Application security in

ADSelfService Plus

ManageEngine
ADSelfService Plus

www.adselfserviceplus.com

Table of Contents

Application security and its importance

In today's digital business landscape, web applications have become inviting targets for attackers. According to the Verizon's 2018 Data Breach Investigations Report, 25 percent of data breaches targeted web applications. Every day, new hacks and attacks are deployed to exploit the security vulnerabilities in web applications. With new vulnerabilities being exposed at a rate most organizations can't keep up with, it isn't surprising that application security has emerged as one of the leading factors impacting a company's brand perception.

That's why the ADSelfService Plus' team focuses on patching identified vulnerabilities and security loopholes when they are detected in the product. The list below describes common application security issues that were found in ADSelfService Plus, from newest to oldest, and how each issue is addressed. Remember that if you configure any XML files to fix an issue, make sure to restart ADSelfService Plus so the changes can take effect.

HttpOnly flag missing from cookies

Severity: Low

The absence of the *HttpOnly* flag in cookies increases the risk of a client-side script accessing cookies, which can lead to a cross-site request forgery (CSRF) attack.

Fix: ADSelfService Plus includes the *HttpOnly* flag in cookies. When a client-side script attempts to read the cookie, the browser returns an empty string as a result. ADSelfService Plus fixed this vulnerability in build 5520, on May 31, 2018.

Exploiting the unused HTTP methods

Severity: Low

HTTP methods such as GET, HEAD, TRACE, PUT, DELETE, and OPTIONS are subject to attacks and pose security threats to web applications. For instance, TRACE is used to echo a string sent to the web server back to the client. Though TRACE was initially crafted for debugging purposes, it can be used to mount a cross-site tracing (XST) attack against servers.

Fix: ADSelfService Plus blocks the unused HTTP methods like GET, HEAD, DELETE TRACE, and OPTIONS. ADSelfService Plus fixed this vulnerability in build 5517, on April 17, 2018.

Vulnerabilities in the older versions of jQuery

Severity: High

Earlier versions of jQuery contain security vulnerabilities.

Fix: ADSelfService Plus has upgraded the jQuery bundle from 1.8.1 to 1.12.2 in build 5517, on April 17, 2018.

Unrestricted file upload vulnerability

Severity: High

In this type of vulnerability, an attacker uploads a multipart or form-data POST request with a specially-crafted filename or MIME type, which leads to cross-site scripting (XSS) and execution of malicious code on the server's side.

Fix: ADSelfService Plus uses a whitelist filter during file uploads. It only accepts PNG, HTML, CSV, PDF, XLS, XLXS, and CSVDE formats. ADSelfService Plus fixed this vulnerability in build 5516, on March 29, 2018.

Server-side request forgery vulnerability

Severity: High

In a server-side request forgery (SSRF) attack, an attacker modifies an existing URL or provides a new URL to be sent to the server. When this manipulated URL request is handled by the server, the server reads or submits data to the manipulated URL. Usually, the attacker targets the NTLM hash of specific accounts to access the resources linked to that account.

Fix: ADSelfService Plus has upgraded the **dd-plist.jar** file **(default location:** Installation Directory\lib\dd-plist.jar) in build 5516, on March 29, 2018.

Reflected XSS vulnerability

Severity: High

The reflected XSS vulnerability is specifically designed to attack websites a user is visiting. When a user clicks on a malicious link in a trusted site, a script is injected into the request, which travels to the server and gets reflected off in such a way that the HTTP response includes the malicious script. The browser executes the malicious script because that script came from a "trusted" server.

Fix: ADSelfService Plus sanitizes the script of characters like **<, >, &,** ', and " present in the query parameters. ADSelfService Plus fixed this vulnerability in build 5516, on March 29, 2018.

Bypassing client-side validations

Severity: High

By exploiting this vulnerability, an attacker bypasses the client-side input validation for targeted content, say, password fields.

Attackers usually bypass a web application's input validations by either removing JavaScript using a web developer tool or by handling the HTTP request (using a proxy tool) in a way that it does not go through the browser.

Fix: There's no fix needed for this vulnerability. ADSelfService Plus is immune to this vulnerability as it practices both client-side and server-side validation.

Information leakage through comments

Severity: Low

Information leakage occurs when an application unintentionally discloses sensitive data, such as the technical details of a network or application, or user-specific data. Depending on what data is leaked, it could be used by an attacker to exploit the target web application, its hosting network, or the application's users.

