Kentico Cloud Security review

Author: Kentico Security Team

Date: 01/23/2018

Contents

I. Session management review

1. Testing for Bypassing Session Management Schema (OTG-SESS-001)

In order to avoid continuous authentication for each page of a website or service, web applications implement various mechanisms to store and validate credentials for a pre-determined timespan. These mechanisms are known as Session Management and while they are important in order to increase the ease of use and user-friendliness of the application, they can be exploited by a penetration tester to gain access to a user account, without the need to provide correct credentials.

Cookies collection

Are all Set-Cookie directives tagged as Secure?
The Kentico Cloud clients communicate with a backend API using Bearer authentication. The server's protected routes will check for a valid JWT in the Authorization header, and if it's present, the user will be allowed to access protected resources. The 3rd party authentication (Auth0) cookie is secured using Secure attribute.

Do any Cookie operations take place over unencrypted transport?
No, Kentico Cloud uses secure SSL connection.

Can the Cookie be forced over unencrypted transport?
No, Kentico Cloud uses secure SSL connection.

Are any Cookies persistent?
Yes but only 3rd cookies are present (inline manual / intercom).

2. Testing for Cookies Attributes (OTG-SESS-002)

Cookies are often a key attack vector for malicious users (typically targeting other users) and the application should always take due diligence to protect cookies. This section looks at how an application can take the necessary precautions when assigning cookies, and how to test that these attributes have been correctly configured.

HttpOnlyAttribute
Kentico Cloud does not use cookies for authentication. There are only 3rd party cookies (inline manual /intercom).

Secure Attribute
Kentico Cloud does not use cookies for authentication. There are only 3rd party cookies (inline manual /intercom).

3. Testing for Session Fixation (OTG-SESS-003)

When an application does not renew its session cookie(s) after a successful user authentication, it could be possible to find a session fixation vulnerability and force a user to utilize a cookie known by the attacker. In that case, an attacker could steal the user session (session hijacking).

Existing SessionID is invalidated when the user is authenticated?
Kentico Cloud uses Bearer token authentication. The only one session cookie is 3rd party – Auth0 identity provider. This cookie is protected against session fixation attacks.

4. Testing for Exposed Session Variables (OTG-SESS-004)

The Session Tokens (Cookie, SessionID, Hidden Field), if exposed, will usually enable an attacker to impersonate a victim and access the application illegitimately. It is important that they are protected from eavesdropping at all times, particularly whilst in transit between the client browser and the application servers.

Sensitive data stored in session
Kentico Cloud does not store sensitive information like credit card numbers in a session container.

5. Testing for Cross Site Request Forgery (CSRF) (OTG-SESS-005)

CSRF is an attack which 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, when it targets a normal user. If the targeted end user is the administrator account, a CSRF attack can compromise the entire web application.

Not applicable
All requests in Kentico Cloud have to be authenticated using Bearer authentication. This implies that an attacker is not able to forge requests to the backend API without knowing the actual authentication token.

6. Testing for Logout Functionality (OTG-SESS-006)

Session termination is an important part of the session lifecycle. Reducing to a minimum the lifetime of the session tokens decreases the likelihood of a successful session hijacking attack. This can be seen as a control against preventing other attacks like Cross Site Scripting and Cross Site Request Forgery. Such attacks have been known to rely on a user having an authenticated session present. Not having a secure session termination only increases the attack surface for any of these attacks.

Session termination
After the user logs out the current session is terminated by external identity provider Auth0. It removes authentication token from local storage and single sign on cookie. Note that access tokens are still usable until expiry (JWTs). However, the JWTs expiration is set to 24 hours.

7. Test Session Timeout (OTG-SESS-007)

In this phase testers check that the application automatically logs out a user when that user has been idle for a certain amount of time, ensuring that it is not possible to “reuse” the same session and that no sensitive data remains stored in the browser cache.

Re-usable token after timeout

Kentico Cloud uses Auth0 provider which does not allow token to be re-used after timeout.

Server side session expiration

Kentico Cloud uses Auth0 provider which specifies the number of minutes after which the single sign on cookie expires to 3 days. Access token expiration is set to 24 hours.

