DataArt

[redacted] Financial Services

Web Application Penetration Testing Report

Prepared by:

[redacted]@dataart.com)

Reviewed by:

Justin Surman (justin.surman@finstrides.com)

Limitations on Disclosure and Use of this Document

This report was prepared by DataArt for the exclusive benefit of Financial Strides and is proprietary information. Unauthorized use or reproduction of this document is prohibited. The Non-Disclosure Agreement (NDA) in effect between DataArt and Financial Strides governs the disclosure of this report to all other parties, including product vendors or suppliers.

This report contains information about potential vulnerabilities of the [redacted] web solution and methods for exploiting them. DataArt recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. DataArt has retained and secured a copy of the report for customer reference. All other copies of the report have been delivered to Financial Strides.

By providing this report to Financial Strides, DataArt does not constitute any form of representation, warranty, or guarantee that the systems are 100% secure from every form of attack. While DataArt's methodology includes both automated and manual testing to identify and attempt exploitation of the most common security issues, testing was limited to an agreed upon time frame.

Document Details

Title	**Web Application (Grey box) Penetration Testing Report**
Project Name	[redacted]
Project Resources	[redacted] – Security Consultant [redacted] Justin Surman - PM & Reviewers
Project Duration	03/12/2020 – 03/18/2020

Document History

Version	Date	Author	Comments
1.0	03/18/2020	[redacted]	Initial Version
1.1	03/18/2020	[redacted]	Internal Peer Review
1.2	03/18/2020	Justin Surman	Review and Corrections
1.3	03/19/2020	[redacted]	Final Version
1.4	03/26/2020	[redacted]	Re-test findings to validate issue remediations by the [redacted] Development Team
1.5	03/26/2020	[redacted] Justin Surman	Final Version
1.6	03/31/2020	[redacted]	Re-test of the M2 issue
1.7	03/31/2020	[redacted]	Re-test of the M2 issue

Contents

Executive Summary

Financial Strides engaged DataArt to perform a penetration testing of the [redacted] web application. The primary goal of this web application (Grey box) penetration testing project was to identify any potential areas of concern associated with the application in its current state and determine the extent to which the system may be breached by an attacker possessing a particular skill and motivation. The assessment was performed in accordance with the "best-in-class" practices as defined by ISECOM's Open Source Security Testing Methodology Manual (OSSTMM), Open Web Application Security Project (OWASP) and Penetration Test Guidance for PCI DSS Standard.

DataArt conducted the penetration testing during the period of March 12 – 18, 2020. All testing activities were performed on the staging environment provided by the customer and completely isolated from the production data. While performing the testing activities, DataArt emulated an external attacker without prior knowledge of the environment. To test the user-authenticated area and privilege escalation vulnerabilities, the customer supplied DataArt credentials for several registered user and admin accounts.

The scope of the assessment included the following:

- https://app-staging[redacted]com/ - (Consumer Facing Web App - public and authenticated areas);
- https://api-staging[redacted]com/ - (Django, Admin facing WebApp and [redacted] web API);

Notes:

- It should be noted, that the Dashboard - Panel "Account Profile" functionality offered by the Consumer Facing Web App was not available during the penetration test and was excluded from the scope of the current assessment.
- The staging web application environment provided by [redacted] for the application penetration testing utilized partner stub & sandbox integrated environments only (Plaid / [redacted]).

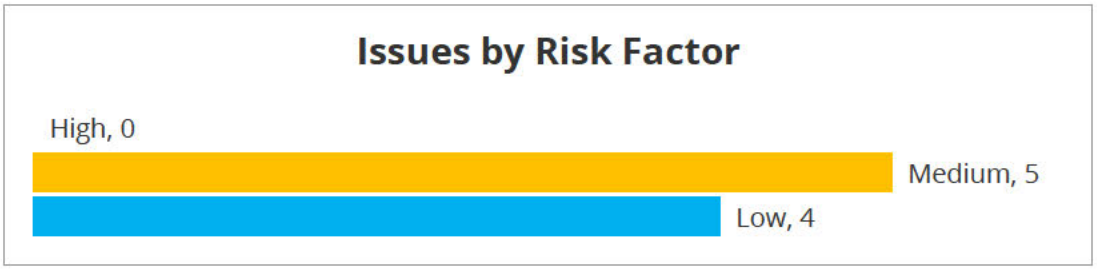
During the course of this assessment, DataArt did not identify any critical vulnerabilities that could lead to full compromise of the system. However, DataArt did find several medium and low severity issues, which should be addressed by [redacted]

As of 03/31/2020, the [redacted] development team has fixed all of the discovered medium vulnerabilities. Further detailed information can be found in the "Issues Remediation" section of this report.

DataArt strongly recommends [redacted] to remediate all medium severity issues detected to mitigate against the possible risk of a sensitive data compromise. The remediation of the low severity findings is not so urgent due to the low probability of their successful exploitation. Nonetheless, it should be noted that the existence of these known issues could decrease the overall security posture of the system.

This report summarizes what DataArt believes are the most important issues to address in the application. The chart below outlines a number of issues identified that are grouped by risk factors. Note the risk ratings

were given to help assist in prioritizing remediation efforts. True risk can only be calculated by an in-depth understanding of business processes and data, as well as the likelihood of exploitation.



The table below summarizes the findings for OWASP Top 10 list for web application vulnerabilities. OWASP Top 10 represents the list of the most critical web application security flaws, which are accompanied by OWASP security experts from around the world. The list provides a powerful awareness document for web application security and is utilized within many security standards:

Category	Discovered
A1: Injection	NO
A2: Broken Authentication	YES
A3: Sensitive Data Exposure	NO
A4: XML External Entities (XEE)	NO
A5: Broken Access Control	NO
A6: Security Misconfiguration	YES
A7: Cross Site Scripting (XSS)	NO
A8: Insecure Deserialization	NO
A9: Using Components with Known Vulnerabilities	NO
A10: Insufficient Logging & Monitoring	NO

DataArt can re-verify the [redacted] remediated issues found during this penetration test within 60 days of this report delivery (**until May 16th, 2020**). DataArt can also arrange to include into this re-scan missing functionality ("Account Profile") that was not available at the time of this current assessment.

Issues Remediation

DataArt re-verified all previously found issues on March 26, 2020. The tested application was accessible by the same URLs used during the initial testing.

As of **03/26/2020**, the following issues were successfully remediated:

- **M1. Enumeration of registered emails**
- **M3. Sensitive cookies without the "Secure" flag**
- **M4. The password reset link is reusable**
- **M5. The application is vulnerable to brute-force attacks**
- **L2. Cross-domain policy misconfiguration**

A subsequent re-test was performed on March 31,2020. As of **03/31/2020** the following issue was successfully remediated:

- **M2. Persistent cookie with sensitive information**

The other issues **(L1, L3, L4)** are still actual.

Assessment Methodology

DataArt based the findings and recommendations, outlined in this report, on application vulnerability scans and manual penetration testing performed against the application.

Automated Application Scan

DataArt used several commercial tools to survey the targeted environment and identify potential vulnerabilities. The automated scanning software identifies application-level vulnerabilities. The scope of testing includes but not limited by the following:

- Parameter Injection
- SQL Injection
- Cross-Site Scripting
- Directory Traversal
- Parameter Overflow
- Buffer Overflow
- Parameter Addition
- Path Manipulation
- Character Encoding
- Site Search
- SSL Strength
- Sensitive Developer Comments
- Web Server/Web Package Identification
- Permissions Assessment
- Brute Force Authentication attacks

Manual Application Testing

Using the information produced by the automated testing software, DataArt also employed manual testing techniques to identify and attempt exploiting additional vulnerabilities in the targeted application, and to eliminate false positives produced by the automated scanning process. The assessment was conducted in accordance with the best-in-class practices as defined by such methodologies as ISECOM's Open Source Security Testing Methodology Manual (OSSTMM) and the Open Web Application Security Project (OWASP).

DataArt performed the following actions as part of this testing:

- Gathered information about the application
- Mapped application content and analyzed it
- Observed types and placement of security controls
- Reviewed web page HTML source code for possible vulnerabilities

- Tested application authentication, session management and access controls
- Tested application for client data validation issues
- Tested application for input-based vulnerabilities
- Tested application for business logic flaws
- Checked for application server vulnerabilities

The test focused on possible vulnerabilities in the application logic, looking for issues including but not limited to:

- SQL and command injection
- Cross-site scripting
- Cross-site request forgery
- Authentication and authorization implementation defects
- Access control issues and privilege elevation
- Session management/hijacking
- File input/output implementation defects
- Parameter overflow and handling
- HTTP/URL manipulation
- Application logic defects
- Improper web server configuration
- Concurrency issues
- Information leakage
- SSL and transport layer weaknesses
- Application-level denial-of-service

Criteria for Risk Ratings

The table below outlines the general rules for assigning risk ratings to identified vulnerabilities:

Risk Rating	Description
HIGH	These issues identify conditions that could directly result in the compromise or unauthorized access of a network, system, application or sensitive information. Examples of High-Risk issues include remote execution of commands, known buffer overflows; unauthorized access and disclosure of sensitive information.

