Frameable Response
Mixed Content Security	Abuse of Functionality	Directory Indexing	Application Misconfiguration
Cacheable Sensitive Response	Fingerprinting	Persistent Session Cookie	Personally Identifiable Information
SQL Injection	Unsecured Session Cookie	Session Prediction	Non-HttpOnly Session Cookie
Insufficient Cookie Access Control	Insufficient Crossdomain	Improper Input Handling	Insecure Indexing
Format String Attack	Mail Command Injection	Remote File Inclusion	Server Misconfiguration
XML External Entities	XML Injection	XQuery Injection	Improper HTTP Method Usage
Content Spoofing	Cross Site Scripting	Cross Site Request Forgery	Insufficient Password Recovery
Frameable Resource	Configuration	Application Code Execution	Missing Transportation Layer
Unpatched Software Version	Clickjacking	Insufficient Anti-automation	

The Index of Content can be found on the last page

Asset List

This report has been generated for following assets:

Sites:

marketplace.atlassian.com	wh.atlassian.net

This table sorts by the importance (score) of your site set by vulnerability assessor. The higher the score, the more important the site. Your vulnerabilities are then categorized by the vulnerability class and then by the vulnerability level.

	Critical	High	Medium	Low	Note
Site Priority: 3	**0**	**1**	**1**	**0**	**2**

M VULN ID - 49535692

Vulnerability Class	Insufficient Anti-automation
Status	Open
URL	wh.atlassian.net/browse/SSP-1
Rating	Medium
Opened	2015-07-06 08:04:13 -0700
Custom Description	There is no anti automation protection on the functionality to share issues with other users. We were able to repeat the same request over 25 times and successfully send another user multiple emails. UPDATED 01/12/2015 - POC below 1. Log in to the application as testing4@whitehatsec.com 2. Click share button to share an issue with another user. 3. Fill out email address field with a valid email 4. Using an intercepting proxy intercept the request after clicking Share button. 5. Repeat the request several times and notice the email being sent each time. URL: https://wh.atlassian.net/rest/share/1.0/issue/SSP-1 POST: {"usernames":[],"emails":["INSERT YOUR EMAIL TO SHARE WITH HERE"],"message":"email abuse test"}
Custom Solution	The application should employ anti-automation measures. ----------

N VULN ID - 50076250

Vulnerability Class	Fingerprinting
Additional Information	version_number
Status	Open
URL	wh.atlassian.net
Rating	Note
Opened	2016-02-03 13:16:48 -0800
Custom Description	We have identified version information being disclosed in the response body. An attacker can use this information to search for known exploits specific to the technology.
Custom Solution	Remove any specific information, at a minimum the version number, from being disclosed.

N VULN ID - 49535710

Vulnerability Class	Cross Site Scripting
Status	Open
URL	wh.atlassian.net/rest/greenhopper/1.0/cardlayout/6/workMode/field/1
Rating	Note
Opened	2015-07-06 08:21:13 -0700
Custom Description	It is possible to use the application to execute malicious scripts. Date: 07/06/2015 Browser: Firefox 38.0.5 Vulnerable Parameter: N/A - User Agent Description of how to reproduce: 1. Append "><marquee onstart=prompt()> to your user agent string. 2. Log in with admin credentials (creds used testing3@whitehatsec.com) 3. Navigate to https://wh.atlassian.net/secure/RapidView.jspa?rapidView=6&tab=cardLayout 4. Add a field and then delete it but be ready to drop the request in a proxy 5. An error message will be displayed at the foot of the page, click on details and the injection will fire. Proof of Concept: N/A - Self xss, informational finding. ----------
Custom Solution	Sanitize and output encode all user supplied input.