8. Session Management Verification Requirements

Task	Result
Verify that the framework’s default session management control implementation is used by the application.	OK
Verify that sessions are invalidated when the user logs out.	OK
Verify that session timeout after a specified period of inactivity.	OK
Verify that session timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout).	OK
Verify that all pages that require authentication to access them have logout links.	OK
Verify that the session id is never disclosed other than in cookie headers; particularly in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies.	OK
Verify that the session id is changed on login to prevent session fixation.	OK
Verify that the session id is changed upon re-authentication.	OK

Verify that only session ids generated by the application framework are recognized as valid by the application.	OK
Verify that authenticated session tokens are sufficiently long and random to withstand session guessing attacks.	OK
Verify that authenticated session tokens using cookies have their path set to an appropriately restrictive value for that site. The domain cookie attribute restriction should not be set unless for a business requirement, such as single sign on.	OK
Verify that authenticated session tokens using cookies sent via HTTP, are protected by the use of "HttpOnly".	OK
Verify that authenticated session tokens using cookies are protected with the "secure" attribute and a strict transport security header (such as Strict-Transport-Security: max-age=60000; includeSubDomains) are present.	OK

II. Authentication testing

1. Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)

Testing for credentials transport means verifying that the user’s authentication data are transferred via an encrypted channel to avoid being intercepted by malicious users. The analysis focuses simply on trying to understand if the data travels unencrypted from the web browser to the server, or if the web application takes the appropriate security measures using a protocol like HTTPS. The HTTPS protocol is built on TLS/SSL to encrypt the data that is transmitted and to ensure that user is being sent towards the desired site.

SSL/TLS functionality

All credentials data in Kentico Cloud are transmitted over HTTPS.

2. Testing for Default Credentials (OTG-AUTHN-002)

Many web applications often make use of popular open source or commercial software that can be installed on servers with minimal configuration or customization by the server administrator. Moreover, a lot of hardware appliances (i.e. network routers and database servers) offer web-based configuration or administrative interfaces.

Default credentials

Kentico Cloud does not consists of any default credentials. The only one is strong system-generated password for invited user. The password is temporary and user is forced to change it on the first log on.

3. Testing for Weak Lock out mechanism (OTG-AUTHN-003)

Account lockout mechanisms are used to mitigate brute force password guessing attacks. Accounts are typically locked after 3 to 5 unsuccessful login attempts and can only be unlocked after predetermined periods of time, via a self-service unlock mechanism, or intervention by an administrator. Account lockout mechanisms require a balance between protecting accounts from unauthorized access and protecting users from being denied authorized access.

Lock out mechanism

Auth0 identity provider has implemented lock out policy. If a user enters their password incorrectly more than 10 times from a single IP address, they will be blocked from logging into that account from that IP address. More information can be found here - https://auth0.com/docs/connections/database/rate-limits

How will accounts be unlocked?

When the account is locked the user receives an email with a unlock link.

Has administrator recovery method in case his account gets locked?

Yes. He can use the unlock link sent by e-mail in case of account lockout.

4. Testing for Bypassing Authentication schema (OTG-AUTHN-004)

While most applications require authentication to gain access to private information or to execute tasks, not every authentication method is able to provide adequate security. Negligence, ignorance, or simple understatement of security threats often result in authentication schemes that can be bypassed by simply skipping the log in page and directly calling an internal page that is supposed to be accessed only after authentication has been performed.

Direct page request (forced browsing).
When accessing protected areas directly (unauthenticated) logon is forced. This means that no page was accessible for non-authorized user. Bearer tokens are checked on every request to the backend API (ensured globally for every request using Auth0 middleware).

Parameter modification.
In Kentico Cloud there are no parameters used to authenticate users. Kentico Cloud does not use pattern like "authenticated=yes" in any URL, hidden fields and cookies

Session ID prediction
The answer can be found in Testing for Bypassing Session Management Schema (OTG-SESS-001) section. Tokens are unforgeable without the adequate private key (protected by Auth0).