MEDIUM	These issues identify conditions that do not immediately or directly result in the compromise or unauthorized access of a network, system, application of information, but do provide a capability or information that could, in combination with other capabilities or information, result in the compromise or unauthorized access of a network, application or information. Examples of Medium Risk issues include directory browsing, partial access to files on the system; disclosure of security mechanisms and unauthorized use of services.
LOW	These issues identify conditions that do not immediately or directly result in compromise of a network, system, application or information, but do provide information that could be used in combination with other information to gain insight into how to compromise or gain unauthorized access to a network, system, application or information.
REMEDIATED	The issue has been fixed since the previous round of penetration testing

Assessment Findings

Summary

The table below outlines summary of findings identified during the penetration testing:

Finding Description	Status
Medium Risk Findings	
M1. Enumeration of registered emails	~~MEDIUM~~ REMEDIATED
M2. Persistent cookie with sensitive information	~~MEDIUM~~ REMEDIATED
M3. Sensitive cookies without the "Secure" flag	~~MEDIUM~~ REMEDIATED
M4. The password reset link is reusable	~~MEDIUM~~ REMEDIATED
M5. The application is vulnerable to brute-force attacks	~~MEDIUM~~ REMEDIATED
Low Risk Findings	
L1. Strict-Transport-Security header is not used	LOW
L2. Cross-domain policy misconfiguration	~~LOW~~ REMEDIATED
L3. Auto-complete feature is not disabled for password fields	LOW
L4. The application server supports TLS cipher suites without forward security	LOW

High Risk Findings

No **high**-severity issues were found in the application.

Medium Risk Findings

Five **medium**-severity issues were found in the application, as described below.

M1. Enumeration of registered emails

Risk Rating: ~~MEDIUM~~ REMEDIATED

Remediation Efforts: MEDIUM

Summary

DataArt identified that the application notified the user about the existence of the entered email address. Such behavior provides the ability to a potential malefactor to enumerate all emails registered within the system. The collected information could be used for further attacks (for instance for spreading phishing emails or other social engineering attacks).

Affected Functionality

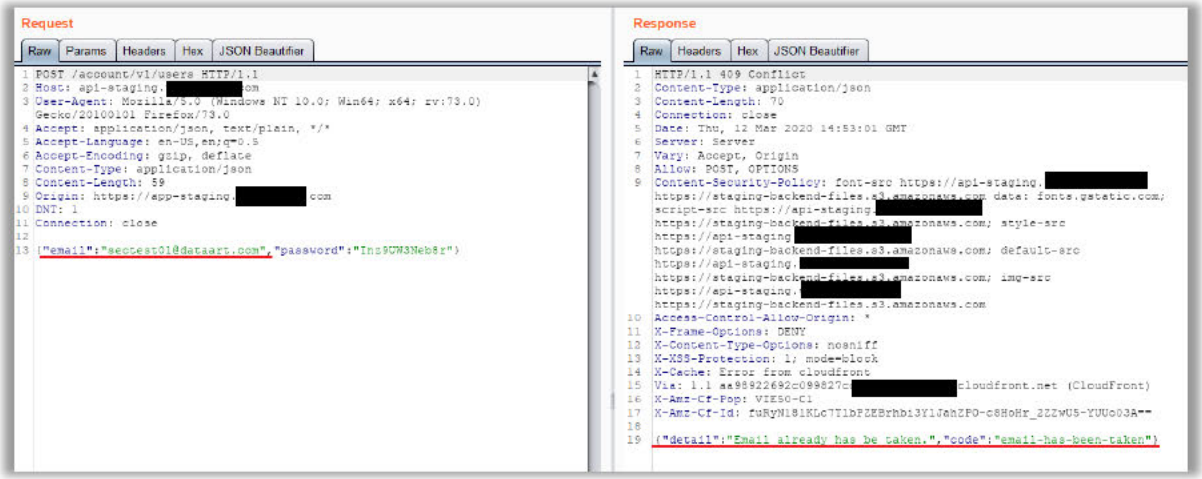
The following request required a user email as an input:

- Registration functionality:
  - **POST** https://api-staging[redacted]com/account/v1/users

```
{"email":"email","password":"password"}
```

Proof of Concept

The image below shows the server response for the signup functionality in the case where the entered email exists in the system:



The image below shows the server response for the signup functionality in the case where the entered email does not exist in the system:

Recommendations

In order to avoid automated enumeration of valid e-mails, DataArt suggests using one of the following techniques:

1. Do not show an error message providing information about the existence or non-existence of the entered email. In both cases, the application should return a generic message that the required information has been sent to the entered email address.
2. Add a CAPTCHA challenge after the series of failed attempts to prevent automated email enumeration.

Remediation

As of **03/26/2020**, the issue was successfully remediated. Currently, a unique random invite code delivered by the application administrator is required for the account creation process:

M2. Persistent cookie with sensitive information

Risk Rating: ~~MEDIUM~~ REMEDIATED

Remediation Efforts: LOW

Summary

DataArt identified that the application used persistent cookies for session tracking in the Django Administration Portal. The cookies were stored to the hard disk and survived a browser restart. As the cookie contained authentication information, this can allow a local attacker to access the application without knowledge of the password.

Moreover, it was noticed that the user's session cookies lifetime was two weeks, this is quite a big time range to allow for potential malicious activities.

Affected Functionality

The following session token of Django Administration portal was persistent:

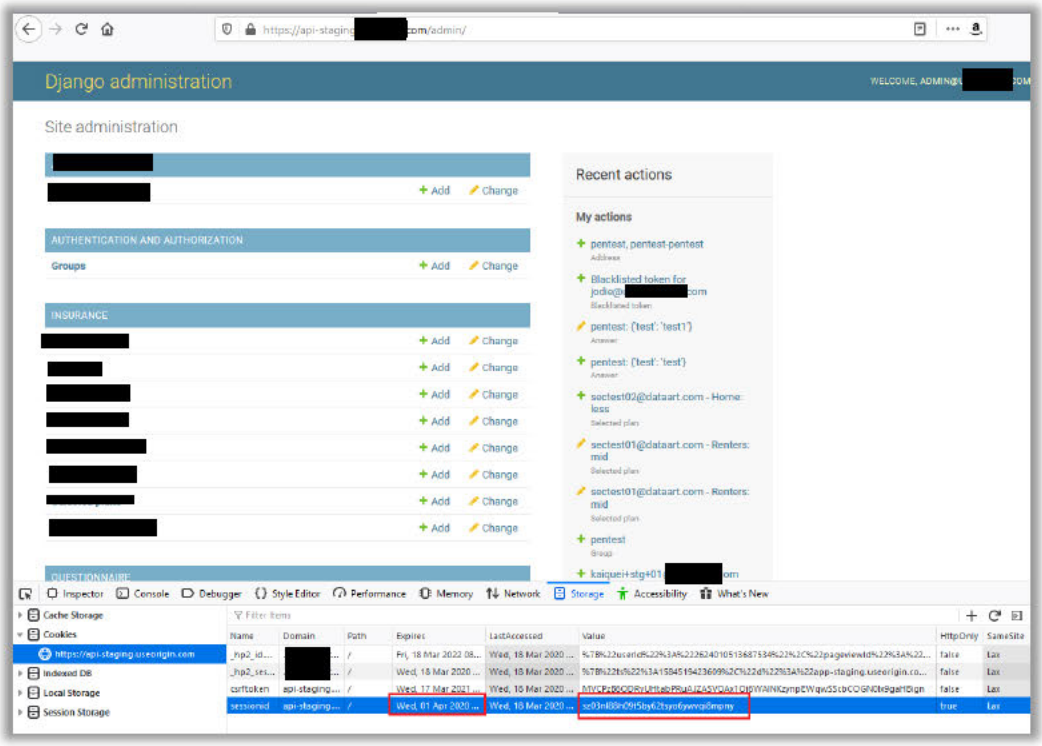
- sessionid

Proof of Concept

Log in to the application as an administrator user and inspect server responses in a web proxy. The method "**POST** /admin/login/?next=/admin/" sets a cookie named "sessionid" that is used by the Django Administration Portal.