H VULN ID - 49535724

Vulnerability Class	Cross Site Request Forgery
Status	Open
URL	wh.atlassian.net/wiki/admin/questions/doeditpermissions.action
Rating	High
Opened	2015-07-06 08:38:20 -0700
Custom Description	---------- ** Update: 02 / 02 / 2016. The creds "testing3@whitehatsec.com" are admin creds ---------- Description: The request to "add question permissions" is vulnerable to Cross-Site Request Forgery attacks, the anti xsrf token is not checked for this request and can be removed. Steps to reproduce 1. Log in using admin credentials. 2. send the following post request: URL: https://wh.atlassian.net/wiki/admin/questions/doeditpermissions.action BODY: confluence_checkbox_useconfluencequestions_group_administrators=on&confluence_initial_useconfluencequestions_group_administrators=on&confluence_checkbox_useconfluencequestions_group_confluence-users=on&confluence_initial_useconfluencequestions_group_confluence-users=on&confluence_checkbox_useconfluencequestions_group_system-

H VULN ID - 49535724

administrators=on&confluence_initial_useconfluencequestions_group_system-administrators=on&groupsToAdd=&usersToAdd=testing1%40whitehatsec.com&usersToAddButton=Add
3 navigate to https://wh.atlassian.net/wiki/admin/questions/viewpermissions.action and confirm testing1@whitehatsec.com now has individual user permissions.

Custom Solution

Cross site request forgery can only succeed if an attacker can predict all necessary parameters. It can therefore be prevented by requiring something that the attacker has no way of knowing. This could be a randomized token (usually called a CSRF token), a CAPTCHA, the user's password, or any other piece of information the attacker cannot predict. Ensure that the application actually enforces the presence of the unpredictable information, that the token or CAPTCHA value has sufficient entropy, and that tokens cannot be re-used or otherwise predicted.

Cross Site Request Forgery

Description
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 is primarily used to perform an action against a target site using the victim's privileges. 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.

References
www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

projects.webappsec.org/Cross-Site-Request-Forgery

Solution
CSRF attacks target predictable aspects of an application. The key CSRF defensive pattern is the token synchronizer pattern, which adds a new requirement to every sensitive application feature: basically, adding a cryptographically random token. This random token adds an unpredictable requirement throughout the application so that unauthorized commands without the accompanying proper token value are rejected. Instead, only application features submitted with the proper token are allowed to execute.

Therefore, when a new user session is initiated, a secondary, cryptographically strong nonce should be generated for use during a particular user's current session. This nonce should be embedded as a hidden input element in any form that performs a sensitive action. When the form is submitted the application should check for the correct nonce before executing the request. To further enhance the security of the application against CSRF, consider generating a unique nonce for every sensitive action. This would result in the generation of a unique nonce per request as opposed to a unique nonce per session. It is important to note, however, that using unique nonces per request can result in usability concerns such as the "Back" browser button possibly resulting in a form with an invalid nonce. When using nonces for CSRF protection it is important that no Cross-Site Scripting vulnerabilities are present on the application, since XSS can be used to read nonces from responses which defeats their use.

Random tokens should be added to all sensitive features. It is a convention that GET requests "SHOULD NOT have the significance of taking an action other than retrieval" per http:/ /www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1
; therefore, only POST requests should require CSRF tokens. However, some security teams use GET requests extensively for data manipulation operations. In these cases, for added security it is best to add CSRF random token requirements to all GET requests, even though token leakage is a potential side effect (residual risk). Specifically, CSRF tokens on GET requests can have a negative impact on usability and scalability. For instance, Content Delivery Services such as Akamai depend on predictable URLs for the purpose of global load balancing and caching. CSRF tokens can also damage the functionality of URL bookmarking, because CSRF tokens on URLs expire and become invalid over time.

Another option is to use a CAPTCHA, which can help ensure that a transaction is not being automated and, instead, has been validated by a human.

Finally, extremely sensitive transactions can require that users submit their authentication information to confirm that they want to conduct a transaction. While this adds a higher level of security, it also restricts the ease of use and therefore should generally be reserved for sensitive transactions.

References
www.owasp.org/index.php/Cross- Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet

www.jtmelton.com/2010/05/16/the-owasp-top-ten-and-esapi-part-6-cross-site- request-forgery-csrf/

projects.webappsec.org/w/page/13246919/Cross-Site-Request-Forgery

cwe.mitre.org/data/definitions/352.html

capec.mitre.org/data/definitions/62.html

Cross Site Scripting

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 be 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 that, 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 be interpreted by the user's browser and executed. 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 will be 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.