5. Test Remember Password functionality (OTG-AUTHN-005)

Browsers will sometimes ask a user if they wish to remember the password that they just entered. The browser will then store the password, and automatically enter it whenever the same authentication form is visited. This is a convenience for the user. Additionally some websites will offer custom "remember me" functionality to allow users to persist logins on a specific client system.

Look for passwords being stored in a cookie. Examine the cookies stored by the application. Verify that the credentials are not stored in clear text, but are hashed.
Passwords and persistent session are handled solely by Auth0 identity provider.

Examine the hashing mechanism: if it is a common, well-known algorithm, check for its strength; in homegrown hash functions, attempt several usernames to check whether the hash function is easily guessable.
Passwords are handled solely by Auth0 identity provider. The default hash algorithm used to secure passwords is PBKDF.

Verify that the credentials are only sent during the log in phase, and not sent together with every request to the application.
Credentials are only sent on login and handled by Auth0 identity provider.

6. Testing for Weak Password policy (OTG-AUTHN-007)

The most prevalent and most easily administered authentication mechanism is a static password. The password represents the keys to the kingdom, but is often subverted by users in the name of usability.

What characters are permitted and forbidden for use within a password? Is the user required to use characters from different character sets such as lower and uppercase letters, digits and special symbols?
No permitted or forbidden characters. Password requirements can be customized in settings (number of special chars, minimal length, regex).

When must a user change their password? After 90 days? After account lockout due to excessive log on attempts?
It depends on settings, both expiration period (days) and maximum invalid logon attempts can be configured.

Is the user prevented from using his username or other account information (such as first or last name) in the password?
No.

7. Testing for Weak Password change or Reset Functionalities (OTG-AUTHN-009)

The password change and reset function of an application is a self-service password change or reset mechanism for users. This self-service mechanism allows users to quickly change or reset their password without an administrator intervening. When passwords are changed they are typically changed within the application. When passwords are reset they are either rendered within the application or emailed to the user. This may indicate that the passwords are stored in plain text or in a decryptable format.

Can users, other than administrators, change or reset passwords for accounts other than their own?
No. User is able to reset only her own password.

The password change or reset process is vulnerable to CSRF.
No. Auth0 handles CSRF protection to avoid this attack.

What information is required to reset the password?
User needs to have an access to his e-mail box.

How are reset passwords communicated to the user?
A confirmation email is sent to the user e-mail.

Are reset passwords generated randomly?
Yes. It is handled by Auth0 identity provider.

Is the reset password functionality requesting confirmation before changing the password?
Yes. Confirmation email is sent to the user.

Is the old password requested to complete a password change?
Password change invokes reset password flow which does not request the old password to be specified.

8. Authentication Verification Requirements

Task	Result
Verify all pages and resources require authentication except those specifically intended to be public (Principle of complete mediation).	OK
Verify all password fields do not echo the user’s password when it is entered.	OK
Verify all authentication controls are enforced on the server side.	OK
Verify all authentication controls (including libraries that call external authentication services) have a centralized implementation.	OK
Verify all authentication controls fail securely to ensure attackers cannot log in.	OK
Verify password entry fields allow or encourage the use of passphrases, and do not prevent long passphrases or highly complex passwords being entered, and provide a sufficient minimum strength to protect against the use of commonly chosen passwords.	OK
Verify all account identity authentication functions (such as registration, update profile, forgot username, forgot password, disabled / lost token, helpdesk or IVR)that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.	OK
Verify users can safely change their credentials using a mechanism that is at least as resistant to attack as the primary authentication mechanism.	OK
Verify that all authentication decisions are logged. This should include requests with missing required information, needed for security investigations.	OK
Verify that account passwords are salted using a salt that is unique to that account (e.g., internal user ID, account creation) and use bcrypt, scrypt or PBKDF2 before storing the password.	OK
Verify that credentials, and all other identity information handled by the application(s), do not traverse unencrypted or weakly encrypted links.	OK
Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user.	OK
Verify that username enumeration is not possible via login, password reset, or forgot account functionality.	OK