Please note that the cookie contains an expiration date that makes it persistent.

Close the browser, restart and open URL [https://api-staging[redacted]com/admin/](https://api-staging[redacted]com/admin/). Notice that the user is still authorized in the application:

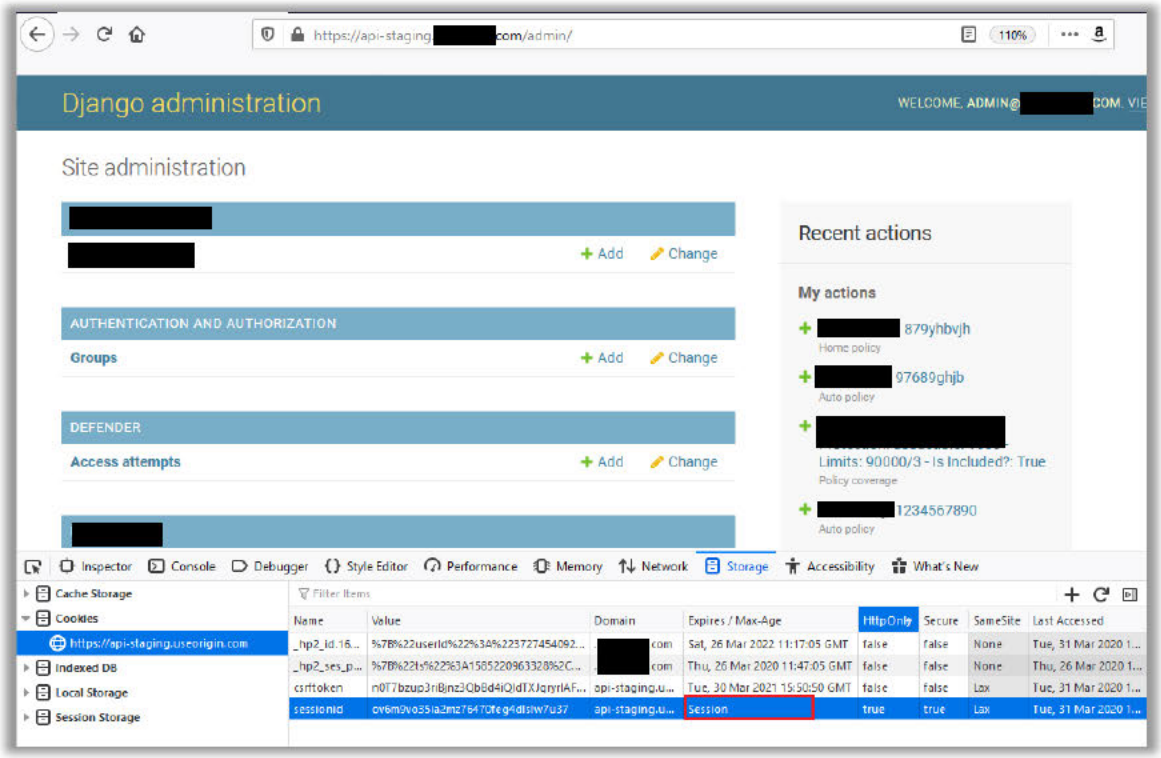


Recommendations

DataArt recommends to avoid using persistent cookies for storing authorization tokens or other sensitive information. Session cookies should be used instead, which are never saved to disk and are automatically cleared when a user closes the browser.

Remediation

As of **03/31/2020**, the issue was successfully remediated. The Django Admin application uses session cookies with sensitive information:



M3. Sensitive cookies without the "Secure" flag

Risk Rating: ~~MEDIUM~~ REMEDIATED

Remediation Efforts: LOW

Summary

DataArt has determined that the Django Administration portal issued an authentication cookie without the "Secure" flag. The purpose of the "Secure" flag is to prevent cookies from being observed by unauthorized parties due to the transmission of the cookie in cleartext. To accomplish this goal, browsers that support the secure flag will only send cookies with the Secure flag when the request is going to an HTTPS page.

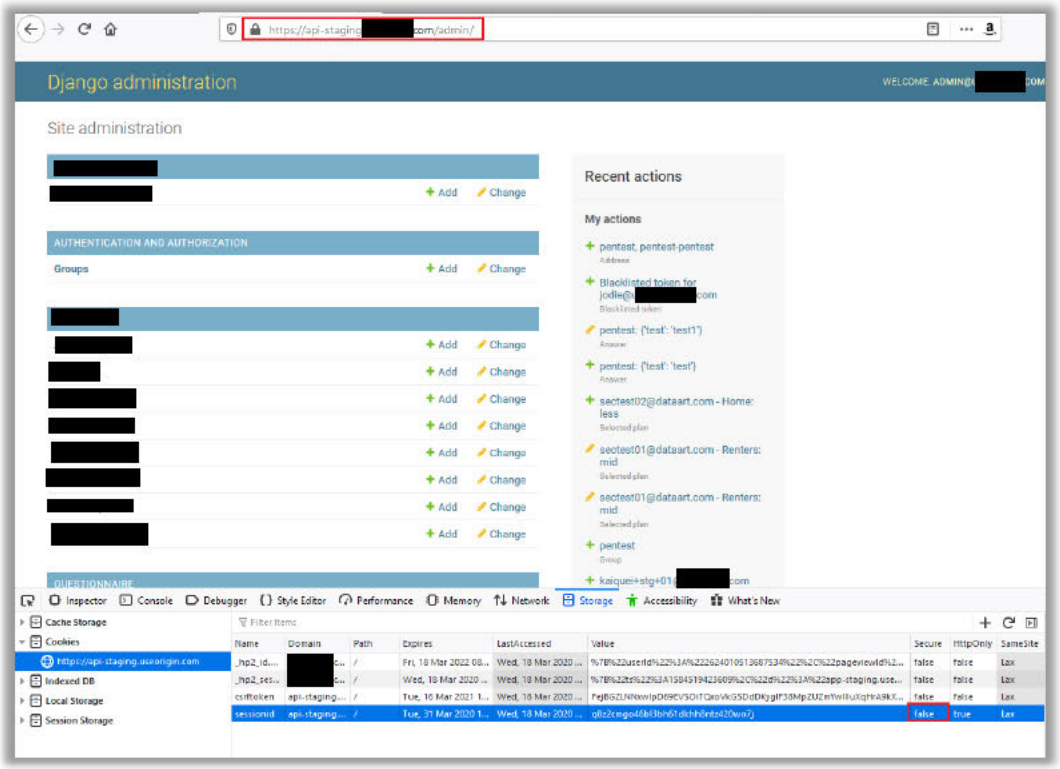
Affected Functionality

The following cookie was not marked by the "Secure" flag:

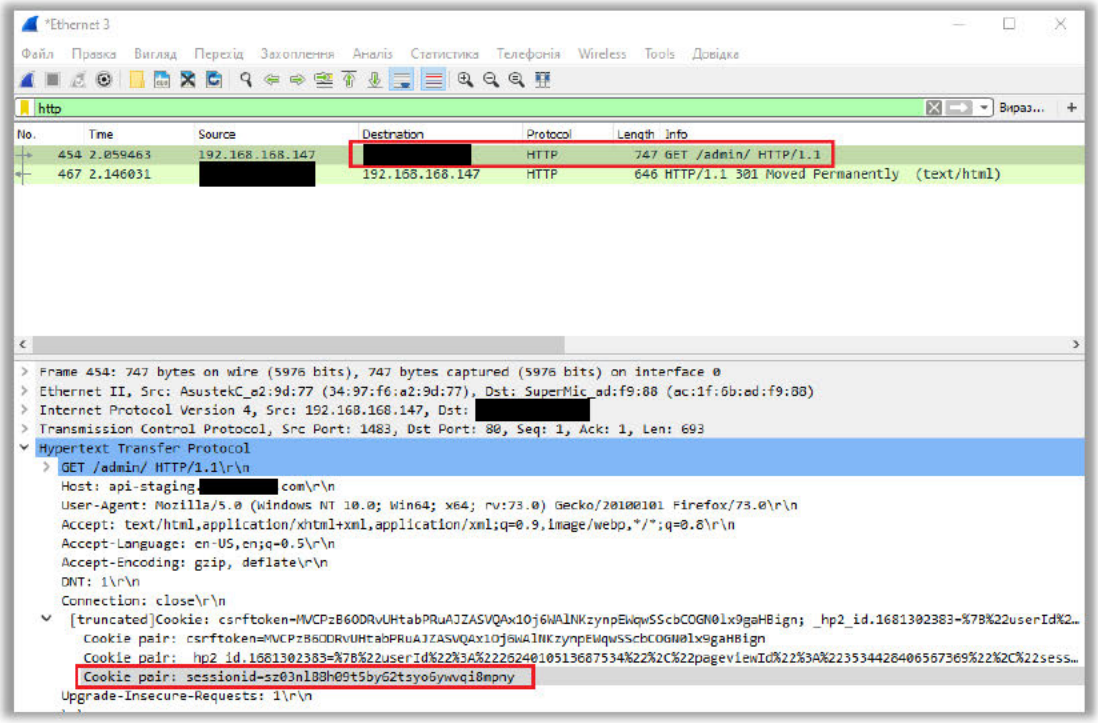
- sessionid

Proof of Concept

The following screenshot demonstrates the authentication cookie without the “Secure” flag:



Launch the Wireshark network analyzer and start capturing the traffic to and from TCP port 80. In the browser open the following URL http://api-staging[redacted]com/admin/ with the insecure “http://” schema and inspect the captured traffic in Wireshark to find unencrypted cookies. That demonstrates that the cookie can be intercepted by sniffing the network traffic.

