Security guide

Use this guide to reduce risk and mitigate threats to Java Agent security.

Threats

Understand and address security threats.

Operating Systems

Secure your operating systems.

Connections

Secure network connections.

Access

Remove non-essential access and features, update patches, and manage cookies.

Keys and Secrets

Manage keys and secrets.

Audit Trails

Audit events in your deployment.

ForgeRock® Identity Platform serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com. The ForgeRock® Common REST API works across the platform to provide common ways to access web resources and collections of resources.

Threats

The following sections describe some possible threats to Java Agent, which you can mitigate by following the instructions in this guide.

Out-of-date software

Prevent the exploitation of security vulnerabilities by using up-to-date versions of the agent and third-party software.

Review and follow the ForgeRock Security Advisories. To receive email notifications for new security advisories, log in to Backstage, and click the **Subscribe** button for agent security advisories. Follow similar lists from all of your vendors.

Cached pages in browsers and web proxies

When browsers and web proxies cache pages that are accessed by a user, the cache can include sensitive information. Caching pages in browsers and web proxies increases risk of unwanted disclosure, especially in shared browsing environments.

Similarly, when web server responses are cached, sensitive information can be accessed by attackers. Caching web server responses is a common method to improve loading times and reduce server load.

To manage caching, set Custom Response Header Map so that HTTP responses generated by the agent include the Cache-Control HTTP header. This header tells browsers and web proxies whether they can be cached. For example, clients can set the following value clear existing cache responses, force the cache to revalidate with the server, and prevent new responses from being cached:

```
org.forgerock.agents.response.header.map[Cache-Control]=max-age=0, no-store
```

In your decision to set this property, consider the impact on the performance of customer applications. Setting this property can reduce performance because browser pages are not cached.

For more information about caching, see HTTP caching.

Reconnaissance

The initial phase of an attack sequence is often reconnaissance. Limit the amount of information available to attackers during reconnaissance, as follows:

- Avoid using words that identify Java Agent in error messages.

When AM is not available, the related error message contains the agent profile name. Consider this in your choice of agent profile name.

- Configure Agent Debug Level to use the lowest level of logging necessary. For example, consider logging at the `ERROR` or `WARNING` level, instead of `TRACE` or `MESSAGE`.
- In custom login redirects, calls to the custom login URL include the parameter Login Reason Parameter Name to indicate why authentication is required, such as `NO_TOKEN` and `TOKEN_EXPIRED`. To reduce the risk of leaking useful information, use alternative strings for the authentication reason, by configuring Login Reason Value Map.

Cross-site scripting

WARNING

Cross-site request forgery attacks (CSRF or XSRF) can be a cause of serious vulnerabilities in web applications. It is the responsibility of the protected application to implement countermeasures against such attacks, because Java Agent cannot provide generic protection against CSRF. ForgeRock recommends following the latest guidance from the OWASP CSRF Prevention Cheat Sheet.

When POST data preservation is enabled, captured POST data that is replayed appears to come from the same origin as the protected application, not from the site that originated the request. Therefore, CSRF defenses that rely solely on checking the origin of requests, such as SameSite cookies or Origin headers, are not reliable. ForgeRock strongly recommends using token-based mitigations against CSRF, and relying on other measures only as a defense in depth, in accordance with OWASP guidance.

Configure the following properties to protect against cross-site scripting attacks:

- Enable Composite Advice Encoding
- XSS Redirect URI Map
- XSS Code Element List

POST data preservation

By default, POST data is stored in the in-memory cache. Consider the following points if you configure POST Data Preservation in Files or Cache to store POST data in the file system:

- Payloads from unauthenticated users are stored in the agent file system. If your threat evaluation does not accept this risk, store the data in the cache, or set POST Data Preservation in Files or Cache to `false`.

- Restrict access to the POST Data Preservation File Directory, to mitigate the risk of permissive access or leakage of personally identifiable information (PII).
- Limit the amount of stored POST data to mitigate the risk of DoS attacks, by configuring POST Data Preservation Storage Size or Max Entries in POST Data Preservation Storage.
- Remove expired POST data as soon as possible by configuring the POST Data Preservation Directory Sweep Interval.
- Identify threats in POST data before it is deleted, by making sure that Intrusion Detection Systems inspect the data within the time specified by POST Data Preservation Directory Sweep Interval.

For information about configuration properties, see POST data preservation.

Compromised passwords

Use secure passwords for server administration.

For information about how to create the agent password, see Preinstallation tasks. The encrypted password is stored in the AgentPassword.properties file.

TIP

Although the agent accepts any password length and content, you are strongly encouraged to generate secure passwords. This can be achieved in various ways, for example using a password manager or by using the command line tool agentadmin --key.