Verify there are no default passwords in use for the application framework or any components used by the application (such as "admin/password").	OK
Verify that a resource governor is in place to protect against vertical (a single account tested against all possible passwords) and horizontal brute forcing all accounts tested with the same password e.g. "Password1").	OK
Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location (not in source code).	OK
Verify that forgot password and other recovery paths send a link including a time-limited activation token rather than the password itself. Additional authentication based on soft-tokens (e.g. SMS token, native mobile applications, etc.) can be required as well before the link is sent over.	OK
Verify that forgot password functionality does not lock or otherwise disable the account until after the user has successfully changed their password. This is to prevent valid users from being locked out.	OK
Verify that the system can be configured to disallow the use of a configurable number of previous passwords.	OK
Verify re-authentication, step up or adaptive authentication, SMS or other two factor authentication, or transaction signing is required before any application-specific sensitive operations are permitted as per the risk profile of the application.	OK

III. Input validation

1. Testing for Reflected Cross site scripting (OTG-INPVAL-001)

Reflected Cross-site Scripting (XSS) occur when an attacker injects browser executable code within a single HTTP response. The injected attack is not stored within the application itself; it is non-persistent and only impacts users who open a maliciously crafted link or third-party web page. The attack string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.

Review process

Reflected XSS vulnerability has been tested on all parts of Kentico Cloud product which renders some HTML code. Penetration testing followed these rules:

- Front-end review
    - React.JS framework
- Back-end review
    - All APIs return responses in JSON format with a correctly specified Content-Type header.

The review was strongly focused on all user input placed in:

- GET URL parameters (query string values)
- POST URL parameters
- Cookies
- HTTP request headers
- React.JS properties performing unsafe operations, e.g. dangerouslySetInnerHTML

We ensured that all user inputs have been properly encoded in appropriate context:

- HTML context
- Javascript context
- URL context

Tools used during XSS review:

- BurpSuite
- Netsparker

2. Testing for Stored Cross site scripting (OTG-INPVAL-002)

Stored Cross-site Scripting (XSS) is the most dangerous type of Cross Site Scripting. Web applications that allow users to store data are potentially exposed to this type of attack. This chapter illustrates examples of stored cross site scripting injection and related exploitation scenarios.

Review process
The same as Testing for Reflected Cross site scripting (OTG-INPVAL-001).

3. Testing for HTTP Verb Tampering (OTG-INPVAL-003)

The HTTP specification includes request methods other than the standard GET and POST requests. A standards compliant web server may respond to these alternative methods in ways not anticipated by developers. Although the common description is ‘verb’ tampering, the HTTP 1.1 standard refers to these request types as different HTTP ‘methods.’

Results
Kentico Cloud app responds to GET, POST, PUT and DELETE requests, providing user data in the URL query string or appended to the request respectively.

HTTP Verb	Used in Kentico Cloud app
GET	YES
POST	YES
PUT	YES
DELETE	YES
HEAD	NO
OPTIONS	NO
TRACE	NO
CONNECT	NO

4. Testing for HTTP Parameter pollution (OTG-INPVAL-004)

Supplying multiple HTTP parameters with the same name may cause an application to interpret values in unanticipated ways. By exploiting these effects, an attacker may be able to bypass input validation, trigger application errors or modify internal variables values. As HTTP Parameter Pollution (in short HPP) affects a building block of all web technologies, server and client side attacks exist.

Results

Kentico Cloud backend is based on the Asp.Net Core. FromQuery attribute instructs the framework to look for a name that matches that in the query-string and accepts only the first value.

Query string value	Assigned value
id=1	1
id=1&id=123	1
missing id parameter	nulll

5. Testing for SQL Injection (OTG-INPVAL-005)

An SQL injection attack consists of insertion or “injection” of either a partial or complete SQL query via the data input or transmitted from the client (browser) to the web application. A successful SQL injection attack 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 existing on the DBMS file system or write files into the file system, and, in some cases, issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

Results

Kentico Cloud avoids SQL injection by using Linq expressions in Entity framework. When you run a Linq query to fetch the data, any Linq "Where" statement will be packaged as a parameter query and sent to SQL server. If there is a need to create a direct SQL query then all user inputs are parametrized using parametrized query pattern.