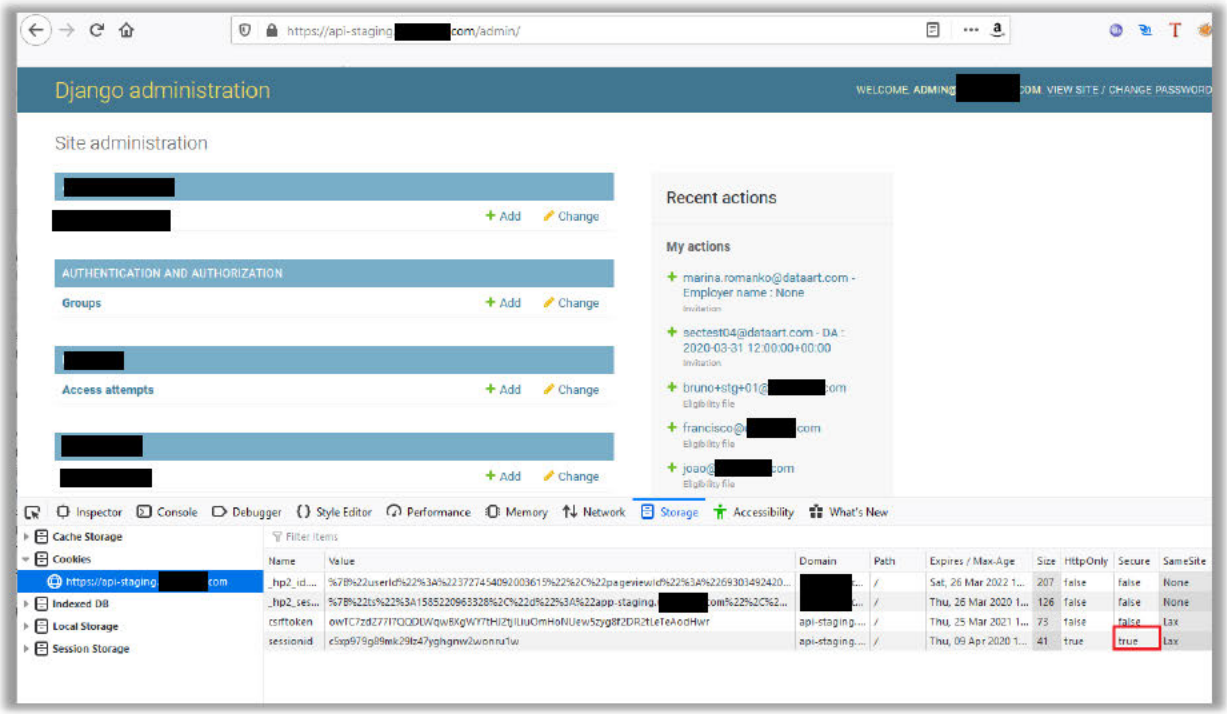


Recommendations

It's recommended to set on the "Secure" flag for any authentication and session cookies. Using such an approach, the browser will not send a cookie with the secure flag set over an unencrypted HTTP request.

Remediation

As of **26/03/2020**, the issue was successfully remediated. DataArt confirms that the "Secure" flag was set for the session cookie:



M4. The password reset link is reusable

Risk Rating: ~~MEDIUM~~ REMEDIATED

Remediation Efforts: MEDIUM

Summary

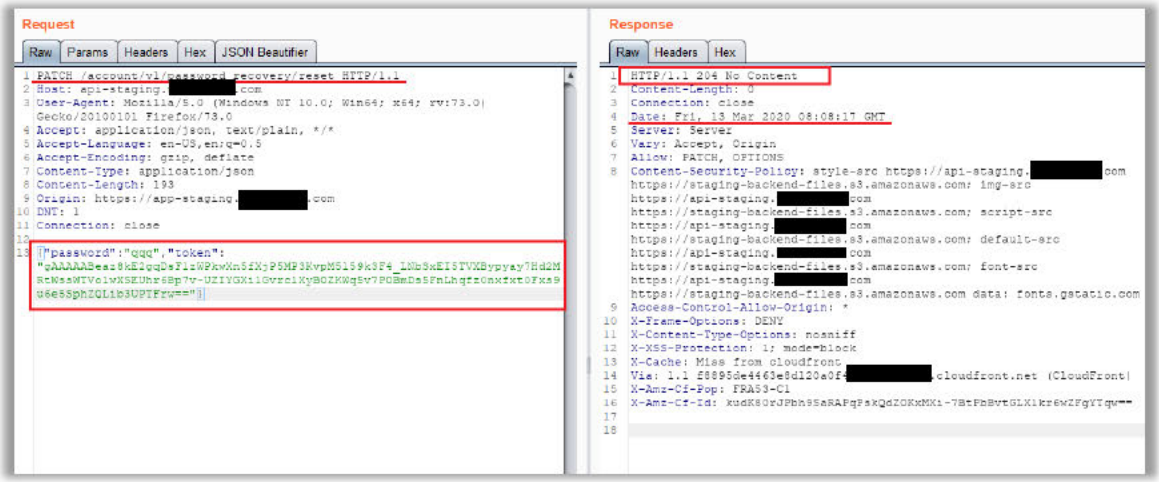
DataArt found that a password reset link could be used multiple times. If a user's email account is compromised, the token used to reset the password would be valid even if the user already reset their password.

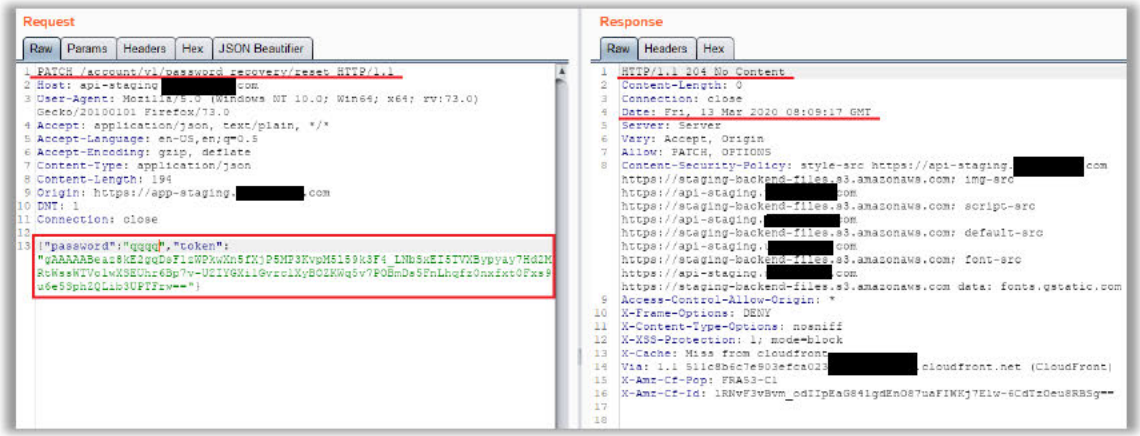
Affected Functionality

The issue affected the forgot password functionality.

