Understand Finesse BOSH Implementation and Troubleshoot Finesse Agent Desktop Presence Disconnect Issues

Contents

Introduction
Prerequisites
Requirements
Components Used
Understanding Finesse BOSH Implementation
Understanding XMPP
Example XMPP Message
XMPP Implementation with Finesse
Example Finesse XMPP Request/Response
Understanding Finesse XMPP Messages and XMPP Nodes
Example 1: Using Pidgin to View Finesse XMPP Nodes
Example 2: Using Browser Developer Tools Network Tab to View HTTP Messages
Troubleshoot: BOSH Disconnect Error Message
Log Analysis
Debug Notification Service Logs
Info Notification Service Logs
Webservices Logs
Common Reasons for BOSH Disconnect
Problem: Agents disconnect at different times (client side issue)
Recommended Actions
Problem: All agents disconnect at the same time (server side issue)
Recommended Actions
Using Fiddler
Common Fiddler Issue
Example Configuration Steps
Using Wireshark
Related Defects
Related Information

Introduction

This document describes the architecture behind Finesse connections that use Bidirectional-streams Over Synchronous HTTP (BOSH) and how BOSH connection problems can be diagnosed.

Prerequisites
Requirements

Cisco recommends that you have knowledge of these topics:

- Cisco Finesse
- Unified Contact Center Enterprise (UCCE)
- Unified Contact Center Express (UCCX)
- Web browser developer tools
- Windows and/or Mac administration

Components Used

The information in this document is based on these software and hardware versions:

- Cisco Finesse 9.0(1) - 11.6(1)
- UCCX 10.0(1) - 11.6(2)

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Understanding Finesse BOSH Implementation

Understanding XMPP

Extensible Messaging and Presence Protocol (XMPP) (also known as Jabber) is a stateful protocol in a client-server model. XMPP allows for fast delivery of small pieces of structured eXtensible Markup Language (XML) data from one entity to another. XMPP/Jabber is extensively used in instant messaging (IM) and presence applications.

All XMPP entities are identified by their Jabber ID (JID).
JID Addressing Scheme: **user@domain/resource**

- **user**: client username on the XMPP server or name of the conference room
- **domain**: XMPP server fully qualified domain name (FQDN)
- **resource**: identifier of the user’s specific entity/endpoint (e.g., laptop, smartphone, etc), a session identifier, or pubsub node name

**Note**: All three JID components are not used in all cases. A server would typically just be defined by the **domain**, a conference room defined by **user@domain**, and a client by **user@domain/resource**.

XMPP messages are called stanzas. There are three core stanzas in XMPP:

1. `<message>`: one direction, one recipient
2. `<presence>`: one direction, publish to many
3. `<iq>`: info/query - request/response

All stanzas have **to** and **from** addresses and most stanzas also have **type**, **id**, and **xml:lang** attributes.

**Stanza attribute Purpose**

- **to**: destination JID
- **from**: source JID
- **type**: purpose of the message
- **id**: unique identifier used to link a request with a response for `<iq>` stanzas
- **xml:lang**: defines the default language for any human readable XML in the stanza

**Example XMPP Message**
XMPP Implementation with Finesse

If a web application needs to work with XMPP, multiple issues arise. Browsers don't support XMPP over Transmission Control Protocol (TCP) natively, so all XMPP traffic must be handled by a program which runs inside the browser. Web servers and browsers communicate via HyperText Transfer Protocol (HTTP) messages, so Finesse and other web applications wrap XMPP messages inside of HTTP messages.

The first difficulty with this approach is that HTTP is a stateless protocol. This means that each HTTP request is not related to any other request. However, this problem can be addressed by applicative means—for example through the use of cookies/post data.

The second difficulty is the unidirectional behavior of HTTP. Only the client sends requests, and the server can only respond. The server’s inability to push data makes it unnatural to implement XMPP over HTTP.

This problem does not exist in the original XMPP Core specification (RFC 6120), where XMPP is bound to TCP. However, if you want to address the problem with XMPP bound to HTTP, for example, because Javascript can send HTTP requests, there are two possible solutions. Both require a bridge between HTTP and XMPP.