Fix: ADSelfService Plus' programmers have made sure to remove sensitive information that might've been disclosed through comments in the source code.

Fingerprinting the web server

Severity: Low

Exploiting the security vulnerabilities of any application is easier when the attackers know the platform on which the web application is built. Though the HTTP headers are mostly used to provide information for effective handling of requests and responses, they can also be exploited by attackers to identify the web server used and its version.

Fix: No fix is needed for this vulnerability. ADSelfService Plus is immune to this vulnerability as it adds a server tag in the *server.xml* file (**default location:** Installation Directory/conf) to hide the actual web server.

Sample:

<Connector port="8888" name="WebServer" maxThreads="150" minSpareThreads="25" maxSpareThreads="75" enableLookups="false" redirectPort="8443" acceptCount="100" connectionTimeout="20000" disableUploadTimeout="true" URIEncoding="UTF-8" server="ADSSP"/>

Simultaneous session logins

Severity: Medium

An application designed to accept concurrent logins can lead to a malicious user inputting valid credentials at the same time as that of a legitimate user to authenticate themselves in the network. This could lead to security issues within the organization like misuse of the user's personal information to perform unauthorized actions.

Fix: ADSelfService Plus' **Deny Concurrent Login** feature prevents users from running multiple sessions at once in the product.

ADSelfService Plus fixed this vulnerability in build 5517, in April 2018.

Cross-site scripting (XSS) vulnerabilities

Severity: High

XSS attacks involve an attacker injecting a client-side script in the target application. The end user's browser has no way of knowing that the script shouldn't be trusted, and will execute the malicious script.

Fix: Remove the # at the beginning of **X-XSS-Protection** in the security_params.xml file (**default location:** Installation Directory/conf) and set it to **1.** Most browsers recognize this header, and they will take necessary actions to prevent XSS attacks upon seeing this header.

Below is what the header will look like after the fix:

X-XSS-Protection=1

ADSelfService Plus fixed this vulnerability in build 4500.

Cross-site request forgery (CSRF) vulnerability

Severity: High

CSRF is an attack that tricks a web browser into executing an unwanted command in an application that a user is logged in to. This is accomplished by a user inadvertently clicking a malicious link on a legitimate website. This sends a HTTP request the user did not intend to raise, which includes a cookie header that contains the user's session ID. Also, because the application authenticates the user at the time of the attack, it's impossible for the application to distinguish between legitimate and forged requests.

Fix: ADSelfService Plus sends out every request with a token. This wards off the execution of actions that do not provide necessary authentication tokens.

ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

Cross-frame scripting (XSF)/Clickjacking

Severity: High

In cross-frame scripting, a user is tricked into clicking on something different than what they thought they were clicking on, making them inadvertently reveal sensitive information or execute an unintended command. Typically, cross-frame scripting is accomplished when an attacker embeds malicious iFrames in a legitimate website to deceive users into entering their information. When a user enters their credentials into the legitimate site within the iFrame, the malicious JavaScript keylogger records the victim's keystrokes and sends them to the attacker's server.

Fix: Remove the # at the beginning of the **x-frame-options** in the *security_params.xml file* (**default location:** Installation Directory/conf) and set it to **SAMEORIGIN.** This fix does not allow other sites to load ADSelfService Plus in their iFrames.

Below is what the header request will look like after the fix:

x-frame-options=SAMEORIGIN

ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

Weak cache policy or server cache policy

Severity: Medium

A browser page stores cached content on a user's machine so that it doesn't have to download content every time the user opens that page. Even in secure SSL channels, sensitive data can be stored by proxies and SSL terminators. If an attacker exploits the browser's cache, sensitive data such as credit card details and usernames are at risk.

Fix: Every HTTP page in ADSelfService Plus is set with Cache-Control, Pragma, and Expires response headers to prevent caching of any data. To enable this fix, you'll have to remove the # at the beginning of **cache-control=no-cache, no-store** in the *security_params.xml* file (**default location:** Installation Directory/conf).

Below is what the header request will look like after the fix:

cache-control=no-cache, no-store

ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

MIME-SNIFFING

Severity: Low

When there isn't enough metadata to determine the content type of data, most browsers, notably Microsoft Internet Explorer, attempt to determine the correct content type with a technique called MIME (also known as media type) sniffing. However, attackers exploit this technique by manipulating the browser to interpret data in a way that allows for unexpected operations, such as cross-site scripting.