Proof of Concept

The screenshots below demonstrate that a reset password link can be used several times:



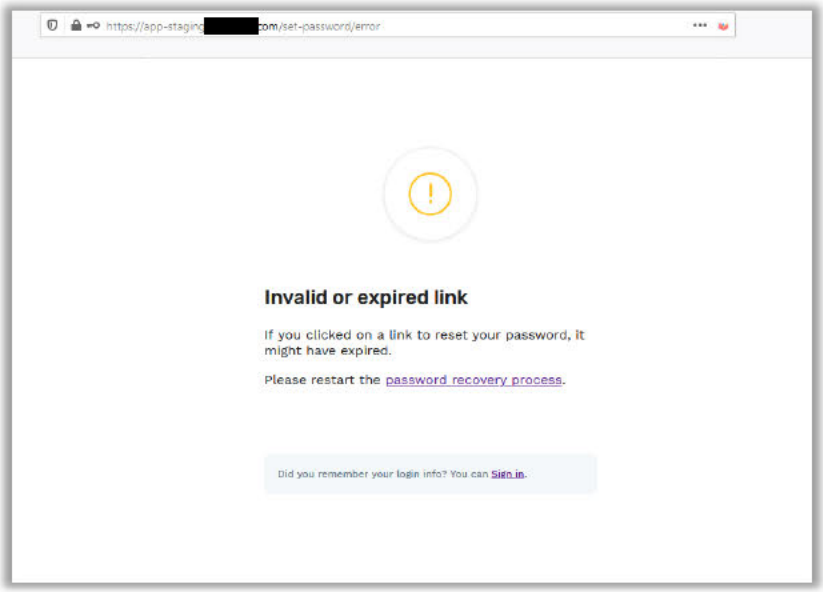


Recommendations

The reset password link should be single-used. Once a user's password has been reset, the randomly-generated token should no longer be valid.

Remediation

As of **03/26/2020**, the issue was successfully remediated. It is not possible to use a reset password link several times:



M5. The application is vulnerable to brute-force attacks

Risk Rating: ~~MEDIUM~~ REMEDIATED

Remediation Efforts: MEDIUM

Summary

While testing authentication functionality, DataArt observed that the Django Administration portal did not utilize any account lockout policy upon failed login attempts. DataArt performed more than 20 failed login attempts for one of the test accounts without receiving any lock messages. As far as DataArt could determine, no account lockout mechanism was in place.

The absence of an account lockout ability could give an attacker an infinite number of attempts to enter guesses of the current password to achieve a valid variant (known as "brute-force" attack). There are many scripts and tools that automate this type of attack available on the Internet that can be used against a web-based application.

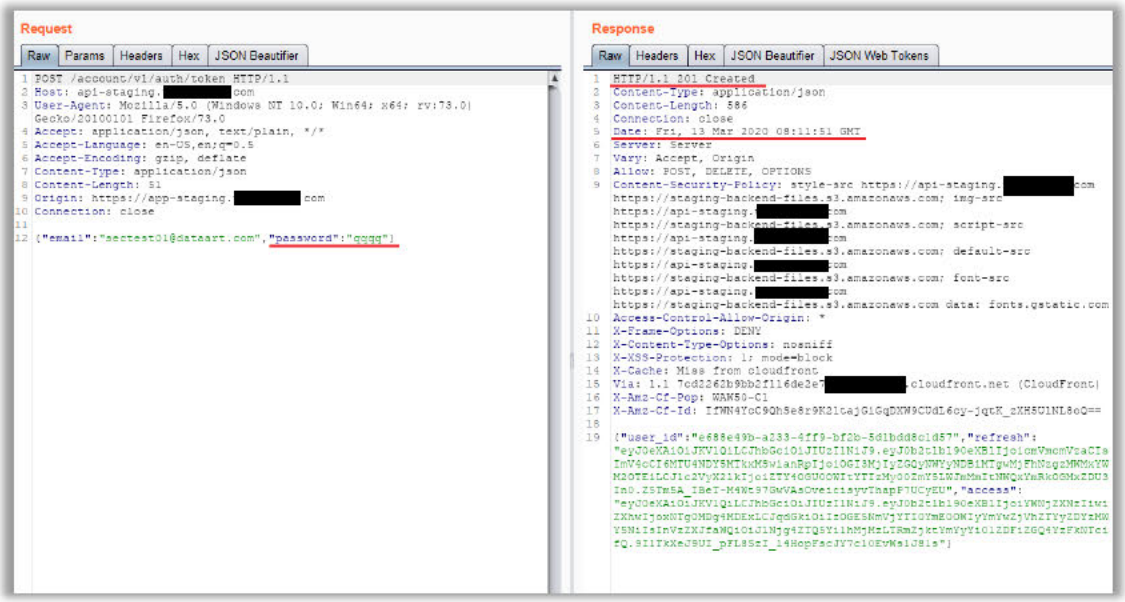
Affected Functionality

The issue affected the login functionality of the Django Administration portal:

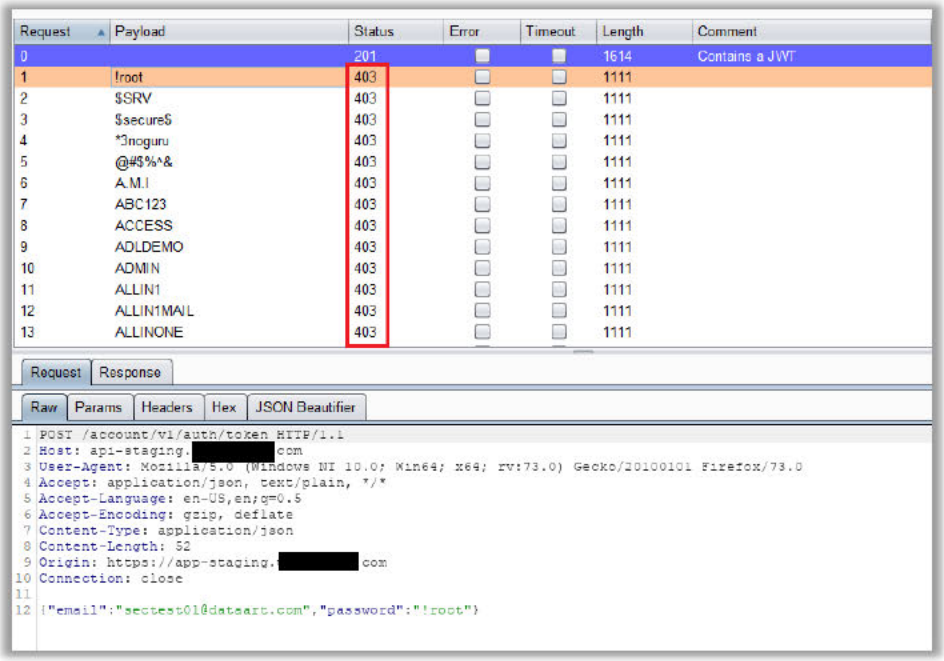
- **POST** https://api-staging[redacted]com/admin/login/?next=/admin/
- **POST** https://api-staging[redacted]com/account/v1/auth/token

Proof of Concept

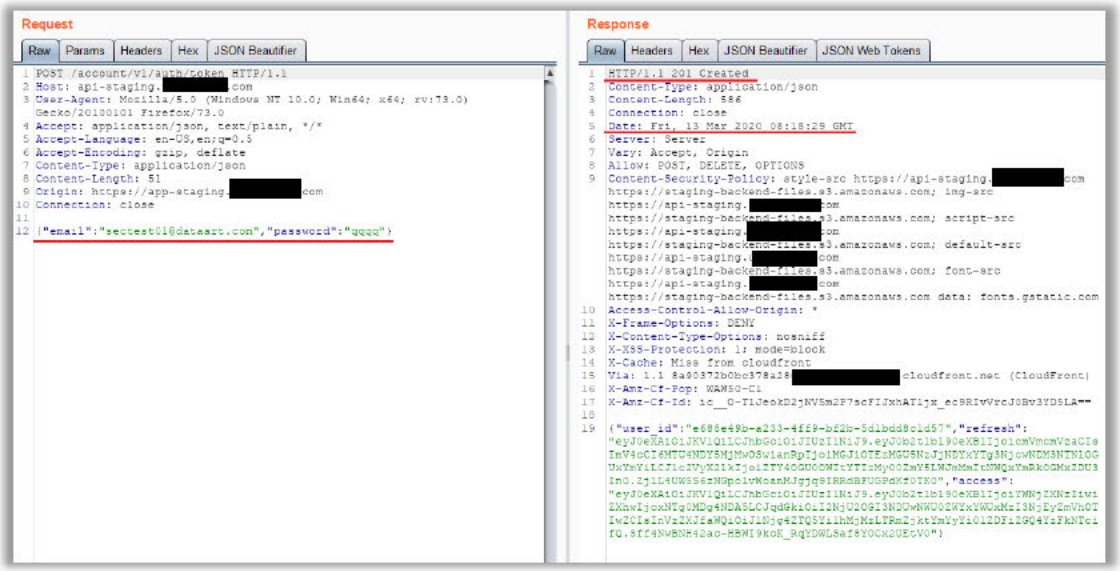
1. A user successfully logins to the application:



2. Logout the user and then try to login with the same login but wrong password multiple times:



3. After that, the user can still successfully log in to the application:



Recommendations

DataArt recommends employing a password lockout mechanism that temporarily locks an account if more than a preset number of unsuccessful login attempts are made. This approach significantly slows down attackers, while allowing the accounts to be open for legitimate users.