6. Testing for LDAP Injection (OTG-INPVAL-006)

The Lightweight Directory Access Protocol (LDAP) is used to store information about users, hosts, and many other objects. LDAP injection is a server side attack, which could allow sensitive information about users and hosts represented in an LDAP structure to be disclosed, modified, or inserted. This is done by manipulating input parameters afterwards passed to internal search, add, and modify functions.

Results

Not Applicable.

7. Testing for ORM Injection (OTG-INPVAL-007)

ORM Injection is an attack using SQL Injection against an ORM generated data access object model. From the point of view of a tester, this attack is virtually identical to a SQL Injection attack. However, the injection vulnerability exists in code generated by the ORM tool.

Results

Not Applicable.

8. Testing for XML Injection (OTG-INPVAL-008)

XML Injection testing is when a tester tries to inject an XML doc to the application. If the XML parser fails to contextually validate data, then the test will yield a positive result.

Results

Not applicable.

- There is no handling of XML input and the system is not working with XML files on the backend.
- It wasn't possible to trick the server into processing XML input where JSON is normally expected.

9. Testing for SSI Injection (OTG-INPVAL-009)

Web servers usually give developers the ability to add small pieces of dynamic code inside static HTML pages, without having to deal with full-fledged server-side or client-side languages. This feature is incarnated by the Server-Side Includes (SSI). In SSI injection testing, we test if it is possible to inject into the application data that will be interpreted by SSI mechanisms. A successful exploitation of this vulnerability allows an attacker to inject code into HTML pages or even perform remote code execution.

Results

Not applicable.

10. Testing for XPath Injection (OTG-INPVAL-010)

XPath is a language that has been designed and developed primarily to address parts of an XML document. In XPath injection testing, we test if it is possible to inject XPath syntax into a request interpreted by the application, allowing an attacker to execute user-controlled XPath queries. When successfully exploited, this vulnerability may allow an attacker to bypass authentication mechanisms or access information without proper authorization.

Results

Not applicable.

11. Testing for IMAP/SMTP Injection (OTG-INPVAL-011)

This threat affects all applications that communicate with mail servers (IMAP/SMTP), generally webmail applications. The aim of this test is to verify the capacity to inject arbitrary IMAP/SMTP commands into the mail servers, due to input data not being properly sanitized.

Results

Not applicable.

- 3rd party mail provider SendGrid is used to send emails.

12. Testing for Code Injection (OTG-INPVAL-012)

This section describes how a tester can check if it is possible to enter code as input on a web page and have it executed by the web server.

Results

Not Applicable.

13. Testing for Command Injection (OTG-INPVAL-013)

This article describes how to test an application for OS command injection. The tester will try to inject an OS command through an HTTP request to the application.

Results

Not Applicable.

14. Testing for Buffer Overflow (OTG-INPVAL-014)

To find out more about buffer overflow vulnerabilities, please go to Buffer Overflow pages.

Results

Not Applicable.

15. Testing for Incubated Vulnerability (OTG-INPVAL-015)

Also often referred to as persistent attacks, incubated testing is a complex testing method that needs more than one data validation vulnerability to work. Incubated vulnerabilities are typically used to conduct “watering hole” attacks against users of legitimate web applications.

Result

Not applicable.

16. Testing for HTTP Splitting/Smuggling (OTG-INPVAL-016)

This section illustrates examples of attacks that leverage specific features of the HTTP protocol, either by exploiting weaknesses of the web application or peculiarities in the way different agents interpret HTTP messages.

Results

Not Applicable.

17. Malicious Input Handling Verification Requirements

Task	Result
Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows.	OK
Verify that all input validation failures result in input rejection.	OK
Verify that a character set, such as UTF-8, is specified for all sources of input.	OK
Verify that all input validation or encoding routines are performed and enforced on the server side.	OK
Verify that a single input validation control is used by the application for each type of data that is accepted.	OK