Fix: Remove the # at the beginning of the **x-content-type** and set it to **nosniff** in the *security_params.xml* file.

Below is what the header request will look like after the fix:

x-content-type=nosniff

ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

Insecure wildcard cross-origin resource sharing (CORS)

Severity: High

Cross-origin resource sharing (CORS) is a standard that defines a set of headers that allow a server and a browser to determine which requests for cross-domain resources are permitted and which are not. The downside of this standard is that it fails to validate/whitelist requestors when the **access-control-allow-origin** is set to '*****'. This symbol is a wildcard and setting access control to ***** essentially allows any domain on the web to access that site's resources.

Fix: Set the **access-control-allow-origin** to a specific **domain name** to fix the CORS vulnerability. ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

Browser auto-complete issue

Severity: Low

Most browsers make a cached copy of a user's credentials that are entered into HTML forms. This function stores credentials on a user's machine, enabling a faster response the next time that user attempts to access the application. This vulnerability can be exploited by an attacker with local access, allowing them to view clear text passwords from the browser cache.

Fix: ADSelfService Plus doesn't allow the auto-complete feature to be used in its password fields.

ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

Missing HTTPOnly flag and secure flag in the session cookies

Severity: Low

If a session cookie doesn't have an *HttpOnly* flag, the cookie can be accessed through JavaScript. Essentially this means that an XSS attack could lead to cookies being stolen, which in turn could lead to an account or session takeover.

Fix: Enable SSL in ADSelfService Plus, then set the *HTTPOnly* flag and the *secure* flag for session cookies. Doing so makes the browser return an empty string as a result when a client-side script attempts to read cookies. ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

SHA1WithRSA vulnerabilities

Severity: High

Using SHA1WithRSA causes a collision vulnerability, which allows an attacker to create two input strings with the same SHA-1 hash with less computational power than it should take for a good hash function.

Fix: ADSelfService Plus uses *SHA256WithRSAENCRYPTION* by default to overcome this security vulnerability.

ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

Session fixation

Severity: High

In this vulnerability, an attacker targets the limitations of the application's session ID management. When the malicious user visits the application, they're assigned a session ID. The attacker makes note of this session ID and leaves the browser open. If another user on the same machine authenticates themselves in the application without closing the browser, they'll be logged in with the session ID set by the attacker. The attacker can use this information to gain complete access to the user's application account until that session ends. This can lead to potential security issues as the attacker can use this access to change the user's password.

Fix: ADSelfService Plus creates new session IDs for every successful authentication (i.e. for every new session).

ADSelfService Plus fixed this vulnerability in build 5300, in April 2015.

SQL Injection through framework build

Severity: High

SQL injection occurs when an attacker adds or injects malicious code into a SQL statement executed by the web application. A successful SQL injection allows attackers to spoof a user's identity, tamper with existing data, and even gain complete control over the web application's server.

Fix: Database operations for ADSelfService Plus are handled through our internal framework to prevent SQL injections and other similar attacks.

Weak SSL cipher

Severity: High

Every application depends on the protection of three parameters, known collectively known as a cipher suite: authentication, encryption, and hashing algorithms. An application relying on SSL/TLS for data transmissions with weak ciphers leaves the application unprotected and allows an attacker to steal or manipulate sensitive data.

Fix: Add the strong ciphers provided below to ADSelfService Plus in the server.xml file (**default location:** Installation Directory/conf).

ciphers="TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA"

allowUnsafeLegacyRenegotiation="false" server="Adselfservice Plus"

sslProtocol="TLS" sslEnabledProtocols="TLSv1.2"

compression="off"

Example:

<Connector SSLEnabled="true" acceptCount="100" compression="off"
ciphers="TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_
128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH
_AES_256_CBC_SHA" clientAuth="false" connectionTimeout="-1" debug="0"
disableUploadTimeout="true" enableLookups="false"
keystoreFile="./conf/server.keystore" keystorePass="adventnet" maxSpareThreads="75"
maxThreads="150" minSpareThreads="25" name="SSL" port="9251" scheme="https"
secure="true"allowUnsafeLegacyRenegotiation="false" server="AdselfservicePlus"
sslProtocol="TLS" sslEnabledProtocols="TLSv1.2"/>