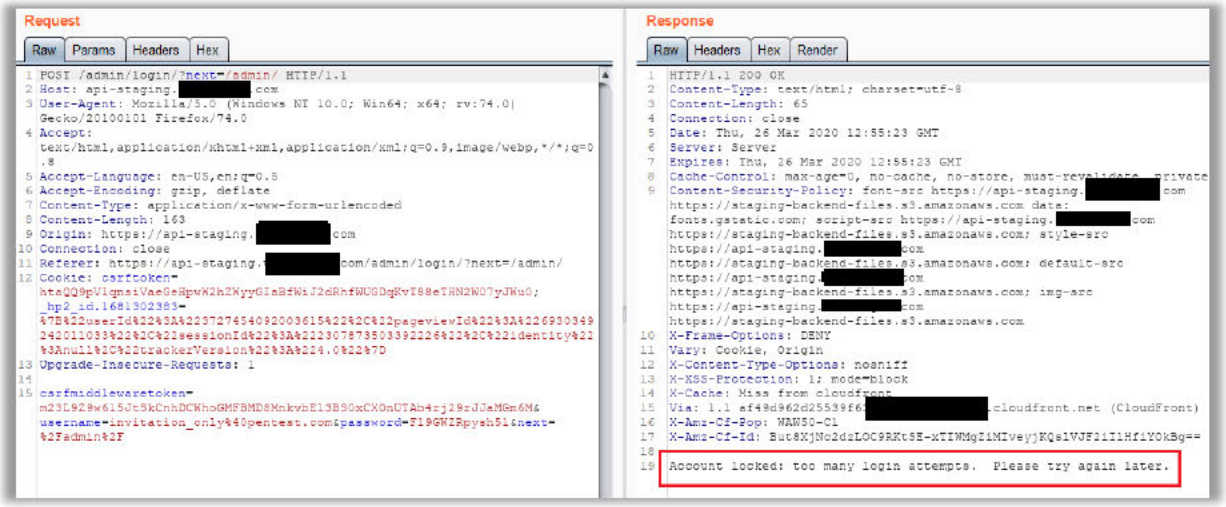
The most secure approach for the implementation of an account locking function assumes notifying the blocked user about the blocking only via a third-party channel (for instance via email or mobile phone). In this way, the malicious user trying to guess the valid password will not be able to know that the account is locked and all his further attempts will be unsuccessful.

Another approach is utilizing CAPTCHA services, which should force the user to input additional information provided in a human only understandable format.

Remediation

As of **03/26/2020**, the issue was successfully remediated. A user account was blocked after 5 incorrect login attempts to the application:

Also, in the Django Admin as well:



Low Risk Findings

Four **low**-severity issues were found in the application, as described below.

L1. Strict-Transport-Security header is not used

Risk Rating: LOW

Remediation Efforts: LOW

Summary

DataArt noticed that all applications did not utilize the "*Strict-Transport-Security*" header for encrypted communication. The aforementioned header forces browsers to use only an encrypted channel for communication with the server even in case the potential malefactor tries to downgrade the communication to an unsafe HTTP connection.

Without a Strict Transport Security policy, the application may be vulnerable against several attacks:

- If the web application mixes the usage of HTTP and HTTPS, an attacker can manipulate pages in the unsecured area of the application or change redirection targets in a manner that the switch to the secured page is not performed or done in a manner, that the attacker remains between client and server.
- If there is no HTTP server, an attacker in the same network could simulate a HTTP server and trick the user to click on a prepared URL by using a social engineering attack

Affected Functionality

The issue affected all responses from the following applications:

- https://app-staging[redacted]com
- https://api-staging[redacted]com

Proof of Concept

The example of the server response is shown below:

Recommendations

The application should instruct web browsers to only access the application using encrypted HTTPS channel. For that, HTTP Strict Transport Security (HSTS) should be enabled by adding a response header with the name 'Strict-Transport-Security' and the value 'max-age=expireTime', where expireTime is the time in seconds that browsers should remember that the site should only be accessed using HTTPS.

To apply the policy to all subdomains, the 'includeSubDomains' flag could be also utilized. As an additional security measure the domain should be submitted to an HSTS preload service.

The detailed information can be found via the following links below:

- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.md
- https://hstspreload.org/?domain=api-staging.[redacted].com
- https://hstspreload.org/?domain=app-staging.[redacted].com

Due to the fact that the affected applications utilize Amazon CloudFront as the CDN system for delivering static content, there is no direct way to set the required server header. However, as a workaround solution, it is possible to introduce an intermediate Edge Lambda function which puts the appropriate headers onto CloudFront HTTP responses before they're sent to the clients, see reference links below. (**Note:** however, such a configuration introduces an extra point of failure and could affect performance and availability of the site):

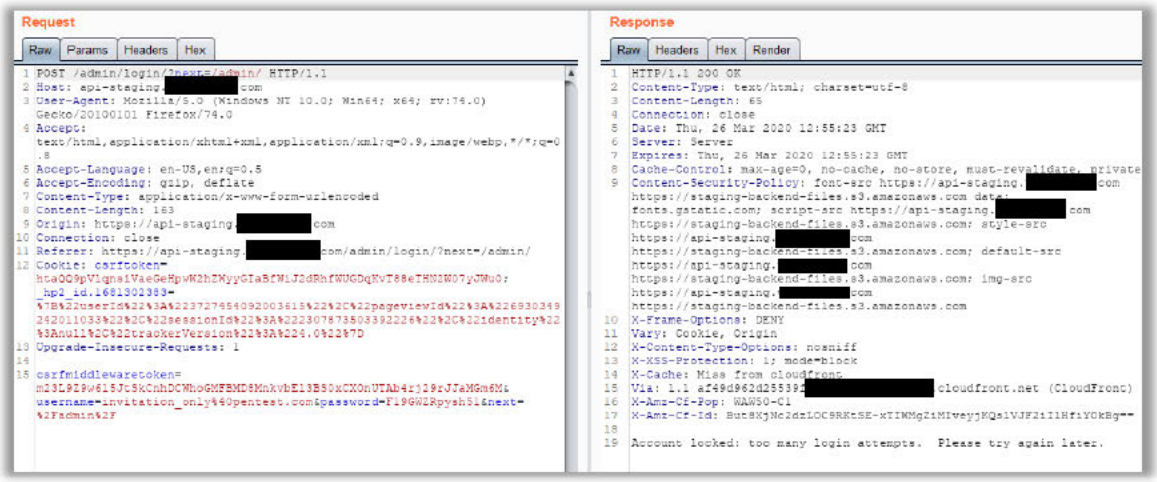
- https://medium.com/@netscylla/adding-security-headers-to-s3-websites-2002f243aa8f
- https://johnlouros.com/blog/setup-security-headers-s3-host-website
- https://adamj.eu/tech/2019/04/15/scoring-a+-for-security-headers-on-my-cloudfront-hosted-static-website/

As a possible way of remediation for the Django CMS used by one of the applications (https://api-staging.[redacted].com/admin), it is recommended to set appropriate security parameters within Django`s middleware as it is described in the article below:

- https://docs.djangoproject.com/en/3.0/ref/middleware/#http-strict-transport-security

Remediation

As of 03/26/2020, the issue **was not remediated**. Applications still do not utilize the "*Strict-Transport-Security*" header for encrypted communication:



L2. Cross-domain policy misconfiguration

Risk Rating: ~~LOW~~ REMEDIATED

Remediation Efforts: LOW

Summary

DataArt found that the application servers provided the ability to execute cross-domain requests from any domain to all the server's resources using HTML5 cross-[redacted] resource sharing (CORS). Trusting arbitrary [redacted] effectively disables the same-[redacted] policy, allowing two-way interaction by third-party web sites. Unless the response consists only of unprotected public content, this policy is likely to present a security risk.