Misconfiguration

Misconfiguration can arise from bad or mistaken configuration decisions, and from poor change management. Depending on the configuration error, features can stop working in obvious or subtle ways, and potentially introduce security vulnerabilities.

For example, if a configuration change prevents the server from making HTTPS connections, many applications can no longer connect, and the problem is detected immediately. However, if a configuration change allows insecure TLS protocol versions or cipher suites for HTTPS connections, some applications negotiate insecure TLS, but appear to continue to work properly.

- Access policy is not correctly enforced.

  Incorrect parameters for secure connections and incorrect Access Control Instructions (ACI) can lead to overly permissive access to data, and potentially to a security breach.
- The server fails to restart.

Although failure to start a server is not directly a threat to security, it can affect service availability.

To guard against bad configuration decisions, implement good change management:

- For all enabled features, document why they are enabled and what your configuration choices mean. This implies a review of configuration settings, including default settings that you accept.
- Validate configuration decisions with thorough testing.
- Maintain a record of your configurations and the changes applied.

  For example, use a filtered audit log. Use version control software for any configuration scripts and to record changes to configuration files.
- Maintain a record of external changes to the system, such as changes to operating system configuration, and updates to software, such as the JVM that introduces security changes.

Unauthorized access

Data theft can occur when access policies are too permissive, and when the credentials to gain access are too easily cracked. It can also occur when the data is not protected, when administrative roles are too permissive, and when administrative credentials are poorly managed.

Poor risk management

Threats can arise when plans fail to account for outside risks. To mitigate risk, develop appropriate answers to at least the following questions:

- What happens when a server or an entire data center becomes unavailable?
- How do you remedy a serious security issue in the service, either in the Java Agent software or the connected systems?
- How do you validate mitigation plans and remedial actions?
- How do client applications work when the Java Agent offline?

  If client applications require always-on services, how do your operations ensure high availability, even when a server goes offline?

For critical services, test expected operation and disaster recovery operation.

Denial of service

To prevent memory exhaustion DOS attacks, configure Maximum Decompression Size to limit the maximum size to which a compressed JWT can be decompressed. This property

reduces the risk of a decompressed JWT consuming too much available memory.

Log overflow attacks

To prevent log overflow attacks, add a custom `agent-logback.xml` with a `DuplicateMessageFilter`. This filter detects duplicate messages, and after the specified number of repetitions, drops repeated messages. For more information, see Limiting repetitive log messages.

Operating systems

When you deploy Java Agent, familiarize yourself with the recommendations for the host operating systems that you use. For comprehensive information about securing operating systems, see the HTTPS://downloads.cisecurity.org/#/[CIS Benchmark] documentation.

System updates

Over the lifetime of a deployment, the operating system might be subject to vulnerabilities. Some vulnerabilities require system upgrades, whereas others require only configuration changes. All updates require proactive planning and careful testing.

For the operating systems used in production, put a plan in place for avoiding and resolving security issues. The plan should answer the following questions:

- How does your organization become aware of system security issues early?

  This could involve following bug reports, mailing lists, forums, and other sources of information.
- How do you test security fixes, including configuration changes, patches, service packs, and system updates?

  Validate the changes first in development, then in one or more test environments, then in production in the same way you would validate other changes to the deployment.
- How do you roll out solutions for security issues?

  In some cases, fixes might involve both changes to the service, and specific actions by those who use the service.
- What must you communicate about security issues?
- How must you respond to security issues?

Software providers often do not communicate what they know about a vulnerability until they have a way to mitigate or fix the problem. Once they do communicate about security issues, the information is likely to become public knowledge quickly. Make sure that you can expedite resolution of security issues.

To resolve security issues quickly, make sure that you are ready to validate any changes that must be made. When you validate a change, check that the fix resolves the security issue. Validate that the system and Java Agent software continue to function as expected in all the ways they are used.

System audits

System audit logs make it possible to uncover system-level security policy violations that are not recorded in Java Agent, such as unauthorized access to Java Agent files. Such violations are not recorded in Java Agent logs or monitoring information.

Also consider how to prevent or at least detect tampering. A malicious user violating security policy is likely to try to remove evidence of how security was compromised.

Unused features

By default, operating systems include many features, accounts, and services that Java Agent software does not require. Each optional feature, account, and service on the system brings a risk of additional vulnerabilities. To reduce the surface of attack, enable only required features, system accounts, and services. Disable or remove those that are not needed for the deployment.

The features needed to run and manage Java Agent software securely include the following:

- A Java runtime environment, required to run Java Agent software.
- Software to secure access to service management tools; in particular, when administrators access the system remotely.
- Software to secure access for remote transfer of software updates, backup files, and log files.
- Software to manage system-level authentication, authorization, and accounts.
- Firewall software, intrusion-detection/intrusion-prevention software.
- Software to allow auditing access to the system.
- System update software to allow updates that you have validated previously.
- If required for the deployment, system access management software such as SELinux.
- Any other software that is clearly indispensable to the deployment.