The proposed solutions are:

1. Polling (legacy protocol): repeated HTTP requests asking for new data defined in XEP-0025: Jabber HTTP Polling

2. Long polling also is known as BOSH: transport protocol that emulates the semantics of a long-lived, bidirectional TCP connection between two entities by efficiently using multiple synchronous HTTP request/response pairs without requiring the use of frequent polling defined in XEP-0124: HTTP Binding and extended by XEP-0206: XMPP Over BOSH

Finesse implements BOSH as it is quite efficient from the server load point of view and traffic-wise. The purpose of using BOSH is to cover up the fact that the server does not have to respond as soon there is a request. The response is delayed up to a specified time until the server has data for the client, and then it is sent as a response. As soon as the client gets the response, the client makes a new request and so forth.

The Finesse desktop client (web application) establishes a stale BOSH connection over TCP port 7443 every 30 seconds. After 30 seconds, if there are no updates from the Finesse Notification Service, the Notification service sends an HTTP reply with a 200 OK and a (nearly) empty response body. If the Notification Service has an update on the presence of an agent or a dialog (call) event, for example, the data is sent immediately to the Finesse web client.

Example Finesse XMPP Request/Response

This example shows the first XMPP message request response shared between the Finesse client and Finesse server to set up the BOSH connection.
Finesse client request:

```xml
<body xmlns="http://jabber.org/protocol/httpbind" xml:lang="en-US" xmlns:xmpp="urn:xmpp:xbosh" hold="1" ver="1.9" to="fin1.ucce.local" wait="30" xmpp:version="1.0" from="47483648@fin1.ucce.local" rid="704654808"/>
```

Finesse server response:

```xml
<body xmlns="http://jabber.org/protocol/httpbind"
```

To summarize:

1. The Finesse web client has a stale HTTP connection (http-bind) set up to the Finesse server via TCP port 7443. This is known as a BOSH long poll.

2. The Finesse Notification Service is a presence service that posts updates regarding the state of an agent, call, etc.

3. If the Notification service has an update, it replies to the http-bind request with the state update as an XMPP message in the HTTP response body.

4. If there are no state updates 30 seconds after receiving the http-bind request, the Notification Service replies without any state updates to allow the Finesse web client to send another http-bind request. This serves as a way for the Notification service to know that the Finesse web client is still able to connect to the Notification Service and that the agent didn’t close their browser or put their computer to sleep, etc.

Understanding Finesse XMPP Messages and XMPP Nodes

Finesse also implements XMPP specification **XEP-0060: Publish-Subscribe**. The purpose of this specification is to allow the XMPP server (Notification service) to get information published to XMPP nodes (topics) and then to send XMPP events to entities subscribed to the node. In the case of Finesse, the Computer Telephony Integration (CTI) server sends CTI messages to the Finesse web service to tell Finesse about configuration updates such as, but not limited to, agent or Contact Service Queue (CSQ) creation or information about a call. This information is then converted into an XMPP message that the Finesse web service publishes to the Finesse Notification service. The Finesse Notification service then sends XMPP over BOSH messages to agents that are subscribed to certain XMPP nodes.

Some of Finesse API objects that are defined in the [Finesse Web Services Developer Guide](#) are XMPP nodes. Agent and supervisor Finesse web clients may subscribe to event updates for some of these XMPP Nodes in order to have up-to-date information about real-time events (such as call events, state events, and so on). This table shows the XMPP nodes that are pubsub enabled.

<table>
<thead>
<tr>
<th>Finessse API Object</th>
<th>Purpose</th>
<th>Subscription</th>
</tr>
</thead>
<tbody>
<tr>
<td>/finesse/api/User/&lt;LoginID&gt;</td>
<td>Shows the state and team mapping of the agent</td>
<td>Agents and Supervisors</td>
</tr>
<tr>
<td>/finesse/api/User/&lt;LoginID&gt;/Dialogs</td>
<td>Shows the call(s) being handled by the agent</td>
<td>Agents and Supervisors</td>
</tr>
</tbody>
</table>
/finesse/api/User/<LoginID>/ClientLog  Used to capture client logs from the Send Error Report button

/一緒/finesse/api/User/<LoginID>/Queue/<queueID> Shows queue statistics data (if enabled)

/一緒/finesse/api/Team/<TeamID>/Users Shows the agents that belong to a certain team including state information

/一緒/finesse/api/SystemInfo Shows the state of the Finesse server. Used to determine if failover is needed

Example 1: Using Pidgin to View Finesse XMPP Nodes

Step 1. Download and install the XMPP client Pidgin.