If another domain is allowed by the policy, then that domain can potentially attack users of the application. If a user is logged in to the application and visits a domain allowed by the policy, then any malicious content running on that domain can potentially retrieve content from the application, and sometimes carry out actions within the security context of the logged-in user.

Even if an allowed domain is not overtly malicious in itself, security vulnerabilities within that domain could potentially be leveraged by an attacker to exploit the trust relationship and attack the application that allows access.

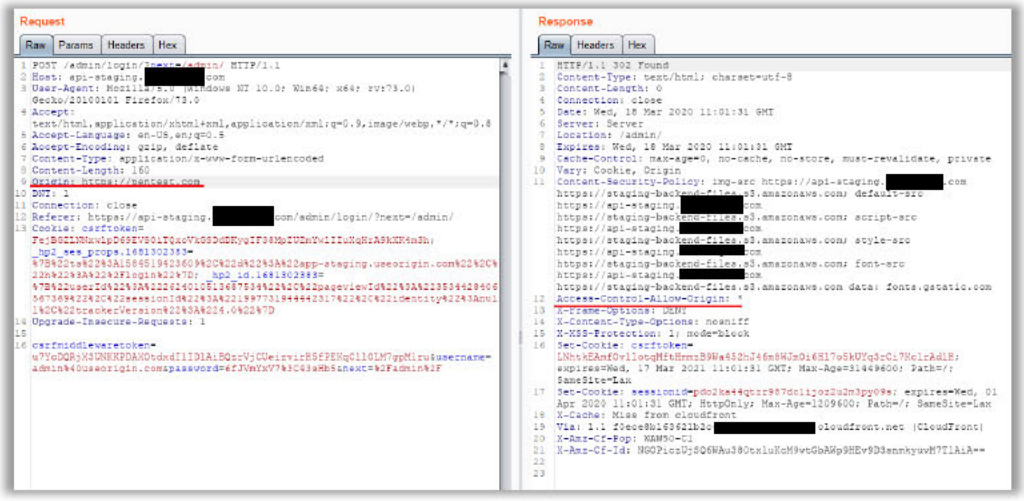
Affected Functionality

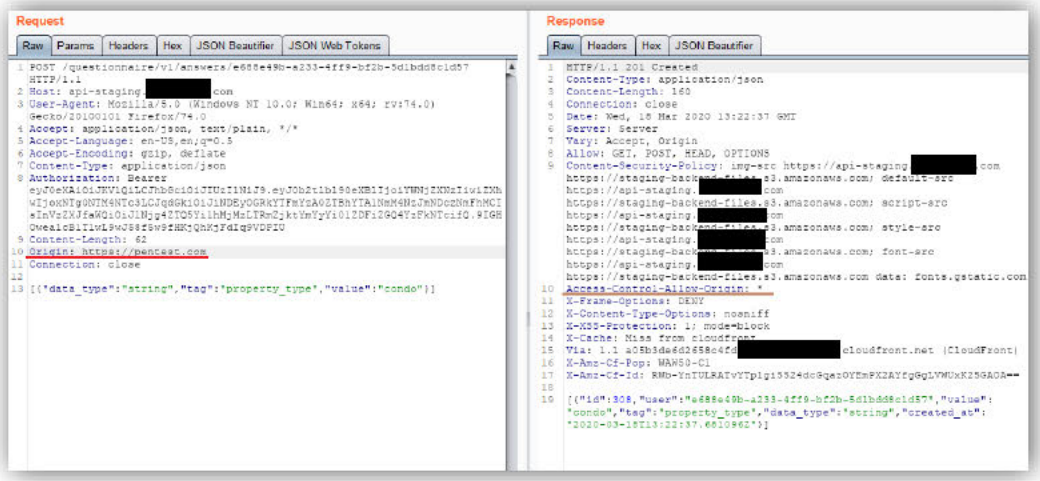
The issue affected all responses from the following applications:

- https://app-staging[redacted]com
- https://api-staging[redacted]com

Proof of Concept

The images below show the examples of the server's responses allowing cross-domain requests from any domain ("Access-Control-Allow-[redacted] *"):





Recommendations

DataArt strongly recommends setting the scope of allowed domains and permissions to be as restrictive as possible. In the case CORS functionality is not used at all, the server must prohibit it.

It has to be noted that the Gunicorn web server does not provide the ability to setup cross-[redacted] access to the resources. It is recommended to implement appropriate settings at the proxy server level located in front the Gunicorn server as outlined in the official documentation referenced below:

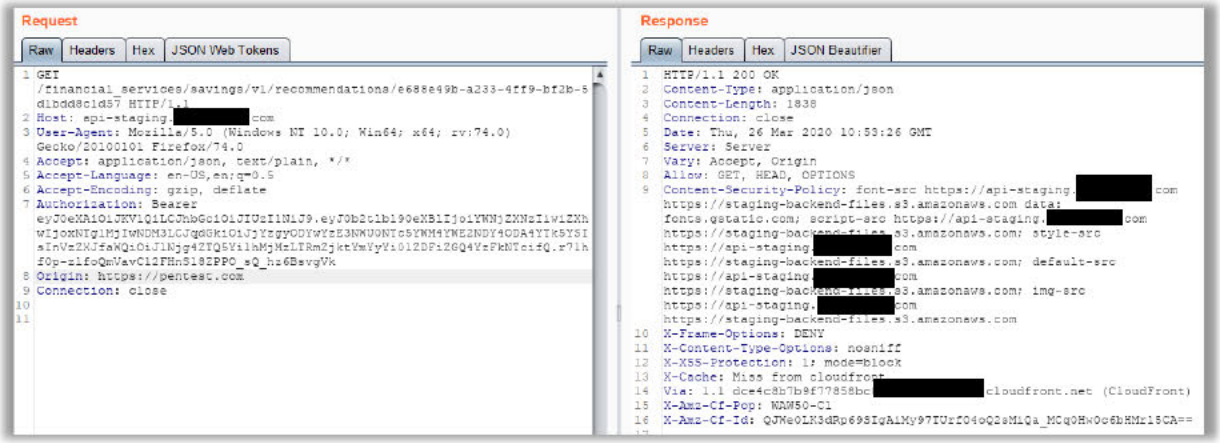
- https://docs.gunicorn.org/en/stable/deploy.html

In the case of the Gunicorn server utilized by the Django CMS it is possible to remediate the issue by setting the appropriate Django security middleware:

- https://docs.djangoproject.com/en/3.0/ref/middleware/#referrer-policy

Remediation

As of **03/26/2020**, the issue was successfully remediated. DataArt confirms, that now the application prohibits cross-domain interaction at all:



L3. Auto-complete feature is not disabled for password fields

Risk Rating: LOW

Remediation Efforts: LOW

Summary

Modern browsers offer users the ability to manage their multitude of credentials by storing them insecurely on their computers. This client-side feature could potentially compromise a user's account on the application in the event of the compromise of a client's workstation.

It was possible to remediate this issue previously by appending this additional parameter "*autocomplete='off'*" to the opening form tag or an individual form element of the password field to indicate to the browser that it should not offer to store that information. However, nowadays, almost all modern browsers ignore this attribute. The best approach for blocking passwords saving on the browser side is using the "*autocomplete='new-password'*" attribute, however, the attribute is not handled by old versions of some browsers (Firefox below version 38, Google Chrome below 34, and Internet Explorer below version 11).

As an alternative way of preventing password autofilling, could be the utilization of a 3rd party client-side library like the "jquery.disableAutoFill" plugin.