Consider the minimal installation options for your operating system, and the options to turn off features.

Consider configuration options for system hardening to further limit access even to required services.

For each account used to run a necessary service, limit the access granted to the account to what is required. This reduces the risk that a vulnerability in access to one account affects multiple services across the system.

Make sure that you validate the operating system behavior every time you deploy new or changed software. When preparing the deployment and when testing changes, maintain a full operating system with Java Agent software that is not used for any publicly available services, but only for troubleshooting problems that might stem from the system being *too* minimally configured.

Network connections

Protect network traffic by using HTTPS where possible.

Recommendations For Incoming Connections (From Clients to Java Agent)

Protocol	Recommendations
HTTP	HTTP connections that are not protected by TLS use cleartext messages. When you permit insecure connections, you cannot prevent client applications from sending sensitive data. For example, a client could send unprotected credentials in an HTTP Authorization header. Even if the server were to reject the request, the credentials would already be leaked to any eavesdroppers. Always use HTTPS for connections up to a load-balancer or proxy in front of the web application or server.
HTTPS	Use HTTPS for secure connections. Follow industry-standard TLS recommendations for Security/Server Side TLS. When using an HTTP connection handler, use HTTPS to protect client connections. Some client applications require a higher level of trust, such as clients with additional privileges or access. Client application deployers might find it easier to manage public keys as credentials than to manage username/password credentials. Client applications can use TLS client authentication.

Recommendations For Outgoing Connections (From Java Agent to Another Service)

Client	Recommendations
Common Audit event handlers	Configure ForgeRock Common Audit event handlers to use HTTPS when connecting to external log services.
AM	Connect to AM over HTTPS, and use Web Socket Secure (WSS) for notifications. When AM listens on HTTPS, by default the agent uses WSS. Otherwise, by default the agent uses Web Sockets (WS).
Custom login pages	Connect to custom login pages over HTTPS.

Message-level security

Server protocols such as HTTP, LDAP, and JMX rely on TLS to protect connections. To enforce secure communication, see Secure communication between the agent and AM.

Communication between the agent and clients is managed by the web application container in which the agent runs. See the web application container documentation for information about how to secure those connections.

Access

The following sections describe how to restrict non-essential access to your deployment, and reduce the amount of non-essential information that it provides.

Remove non-essential features

The more features you have turned on, the more features you need to secure, patch, and audit. If something is not being used, uninstall it, disable it, or protect access to it.

Remove non-essential access

Make sure that only authorized people can access your servers and applications through the appropriate network, using the appropriate ports, and presenting strong-enough credentials.

Make sure that users connect to systems through the latest versions of TLS, and audit system access periodically.

Protect read-access to endpoints that monitor Common REST, Prometheus, CSV file-based metrics. For Common REST and Prometheus endpoints:

- Name exposed base endpoints to prevent them from being easily associated with an application.
- Set up strict not-enforced rules, to minimize unauthenticated access.

For information, see Manage endpoints for Common REST and Prometheus metrics.

Update patches

Prevent the exploitation of security vulnerabilities by using up-to-date versions of the agent and third-party software.

Review and follow the ForgeRock Security Advisories. To receive email notifications for new security advisories, log in to Backstage, and click the **Subscribe** button for agent security advisories. Follow similar lists from all of your vendors.

Manage sessions

On startup, Java Agent uses the following properties to obtain a session from AM:

- Agent Profile Name
- Encrypted Agent Password
- Agent Profile Realm

The session lifetime is defined by the AM version and configuration, and is essentially indefinite. Consider the following points when you configure the agent session lifetime in AM:

- If the lifetime is too short, the agent has to re-authenticate with AM too frequently, using network bandwidth and delaying user requests.
- If the lifetime is too long, the CTS can be cluttered with zombie sessions that are no longer in use.
- A value between 60 minutes and 1440 minutes (24 hours) is suitable for many use cases.

To set the agent session lifetime in AM, add the property `com.iplanet.am.session.agentSessionIdleTime` to the JVM properties in the container where the agent runs, and restart the container. The following example sets the agent session lifetime to 1440 minutes (24 hours):

```
JAVA_OPTS="$JAVA_OPTS -
Dcom.iplanet.am.session.agentSessionIdleTime=1440"
```

Manage cookies

Increase the security of cookies generated by Java Agent or the protected application in the following ways:

