

# Security guide

---

Use this guide to reduce risk and mitigate threats to Web 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. For more information, visit <https://www.pingidentity.com><sup>↗</sup>. The common REST API provides ForgeRock Identity Platform software common ways to access web resources and collections of resources.

# Threats

---

The following sections describe some possible threats to Web 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## Out-of-date software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 Ping Identity 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 [Add Cache-Control Headers](#) so that HTTP responses generated by the agent include the Cache-Control HTTP header.

When the Cache-Control HTTP header is present, the response cannot be cached. The header uses the following values: `max-age=0` , `no-cache` , `no-store` , `must-revalidate` .

If you need the Cache-Control header for each page, set it in the application or in the web server.

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 help to identify Web 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 .

## 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 Web Agent cannot provide generic protection against CSRF. ForgeRock recommends following the latest guidance from the [OWASP CSRF Prevention Cheat Sheet](#) <sup>[↗]</sup>.

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.

To defend against CSRF attacks when POST data preservation is enabled, the agent uses a secure cookie and a nonce. The nonce must correspond to the authentication response from AM. This defense during authentication is specified in [Cross-Site Request Forgery](#) <sup>[↗]</sup>.

ForgeRock strongly recommend using token-based mitigations against CSRF, and relying on other measures only as a defense in depth, in accordance with OWASP guidance.

For more protection against cross-site scripting attacks, configure [Composite Advice Encode](#).

## POST data preservation

POST data is stored temporarily in the agent file system before a user is authenticated. Therefore, any unauthenticated user can POST a file that is then stored by the agent. Consider the following points when you configure POST data preservation:

- Payloads from unauthenticated users are stored in the agent files system. If your threat evaluation does not accept this risk, do not use POST data preservation; set [Enable POST Data Preservation](#) to `false` .
- By default, POST data is stored in the installation log directory, `/path/to/web_agents/agent_type/instances/agent_n/log` . Although the log directory is considered secure, using it for POST data can cause the following risks:

- Permissive access to POST data.

Log directories can have relatively permissive read or copy permissions compared to other directories.

- Leakage of personally identifiable information (PII).

If POST data in log directories is processed by log applications, it could be included in a log view.

To store POST data in a dedicated directory, set [POST Data Storage Directory](#).

- POST data is stored for the time defined by [POST Data Entries Cache Period](#) and then deleted. To identify threats in POST data before it is deleted, make sure that Intrusion Detection Systems inspect the data within the specified time.

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 profile password, see [Preinstallation tasks](#).

### 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, by using a password manager.

## 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.

## Path traversal attempts

Some Java servers, such as those based on J2EE and Spring, use path parameters (also known as matrix parameters) in URL paths. These servers can process requests in a non-standard way that is unsafe for the agent forwarding the request. As a result, bad actors can make use of these parameters for path traversal attempts to access protected resources.

The agent requires the URL path to be normalized consistently to be able to effectively enforce rules.

Web Agent rejects unsafe uses of path parameters with an HTTP 400 in the following scenarios:

- The request contains one or more `%2F` or `%2f` (encoded forward slash) characters in the path parameters.
- The request includes empty path segments or dot path segments with path parameters. Some example unsafe uses include:
  - `/;/`
  - `/..;`
  - `/.;`
  - `/..;parameter/`

Legitimate uses of `;` as a path parameter are still permitted. For example, the agent won't reject this request with the `jsessionId` parameter:

```
/segment1/segment2/;jsessionid=1234
```

## 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 Web Agent software or the connected systems?
- How do you validate mitigation plans and remedial actions?
- How do client applications work when the Web 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.

## Operating systems

---

When you deploy Web Agent, familiarize yourself with the recommendations for the host operating systems that you use. For comprehensive information about securing operating systems, see the [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 Web 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 Web Agent, such as unauthorized access to Web Agent files. Such violations are not recorded in Web 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 Web 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 Web Agent software securely include the following:

- 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 Web 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 Web Agent)*

Protocol	Recommendations
HTTP	<p>HTTP connections that are not protected by SSL/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.</p> <p>Always use HTTPS for connections up to a load-balancer or proxy in front of the web application or server.</p>



Protocol	Recommendations
HTTPS	<p>Use HTTPS for secure connections. Follow industry-standard TLS recommendations for <a href="#">Security/Server Side TLS</a>.</p> <p>When using an HTTP connection handler, use HTTPS to protect client connections.</p> <p>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 SSL client authentication.</p>

#### *Recommendations For Outgoing Connections (From Web 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.

## Message-level security

Server protocols such as HTTP, LDAP, and JMX rely on TLS to protect connections. To enforce secure communication, see [Configure SSL communication between the agent and AM](#).

Communication between the agent and clients is managed by the web server in which the agent runs. For information about how to secure connections, see the web server documentation.

## 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.

Provide access only as necessary; restrict to required users, and limit their access to the information they need.

## 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 Ping Identity security advisories. Follow similar lists from all of your vendors.

## Manage cookies

Increase the security of cookies generated by Web 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 the SSO cookie in [Cookie Name](#).
- To transmit securely all cookies written by the agent, set [Enable Cookie Security](#).
- To reduce the risk of cross-site request forgery (CSRF) attacks, set the SameSite attribute of cookies in [SameSite Cookie Attribute](#).
- To ensure that cookies cannot be accessed through client-side scripts, and to mitigate any XSS attacks, set [Enable HTTP Only Mode](#) 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.

## Keys and secrets

---

Web 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](#).

For more information about strong encryption, see the documentation for the web server where the agent runs. For NGINX, for example, see [Security controls](#).

## 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.

## 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](#).

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

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

Make sure that Agent Debug Level is 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.

Was this helpful?  

---

Copyright © 2010-2025 ForgeRock, all rights reserved.