Step 2. Navigate to Accounts > Modify > Basic and configure the Login Options:

- Protocol: XMPP
- Username: LoginID for any agent
- Domain: FQDN of Finesse server
- Resource: Placeholder - any value may be used, for example, test
- Password: Agent password
- Check the Remember password checkbox
Step 3. Navigate to **Accounts > Modify > Advanced** and configure:

- **Connection security**: Use encryption if available
- Check the **Allow plaintext auth other unencrypted streams**
- **Connection port**: 5222. Use the default port 5222. This port is required for external XMPP clients. Finesse desktop clients use 7443. Do not use port 7443.
- **Connection server**: Finesse server FQDN
Note: Port 5222 is used because only Finesse web clients can use port 7443 to connect to the Notification Service.

Step 4. Navigate to **Tools > Plugins** and enable the XMPP Console.
Step 5. Navigate to **Tools > XMPP Console > XMPP Console** to open the XMPP Console.
Step 6. Execute this `<iq>` message to see all of the XMPP nodes that exist.

```xml
<iq type='get' from='loginID@<Finesse_FQDN>/test' to='pubsub.<Finesse_FQDN>' id='testId1'
<query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

For example:

```xml
<iq type='get' from='47483648@fin1.ucce.local/test' to='pubsub.fin1.ucce.local' id='testId1'>
<query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```
In a lab environment with two agents and two CSQs configured, this output is contained in the Finesse response:

```xml
<iq type='result' id='testId1' from='pubsub.fin1.ucce.local' to='47483648@fin1.ucce.local/test'>
<query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/Team/5000/Users'/>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/User/47483648/Dialogs'/>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/User/47483651'/>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/User/47483649/Queues'/>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/Team/5001/Users'/>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/User/47483650/ClientLog'/>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/User/47483651/Dialogs'/>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/User/47483650/Queues'/>
    <item jid='pubsub.fin1.ucce.local' name='' node='/finesse/api/Team/1/Users'/>
</query>
</iq>
```
Example 2: Using Browser Developer Tools Network Tab to View HTTP Messages

Each browser has a set of developer tools. The **Network** tab of the developer tools shows the HTTP messages sent and received by the Finesse web client (browser). For example, this image shows how the Finesse web client sends a SystemInfo request which checks Finesse Tomcat status every minute as a failover check. Additionally, the http-bind messages from the BOSH connection are also displayed. The Finesse server sends back a response within 30 seconds if there are no updates to publish on the XMPP nodes the web client is subscribed to.

Troubleshoot: BOSH Disconnect Error Message

When a BOSH disconnect occurs, the error "Lost connection to {Finesse Server FQDN}. Please wait for a reachable Finesse Server to be found..." is displayed in a red banner at the top of the Finesse desktop.

This message is displayed because at this time, no XMPP subscription events can be received from the Cisco Finesse Notification Service. Hence, state information and call details cannot be displayed on the agent desktop.

For UCCX, 60 seconds after the browser disconnects, the agent is put into a **Logout** state. The agent can be in the **Ready** or **Not Ready** state for the logout to happen.

For UCCE, Finesse takes up to 120 seconds to detect when an agent closes the browser or the
browser crashes and Finesse waits 60 seconds before sending a forced logout request to the CTI server which causes the CTI server to put the agent in a Not Ready state. Under these conditions, Finesse can take up to 180 seconds to sign out the agent. Unlike in UCCX, the agent moves into a Not Ready state instead of the Logout state.

**Note:** The CTI disconnect Not Ready vs. Logout state behavior in UCCE is controlled by the PG /LOAD parameter. Per the Release Notes for Unified Contact Center Enterprise & Hosted Release 10.0(1), the /LOAD parameter is no longer supported starting in UCCE 10.0.

For more information on UCCE Finesse Desktop behavior, refer to the Desktop Behavior section of the Cisco Finesse Failover Mechanisms chapter in the Cisco Finesse Administration Guide.

**Note:** Timer values might change in future as per the product requirement.

**Log Analysis**

The Finesse and UCCX Notification service logs can be collected via RTMT or via the CLI:

`file get activelog /desktop recurs compress`

**Debug Notification Service Logs**

**Note:** Set debug level logs only while reproducing an issue. Turn off the debugs after the issue has been reproduced.

**Note:** Finesse 9.0(1) does not have debug level logging. Debug level logging was introduced in Finesse 9.1(1). The process for enabling the logging is different in 9.1(1) compared to Finesse 10.0(1) - 11.6(1). For this process, consult the Finesse Administration and Serviceability guide.