Verify that all input data is canonicalized for all downstream decoders or interpreters prior to validation.	OK
Verify that the runtime environment is not susceptible to SQL Injection, or that security controls prevent SQL Injection.	OK
Verify that the runtime environment is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection.	OK
Verify that the runtime environment is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection.	OK
Verify that the runtime environment is not susceptible to XML External Entity attacks or that security controls prevents XML External Entity attacks.	OK
Verify that the runtime environment is not susceptible to XML Injections or that security controls prevents XML Injections.	OK
Verify that all untrusted data that are output to HTML (including HTML elements, HTML attributes, JavaScript data values, CSS blocks, and URI attributes) are properly escaped for the applicable context.	OK
If the application framework allows automatic mass parameter assignment (also called automatic variable binding) from the inbound request to a model, verify that security sensitive fields such as “accountBalance”, “role” or “password” are protected from malicious automatic binding.	OK
Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, environment, etc.)	OK
Verify that for each type of output encoding/escaping performed by the application, there is a single security control for that type of output for the intended destination.	OK

IV. Authorization testing

1. Testing Directory traversal/file include (OTG-AUTHZ-001)

Many web applications use and manage files as part of their daily operation. Using input validation methods that have not been well designed or deployed, an aggressor could exploit the system in order to read or write files that are not intended to be accessible. In particular situations, it could be possible to execute arbitrary code or system commands.

Results

No Directory traversal of file inclusion was found.

2. Testing for Bypassing Authorization Schema (OTG-AUTHZ-002)

This kind of test focuses on verifying how the authorization schema has been implemented for each role or privilege to get access to reserved functions and resources.

Results

Is it possible to access that resource even if the user is not authenticated?

- No.

Is it possible to access that resource after the log-out?

- No.

Is it possible to access functions and resources that should be accessible to a user that holds a different role or privilege?

- No.

Is it possible to access administrative functions also if the tester is logged as a user with standard privileges?

- No.

Is it possible to use these administrative functions as a user with a different role and for whom that action should be denied?

- No

3. Testing for Privilege escalation (OTG-AUTHZ-003)

This section describes the issue of escalating privileges from one stage to another. During this phase, the tester should verify that it is not possible for a user to modify his or her privileges or roles inside the application in ways that could allow privilege escalation attacks.

Results

No privilege escalation has been found in these tested scenarios:

I. Invitations

Can lower privilege level invite user with higher privilege level to the project?

- No. Only project managers and subscription administrators can invite users to projects.

II. Assigning the role

Can user assign higher privilege level role to himself?

- No.

Can lower privilege level user assign user to the higher privilege level role?

- No. Only project managers and subscription administrators can invite users to projects.

III. Admin promotion

Can unauthorized user assign himself to the admin role?

- No.

Can lower privilege level user assign other user to the admin role?

- No. Only subscription administrators can assign users to the admin role.

4. Testing for Insecure Direct Object References (OTG-AUTHZ-004)

Insecure Direct Object References occur when an application provides direct access to objects based on user-supplied input. As a result of this vulnerability attackers can bypass authorization and access resources in the system directly, for example database records or files.

Results

No Insecure Direct Object Reference has been found.

5. Access Control Verification Requirements

Task	Result
Verify that users can only access secured functions or services for which they possess specific authorization.	OK
Verify that users can only access secured URLs for which they possess specific authorization.	OK
Verify that users can only access secured data files for which they possess specific authorization.	OK
Verify that direct object references are protected, such that only authorized objects or data are accessible to each user (for example, protect against direct object reference tampering).	OK
Verify that directory browsing is disabled unless deliberately desired.	OK
Verify that access controls fail securely.	OK
Verify that the same access control rules implied by the presentation layer are enforced on the server side for that user role, such that controls and parameters cannot be re-enabled or re-added from higher privilege users.	OK
Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	OK
Verify that all access controls are enforced on the server side.	OK

V. Client side testing

1. Testing for DOM based Cross Site Scripting (OTG-CLIENT-001)

DOM-based Cross-Site Scripting is the de-facto name for XSS bugs which are the result of active browser-side content on a page, typically JavaScript, obtaining user input and then doing something unsafe with it which leads to execution of injected code. This document only discusses JavaScript bugs which lead to XSS.

Results

See OTG-INPVAL-002.

2. Testing for JavaScript Execution (OTG-CLIENT-002)

A JavaScript Injection vulnerability is a subtype of Cross Site Scripting (XSS) that involves the ability to inject arbitrary JavaScript code that is executed by the application inside the victim’s browser. This vulnerability can have many consequences, like disclosure of a user’s session cookies that could be used to impersonate the victim, or, more generally, it can allow the attacker to modify the page content seen by the victims or the application behavior.