- To prevent cookies from being easily associated with an application, change the default name of key cookies. For example, change pre-authentication cookies in Pre-Authentication Cookie Name, and JWT cookies in JWT Cookie Name.
- To transmit securely all cookies written by the agent, set Transmit Cookies Securely.
- To reduce the risk of cross-site request forgery (CSRF) attacks, set the SameSite attribute of cookies in Set-Cookie Internal Map or Set-Cookie Attribute Map.
- To ensure that cookies cannot be accessed through client-side scripts, and to mitigate any XSS attacks, set Enable HTTP Only Cookies to create cookies with the `httpOnly` flag.
- To make cookies accessible only from HTTPS sites, prefix the cookie name with `__Secure-`. A forged insecure site cannot overwrite a secure cookie.
- To make cookies accessible only on the same host where they are set, prefix the cookie name with `__Host-`. A subdomain cannot overwrite the cookie value.
- To protect the CDSSO session cookie from hijacking, configure AM as described in Enabling restricted tokens for CDSSO session cookies in AM's *Security guide*.

Keys and secrets

Java Agent uses cryptographic keys for encryption, signing, and securing network connections, and passwords. The following sections discuss how to secure keys and secrets in your deployment.

Use strong keys

Small keys are easily compromised. Use at least the recommended key size.

In JVM, the default ephemeral Diffie-Hellman (DH) key size is 1024 bits. To support stronger ephemeral DH keys, and protect against weak keys, installations in Tomcat 8.5.37 and later versions use the Tomcat default DH key size of 2048-bit.

Increase the DH key size to protect against weak keys. For more information, refer to Customizing Size of Ephemeral Diffie-Hellman Keys

Rotate keys

Rotate keys regularly to:

- Limit the amount of data protected by a single key.

- Reduce dependence on specific keys, making it easier to migrate to stronger algorithms.
- Prepare for when a key is compromised. The first time you try key rotation shouldn't be during a real-time recovery.
- Conform to internal business compliance requirements.

Rotate the agent profile password

During installation, the agent requests the path to a file containing the agent profile password. The agent then uses the following properties to encrypt and store the password:

- `am.encryption.pwd` in the AgentKey.properties file
- `org.forgerock.agents.encrypted.password` in the AgentPassword.properties file

If the path is empty, the installation terminates with a configuration error.

The following steps describe how to rotate the agent profile password:

1. Change the profile password for your agent instance. For example, in the AM admin UI, change the password as follows:
   a. Select **REALMS** > realm name > **Applications** > **Agents** > **Java**.
   b. Select your agent.
   c. In the **Global** tab, enter a new password in the **Password** field.
2. Generate an encryption key for the agent profile password, using the agentadmin --getEncryptKey command:

   ```
   $ agentadmin --getEncryptKey
   ```

3. In `AgentKey.properties`, set the value of `am.encryption.pwd` to the new value.
4. Encrypt the agent profile password, using the agentadmin --encrypt command:

   ```
   $ agentadmin --encrypt agent-instance password-file
   ```

   The agent encrypts the password by using the value of `am.encryption.pwd` from `AgentKey.properties`.
5. In `AgentPassword.properties`, set the value of `org.forgerock.agents.encrypted.password` to the new value.
6. Restart the agent instance.

Rotate cookie signing keys

During installation, the agent requests the path to a file containing the cookie signing key, and then uses the key to configure the property `org.forgerock.agents.cookie.signing.value` in AgentKey.properties. If the path is empty, cookie signing is disabled.

The key must be at least 64 characters long. If it is shorter, the agent rejects it and leaves cookies unsigned. For security, use a key of at least 80 characters.

The following steps describe how to rotate the cookie signing key for an agent instance:

1. Generate an 80-character key, using the agentadmin --key command:
    1. Unix
    2. Windows

    ```
    $ agentadmin --key 80
    ZRY...xXO
    ```

    ```
    C:> agentadmin --key 80
    ZRY...xXO
    ```
2. In AgentKey.properties, set the value of `org.forgerock.agents.cookie.signing.value` to the key value.
3. Restart the agent instance.

Audits and logs

Audit trails

For security, troubleshooting, and regulatory compliance, agents are able to audit information for allowed and/or denied requests.

The agent audit logging service adheres to the log structure common across the ForgeRock Identity Platform. For information, see Auditing.

Java Agent supports propagation of the transaction ID across the ForgeRock Identity Platform, using the HTTP header `X-ForgeRock-TransactionId`. Consider configuring this header to prevent malicious actors from flooding the system with requests using the

same transaction ID header to hide their tracks. For information, see Configuring the trust transaction header system property in AM's *Security guide*.

Log files

Agent logs can contain informational, error, and warning events, to troubleshoot and debug transactions and events that take place within the agent instance.

Protect logs from unauthorised access, and make sure they contain a minimum of sensitive or personally identifiable information that could be used in attacks.

Use the lowest level of logging necessary. For example, consider logging at the `ERROR` or `WARNING` level, instead of `TRACE` or `MESSAGE`. For more information, see logging configuration properties.