Enable Notification Service debug logs of Unified Contact Center Express (UCCX), as shown:

```
admin:utils uccx notification-service log enable
```

WARNING! Enabling Cisco Unified CCX Notification Service logging can affect system performance and should be disabled when logging is not required.

Do you want to proceed (yes/no)? yes

Cisco Unified CCX Notification Service logging enabled successfully.

NOTE: Logging will be disabled automatically if Cisco Unified CCX Notification Service is restarted.

Enable Notification Service debug logs of Unified Contact Center Enterprise (UCCE) (Finesse Standalone), as shown:

```
admin:utils finesse notification logging enable
```
Checking that the Cisco Finesse Notification Service is started...
The Cisco Finesse Notification Service is started.

Cisco Finesse Notification Service logging is now enabled.

WARNING! Cisco Finesse Notification Service logging can affect system performance and should be disabled when logging is not required.

Note: Logging will be disabled automatically if you restart the Cisco Finesse Notification Service.

These logs are in the /desktop/logs/openfire folder and are named **debug.log**.

As shown in the image, the Notification Service (Openfire) **debug.log** shows the http binding with desktop along with the IP address and port of the agent PC.

As shown in the image, the **last active 0 ms** shows that session is still active.

Openfire closing the idle session indicates the agent logout will trigger in 60 seconds where Finesse will send a forced logout with a reason code of 255 to the CTI server. The actual behavior of the desktop under these conditions depends on the setting for Logout on Agent Disconnect (LOAD) in UCCE. In UCCX, this is always the behavior.

If the Fineese client does not send http-bind messages to the Finesse server, the logs will show the session up time and show the session close.

admin: **utils finesse notification logging enable**

Checking that the Cisco Finesse Notification Service is started...
The Cisco Finesse Notification Service is started.

Cisco Finesse Notification Service logging is now enabled.

WARNING! Cisco Finesse Notification Service logging can affect system performance and should be disabled when logging is not required.

Note: Logging will be disabled automatically if you restart the Cisco Finesse Notification Service.

**Info Notification Service Logs**

These logs are in the /desktop/logs/openfire folder and are named **info.log**. If the Fineese client does not send http-bind messages to the Finesse server, the logs will show the the session become inactive.
admin: utils finesse notification logging enable

Checking that the Cisco Finesse Notification Service is started...
The Cisco Finesse Notification Service is started.

Cisco Finesse Notification Service logging is now enabled.

WARNING! Cisco Finesse Notification Service logging can affect system performance and should be disabled when logging is not required.

Note: Logging will be disabled automatically if you restart the Cisco Finesse Notification Service

Webservices Logs

These logs are in the /desktop/logs/webservices folder and are named Desktop-webservices.YYYY-MM-DDTHH-MM-SS.sss.log. If the Fineese client does not send http-bind messages to the Finesse server within the specified amount of time, the logs will show the the agent presence become unavailable and 60 seconds later, a presense driven logout will occur.

admin: utils finesse notification logging enable

Checking that the Cisco Finesse Notification Service is started...
The Cisco Finesse Notification Service is started.

Cisco Finesse Notification Service logging is now enabled.

WARNING! Cisco Finesse Notification Service logging can affect system performance and should be disabled when logging is not required.

Note: Logging will be disabled automatically if you restart the Cisco Finesse Notification Service

Common Reasons for BOSH Disconnect

BOSH connections are setup by the web client, and the Finesse server determines if the agent presence is unavailable. These issues are almost always client side issues relating to the browser, agent computer, or network as the onus of starting up the connection is up to the client.

Problem: Agents disconnect at different times (client side issue)

Recommended Actions

Check for these issues:

1. Network issue:

   - Review firewall rules and logs -- TCP port 7443 must not be blocked or throttled

   - Use an HTTP web traffic sniffer like Fiddler® or Wireshark® to confirm that the browser sends http-bind requests over TCP port 7443 and receives responses

   - Check all network devices/interfaces between the agent computer and the Finesse server for excessive delay or packet drops Traceroute may be useful to determine the path and determine delays On a Microsoft® Windows® PC: tracert {Finesse Server IP | Finesse
On a Mac®: traceroute {Finesse Server IP | Finesse Server FQDN}

On Cisco IOS® software, the interface statistics can be checked: show interfaces

Refer Troubleshooting Input Queue Drops and Output Queue Drops

- Collect Finesse Client logs for a test agent. Client logs can be collected in three ways:
  - Browser web console logs
  - Firefox web console
  - Internet Explorer web console
  - Chrome web console
  - Press the Send Error Report button on the Finesse page and collect the Finesse server logs. The logs are located in /desktop/logs/clientlogs. Log in via https://<Finesse-FQDN>/desktop/locallog and collect the logs after the issue occurs.