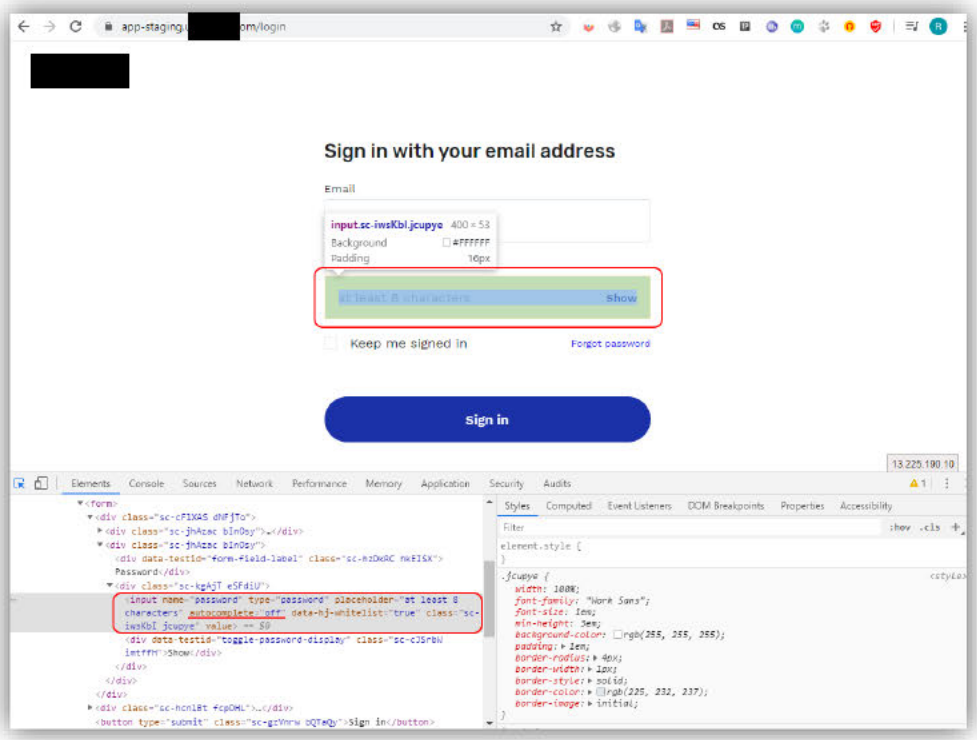
Affected Functionality

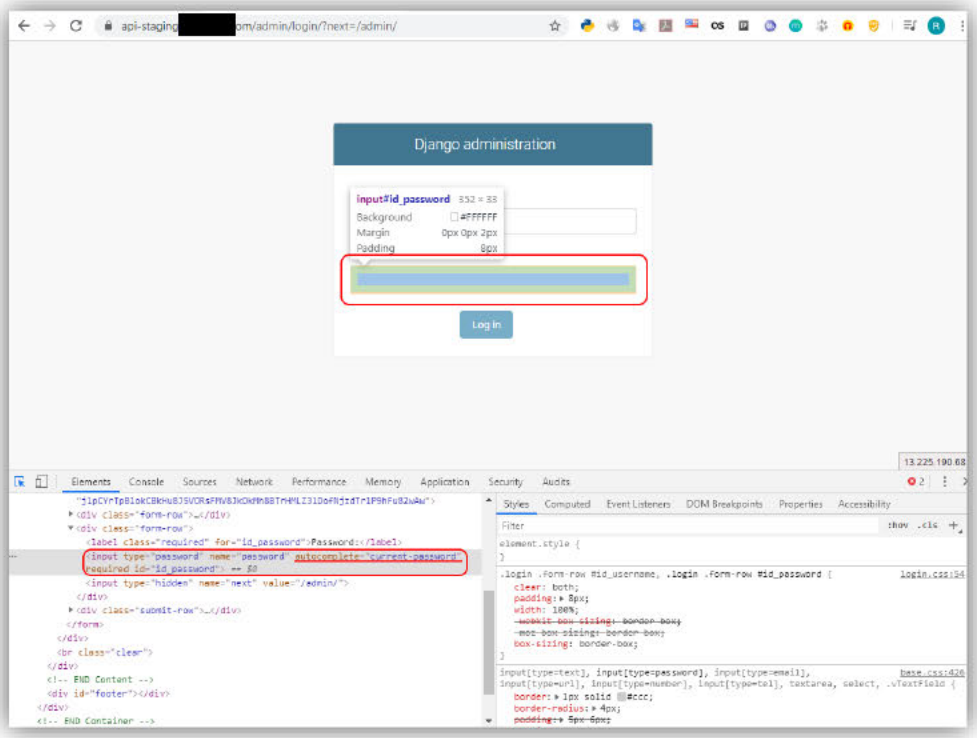
The issue affected the following functionality:

- https://app-staging[redacted].com/login (autocomplete='off')
- https://api-staging[redacted].com/admin/login/?next=/admin/ (autocomplete='current-password')

Proof of Concept

The image below shows an example of enabled autocomplete for the application login forms:





Recommendations

The password autocomplete should always be disabled, especially in sensitive applications, since an attacker, if able to access the browser cache, could easily obtain the password in clear-text.

The detailed information about autocomplete implementation in all modern browsers can be found in these articles below:

- https://developer.mozilla.org/en-US/docs/Web/Security/Securing_your_site/Turning_off_form_autocompletion
- https://developer.mozilla.org/en-US/docs/Web/Security/Securing_your_site/Turning_off_form_autocompletion#Tools_for_disabling_autocompletion
- https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/autocomplete#Browser_compatibility

Remediation

As of 03/26/2020, the issue **was not remediated**. The password autocomplete is still enabled for the password fields.

L4. The application server supports TLS cipher suites without forward security

Risk Rating: LOW

Remediation Efforts: LOW

Summary

DataArt noticed that the application servers supported TLS cipher modes that used RSA encryption for key exchange. Though the encryption algorithm was considered secure, it did not provide Forward Secrecy (FS). It also could potentially lead to a private key compromise in the presence of such vulnerabilities as Bleichenbacher's Oracle or ROBOT or other yet unknown vulnerabilities.

Affected Functionality

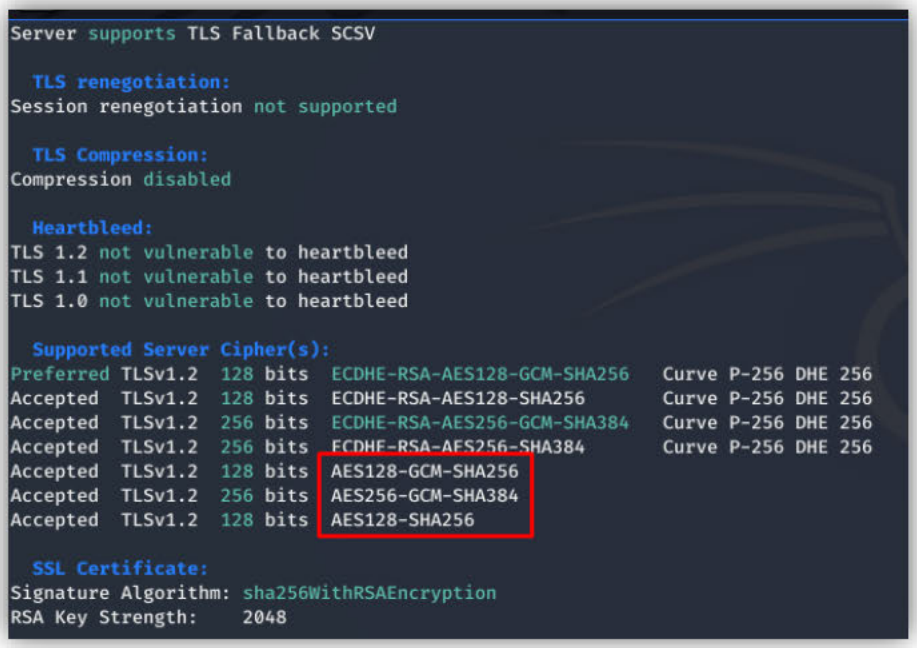
The issue affected all cipher suites except the ECDHE ones allowed by the application servers:

- app-staging [redacted] com (443/TCP);
- api-staging [redacted] com (443/TCP);

  AES128-GCM-SHA256
  AES256-GCM-SHA384
  AES128-SHA256

Proof of Concept

The images below show all cipher suites allowed by the application servers. The weak ones are marked red:



Recommendations

It is strongly recommended to utilize only cipher suites allowing perfect forward secrecy features. Such an approach should minimize possible security risks connected with decryption of previously sniffed encrypted traffic in case the private key has been compromised.

Detailed information about perfect forward secrecy could be found in the article below:

- https://www.keycdn.com/blog/perfect-forward-secrecy

Since the affected application utilizes Amazon CloudFront, the issue could be fixed by applying up-to-date TLS security policies for the corresponding AWS resources:

- CloudFront -TLSv1.2 2018 is recommended in AWS documentation. The Cloud Conformity page explains where the corresponding TLS policy settings can be found. Please note that there's no way to deselect particular TLS cipher suites for CloudFront, as these sets of cipher suites are predefined by AWS-supplied policies;
- AWS S3 - it's necessary to verify that Secure Transfer is enforced for all buckets (including those which serve as storage for CloudFront distributions).

Remediation

As of 03/26/2020, the issue **was not remediated**. The application servers still accept the mentioned cipher suites.

Conclusion

DataArt completed the penetration testing of the [redacted] web application. This testing was based on the technologies and known threats as of the date of this document. All the security issues discovered during that exercise were analyzed and described in this report.

Please note that as technologies and risks change over time, the vulnerabilities associated with the operation of systems described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities, will also change.