References
www.owasp.org/index.php/Cross-site_Scripting_(XSS)

projects.webappsec.org/Cross-Site+Scripting

Solution
XSS is introduced into the application through untrusted user input. Most XSS threats can be avoided by: (1) Limiting both where and how untrustworthy input is used; and (2) ensuring that all user input is strongly validated and contextually encoded.

Use positive whitelist validation for all user input. For example, use built-in optimized platform-specific parsers (e.g., Integer.parseInt()) or regular expressions). Be aware that regular expressions are CPU intensive, and that they can be used by attackers to tie up system resources in a denial-of-service attack.

If the application's framework supports casting, then ensure that all types (booleans, integers, floats, etc.) are cast. For rich input validation, an HTML policy engine such as AntiSamy must be used to ensure that the rich input does not contain spoofed content or XSS.

Encoding must be done contextually, making note of the following contexts:

- String: validate and encode based on context
- CSS: validate and remove "expression" call, plus XSS HEX encode
- HTML Body: HTML Entity encode
- HTML Attribute: Aggressive HTML Entity Encoding, even encoding spaces
- URL Attribute: Validate that the URL is legal and only contains safe protocols (and that the URL is specifically NOT a javascript:// url); then attribute encode
- JavaScript: JS Output encode and ensure certain functions like eval() or setTimeout() are not used.

References

www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

projects.webappsec.org/w/page/13246920/Cross-Site-Scripting

capec.mitre.org/data/definitions/18.html

capec.mitre.org/data/definitions/19.html

capec.mitre.org/data/definitions/63.html

cwe.mitre.org/data/definitions/79.html

Fingerprinting

Description
The most common methodology for attackers is to begin by getting the target's web presence "footprint" and enumerating as much information as possible -- e.g. the target's platform, web application software technology, backend database version, configurations and possibly even their network architecture/topology. With this information, the attacker may develop an accurate attack scenario, which will effectively exploit a vulnerability in the software type/version being utilized by the target host.

References
projects.webappsec.org/Fingerprinting

Solution
The goal of Fingerprinting is to determine what technologies are being utilized or exposed via the application, and specifically what version of software is available to the attacker. Once the technology and version are determined the attacker can review known vulnerabilities based on that data. Therefore, it is critical to limit the exposure of information as to what software version is being utilized and, when possible, what software/technology.

References
http://projects.webappsec.org/Fingerprinting

Insufficient Anti-automation

Description
Insufficient Anti-automation occurs when a web application permits an attacker to automate a process that was originally designed to be performed only in a manual fashion, i.e. by a human web user.

References
projects.webappsec.org/Insufficient+Anti-automation

Solution
A wide variety of application code patterns can introduce automation vulnerabilities into a Web application. Therefore, the only way to stop all automated attacks is to build a defense-in-depth net of standard security controls within your Web application.

Note that all sensitive operations used to view or modify data must be located behind a properly designed strong authentication layer.

One way to prevent automated attacks against sensitive features is to require re-authentication before any feature can be executed. Re-authentication is often used in "password reset" features that require confirmation of the "old" password before allowing the user to reset to a "new" password. All e-Commerce applications should also be protected by forced re-authentication at several boundaries within the application. These boundaries should include changing an email address, shipping to a new address, or the management of any credit card information.

Perimeter technologies such as geocaching and proxy services are also good solutions for anti-automation and can shield an application from attack without requiring changes to the application itself.

Another defense is request rate throttling which would be limiting the number of requests a certain user or session can submit within a certain time period. For example you could implement a limit of how many requests a user can send per minute as well as a maximum request limit per day.

CAPTCHAs are one fairly common anti-automation soltuion used to protect public pages such as an account registration form. CAPTCHA is an acronym for "Completely Automated Public Turing Test to Tell Computers and Humans Apart". Common CAPTCHA mechanisms include distorted text within image files that is difficult for automated software to interpret, but reasonably easy for humans to read. Other CAPTCHAs include analyzing sound files or answering security questions that only humans can reliably solve.