Every minute, the client connects to the Finesse server to calculate drift and network latency:

```bash
admin:utils finesse notification logging enable
```

Checking that the Cisco Finesse Notification Service is started...
The Cisco Finesse Notification Service is started.

Cisco Finesse Notification Service logging is now enabled.

WARNING! Cisco Finesse Notification Service logging can affect system performance and should be disabled when logging is not required.

Note: Logging will be disabled automatically if you restart the Cisco Finesse Notification Service

In case of any log collection issues, refer Troubleshoot Cisco Finesse Desktop Persistent Logging Problem

2. Unsupported browser and/or version:

Use supported browser/version and settings as per the compatibility matrices:

UCCE Compatibility Matrix

UCCX Compatibility Matrix

3. Browser stuck condition due to content/processing of other tab/window:

Check the agent workflow to see if they:

- Commonly have other tabs or windows up that are constantly running other real-time applications such as music/video streaming, WebSocket connections, custom Customer Relationship Management (CRM) web clients, etc
- Have a very large number of tabs or windows open
- Have disabled browser caching
- Have kept their browser running for a long time and do not close the browser at the end of the workday

4. Computer put to sleep:

Check to see if the agent puts their computer to sleep before logging out of Finesse or if their computer sleep setting timer is very low.

5. High CPU or high memory issue on client computer:
- If the agent browser runs in a shared environment such as Microsoft Windows Remote Desktop Services, Citrix® XenApp®, Citrix XenDesktop®, determine if the browser performance depends on the number of users running the browser at the same time. Ensure that the proper memory and CPU resources is configured based on the number of users.

- Check computer resource utilization issues:
  
  Windows: Windows PowerShell `Get-Counter` command that checks % of CPU time, Megabytes of memory available, and % of memory in use every 2 seconds: `Get-Counter -Counter "\Processor(_Total)\% Processor Time",\Memory\Available MBytes","\Memory\% Committed Bytes In Use" -SampleInterval 2 -Continuous
  
  Alternative to using PowerShell to view the Windows' performance counters, Windows Performance Monitor can be used. Task Manager can be used to view live CPU and memory statistics globally and on a process-by-process basis.

  Mac: Terminal `top` command that checks live total CPU and memory: `top` Check processes and sort by CPU utilization: `top -o CPU` Check processes and sort by memory utilization: `top -o MEM` Activity Monitor can be used to view live CPU and memory statistics globally and on a process-by-process basis.

  6. 3rd party gadgets performing unexpected, problematic activity in background:

Test the Finesse desktop behavior with all 3rd party gadgets removed.

7. NTP issue on server or client:

- Check `utils ntp status` on the Finesse publisher server to ensure the NTP server stratum is 4 or lower
- In the client logs, check drift and network latency

**Problem: All agents disconnect at the same time (server side issue)**

**Recommended Actions**

Check for these issues:

1. Cisco Unified Communications Manager CTIManager service disconnect. If all CTIManager providers for UCCX are shutdown or crash, UCCX agents see the red banner error. UCCE agents do not see the red banner if this happens, but calls fail to route properly to the agents.

- Check to see if the Cisco CTIManager service is started on the CUCM servers used as CTI providers
- Check to see if the Cisco CTIManager service crashed via the Event Viewer - Application logs on RTMT to see if the Cisco CTIManager service crashed. To collect event viewer logs on RTMT: `System > Tools > Trace and Log Central > Collect Files > Select System Services/Applications > Event Viewer-Application Log`
To collect the Event Viewer-Application logs on CLI:

```
file get activelog /syslog/CiscoSyslog*
abstime hh:mm MM/DD/YY hh:mm MM/DD/YY
```

To view core dumps on the CLI:

```
utils core active list
```

Note: Core dumps file names use the format: `core.<ProcessID>.<SignalNumber>.<ProcessName>.<EpochTime>`. Example: `core.24587.6.CTIManager.1533441238` Hence, the time of the crash can be determined from the epoch time.