Results

See OTG-CLIENT-001, OTG-INPVAL-001/2.

3. Testing for HTML Injection (OTG-CLIENT-003)

HTML injection is a type of injection issue that occurs when a user is able to control an input point and is able to inject arbitrary HTML code into a vulnerable web page. This vulnerability can have many consequences, like disclosure of a user’s session cookies that could be used to impersonate the victim, or, more generally, it can allow the attacker to modify the page content seen by the victims.

Results

See OTG-CLIENT-001, OTG-INPVAL-001/2.

4. Testing for Client Side URL Redirect (OTG-CLIENT-004)

This section describes how to check for Client Side URL Redirection, also known as Open Redirection. It is an input validation flaw that exists when an application accepts a user controlled input which specifies a link that leads to an external URL that could be malicious. This kind of vulnerability could be used to accomplish a phishing attack or redirect a victim to an infection page.

Results
Static code analysis has been performed on the whole KC solution. No Client Side Redirection issues have been found.

5. Testing for CSS Injection (OTG-CLIENT-005)

A CSS Injection vulnerability involves the ability to inject arbitrary CSS code in the context of a trusted web site, and this will be rendered inside the victim's browser. The impact of such a vulnerability may vary on the basis of the supplied CSS payload: it could lead to Cross-Site Scripting in particular circumstances, to data exfiltration in the sense of extracting sensitive data or to UI modifications.

Results
No CSS injection issues have been found.

6. Testing for Client Side Resource Manipulation (OTG-CLIENT-006)

A ClientSide Resource Manipulation vulnerability is an input validation flaw that occurs when an application accepts a user controlled input which specifies the path of a resource (for example the source of an iframe, js, applet or the handler of an XMLHttpRequest). Specifically, such a vulnerability consists in the ability to control the URLs which link to some resources present in a web page. The impact may vary on the basis of the type of the element whose URL is controlled by the attacker, and it is usually adopted to conduct Cross-Site Scripting attacks.

Results

Static code analysis has been performed on the whole KC solution. No Client Side Resource Manipulation issues have been found.

7. Test Cross Origin Resource Sharing (OTG-CLIENT-007)

Cross Origin Resource Sharing or CORS is a mechanism that enables a web browser to perform "cross-domain" requests using the XMLHttpRequest. In the past, the XMLHttpRequest L1 API only allowed requests to be sent within the same origin as it was restricted by the same origin policy.

Results

CORS is generally enabled on all endpoints for all origins. However, since every request has to be authenticated using an unforgeable token supplied in a request header, the current state is believed to be secure.

8. Testing for Cross Site Flashing (OTG-CLIENT-008)

ActionScript is the language, based on ECMAScript, used by Flash applications when dealing with interactive needs. There are three versions of the ActionScript language. ActionScript 1.0 and ActionScript 2.0 are very similar with ActionScript 2.0 being an extension of ActionScript 1.0. ActionScript 3.0, introduced with Flash Player 9, is a rewrite of the language to support object orientated design.

Results

Not Applicable.

9. Testing for Clickjacking (OTG-CLIENT-009)

"Clickjacking" (which is a subset of the "UI redressing") is a malicious technique that consists of deceiving a web user into interacting (in most cases by clicking) with something different to what the user believes they are interacting with. This type of attack, that can be used alone or in combination with other attacks, could potentially send unauthorized commands or reveal confidential information while the victim is interacting on seemingly harmless web pages.

Results

All HTTP responses in Kentico Cloud app define the X-Frame-Options header with the value "DENY". This configuration prevents the site page from being included in an iFrame.

10. Testing WebSockets (OTG-CLIENT-010)

Traditionally the HTTP protocol only allows one request/response per TCP connection. Asynchronous JavaScript and XML (AJAX) allows clients to send and receive data asynchronously (in the background without a page refresh) to the server, however, AJAX requires the client to initiate the requests and wait for the server responses (half-duplex).

Results

Not applicable.