References
projects.webappsec.org/w/page/13246938/Insufficient%20Anti-automation

cwe.mitre.org/data/definitions/799.html

cwe.mitre.org/data/definitions/804.html

Appendix - Assessment Methodology for Dynamic Analysis

WhiteHat Security combines a proprietary vulnerability scanning engine with human intelligence and analysis from its Threat Research Center to deliver thorough and accurate assessments of web applications with its Sentinel Service.

WhiteHat Sentinel dynamic scanning services are all based on a continuously evolving top of class scanning engine with manual verification of all vulnerabilities to ensure quality results. WhiteHat's model allows customers to keep all sites covered at all times with minimal investment of personnel, while having access to the worlds largest team of web application security experts who keep on top of the latest web security issues, manage security assessments for customers, and provide support and information. With Premium service the security experts in the Threat Research Center also perform business logic assessments of sites, which may uncover additional issues which cannot be found through automatic scanning. This combination provides the highest quality of security assessments in the industry with high scalability and ease of use, to keep customers on top of their risk posture and help them secure their assets.

Appendix - Vulnerability Level Definitions (by Risk)

Risk Levels for the WhiteHat Sentinel Source solution are based on the OWASP risk rating methodology, which is summarized below. The methodology is essentially based on the standard risk model (Risk = Likelihood x Impact) with several factors contributing to the likelihood and impact.

The likelihood can be broken down into Threat Agent Factors and Vulnerability Factors. By analyzing the threat agent, we estimate the likelihood of a successful attack. By analyzing the vulnerability factors, we estimate the likelihood of the vulnerability being discovered and exploited. Because there are so many possible threat agents and factors involved, the worst case scenario is used when determining likelihood. Factors that influence the threat agent include: skill level, motive, opportunity, and size of the threat. Factors that influence the vulnerability include: easy of discovery, ease of exploit, awareness, and intrusion detection.

The Impact can be broken down into the Technical Impact and Business Impact. Technical impact considers the traditional areas of security: confidentiality, integrity, availability, and accountability. The business impact stems from the technical impact and consider things such as: financial damage, reputational damage, non-compliance, and privacy violations.

High	Medium	Low
6 - 9	3 - 5	0 - 2

After scoring the Likelihood and Impact, the Risk Rating is determined using the following table:

Impact	Likelihood		
	Low	Medium	High
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Note or Low	Low	Medium

Risk ratings are defined below:

Rating	Description
Critical	A vulnerability that could have a catastrophic impact if the attack succeeds, and the vulnerability is easy to identify and exploit. The vulnerability likely affects all or many users. The vulnerability poses an immediately danger and should be mitigated immediately - In some cases, the application should even be taken offline.
High	A vulnerability that is likely to have a significant impact if the attack succeeds and the vulnerability is fairly easy to identify and exploit. The vulnerability may affect more than one user. The vulnerability should be mitigated as soon as possible.
Medium	A vulnerability that is likely to have a moderate to significant impact if the attack succeeds, but may be difficult to identify or exploit or only affects a small number of users. The vulnerability should be mitigated relatively soon.
Low	A vulnerability that is likely to have a low to moderate impact if the attack succeeds, but is difficult to identify or exploit, or only affects a small number of users. The vulnerability should be mitigated if there is time and whenever it is convenient (e.g. next release)
Note	A finding that does not pose any risk for the application and does not need to be fixed. However, it is something that should be considered to further improve security from an already acceptable level

About WhiteHat Security

WhiteHat Security is the leading provider of application risk assessment and management services that enable customers to protect critical data, ensure compliance, and narrow windows of risk. By providing accurate, complete, and cost-effective application vulnerability assessments as a software-as-a-service, we deliver the visibility, flexibility, and guidance that organizations need to prevent web attacks.

Deloitte, SC Magazine, the San Jose/Silicon Valley Business Journal, Gartner and the American Business Awards have all recognized WhiteHat Security for our remarkable innovations, executive leadership and our ability to execute in the application security market.

To learn more about WhiteHat Security and how our solutions can support your applications throughout the entire software development lifecycle, please visit our website at www.whitehatsec.com.

Contents