2. Finesse/UCCX Notification Service stopped or crashed:

- Check the Event Viewer-Application Logs for Notification Service errors or to see if the service was stopped
- Check to see if the Notification service is up: `utils service list`
- Check the times the Notification service shut down: `file search activelog /desktop/logs/openfire "Openfire stopped"`
- Check the times the Notification service started: `file search activelog /desktop/logs/openfire "HTTP bind service started"`
- Check for Notification service memory dumps that resulted from a crash: `file list activelog /desktop/logs/openfire/*.hprof`
- Check to see if the Notification service is listening for traffic on TCP port 7443: `show open ports regexp 7443.*LISTEN`
- Check to see if these defects are applicable (these defects would cause login failure for agents logging in and for agents already logged in, those agents would see the red banner Finesse disconnect message): CSCva72280 - Finesse Tomcat and Openfire Crash for invalid XML characters CSCva72325 - UCCX: Finesse Tomcat and Openfire Crash for
Restart Cisco Finesse Tomcat and Notification Service if a crash is suspected. This is only recommended in a network down situation, otherwise, these restarts disconnect agents from the Finesse server.

- Steps for UCCE: `utils service stop Cisco Finesse Tomcat` `utils service stop Cisco Finesse Notification Service` `utils service start Cisco Finesse Tomcat` `utils service start Cisco Finesse Notification Service`
- Steps for UCCX: `utils service stop Cisco Finesse Tomcat` `utils service stop Cisco Unified CCX Notification Service` `utils service start Cisco Finesse Tomcat` `utils service start Cisco Unified CCX Notification Service`

Using Fiddler

Configuring Fiddler can be a somewhat challenging task without understanding the steps needed and understanding how Fiddler works. Fiddler is a man-in-the-middle web proxy that stands between the Finesse client (web browser) and the Finesse server. Due to the connections being secured between the Finesse client and Finesse server, this adds a layer of complexity to the Fiddler configuration in order to view secured messages.

Common Fiddler Issue

Since Fiddler stands in between the Finesse client and Finesse server, the Fiddler application needs to create signed certificates for all Finesse TCP ports that require certificates:

Cisco Finesse Tomcat service certificates

1. Finesse publisher server TCP 8445 (and/or 443 for UCCE)
2. Finesse subscriber server TCP 8445 (and/or 443 for UCCE)

Cisco Finesse (Unified CCX) Notification Service certificates

1. Finesse publisher server TCP 7443
2. Finesse subscriber server TCP 7443

HTTPS decryption must be enabled for Fiddler to dynamically generate certificates on behalf of the Finesse server. This is not enabled by default.

If HTTPS decryption is not configured, the initial tunnel connection to the Notification service is seen, but the http-bind traffic is not. Fiddler only shows:

```
admin:utils finesse notification logging enable
```

Checking that the Cisco Finesse Notification Service is started...
The Cisco Finesse Notification Service is started.

Cisco Finesse Notification Service logging is now enabled.

WARNING! Cisco Finesse Notification Service logging can affect system performance and should be disabled when logging is not required.

Note: Logging will be disabled automatically if you restart the Cisco Finesse Notification Service
Then, the Finesse certificates signed by Fiddler must be trusted by the client. If these certificates are not trusted, moving past the Establishing encrypted connection... stage of Finesse login is not possible.

In some cases, accepting the certificate exceptions from the login does not work, and the certificates need to be trusted by the browser manually.

Example Configuration Steps

**Caution:** The example configuration provided is for Fiddler v5.0.20182.28034 for .NET 4.5 and Mozilla Firefox 64.0.2 (32-bit) on Windows 7 x64 in a lab environment. These procedures may not generalize to all versions of Fiddler, all browsers, or all computer operating systems. If your network is live, ensure that you understand the potential impact of any configuration. Reference the official Fiddler documentation for more information.

**Step 1.** Download Fiddler

**Step 2.** Enable HTTPS decryption: Tools > Options > HTTPS > check the Decrypt HTTPS traffic checkbox
Step 3. A warning message box open to ask to trust the Fiddler Root Certificate. Select Yes.

Step 4. A warning message box opens with the message "You are about to install a certificate from a certification authority (CA) claiming to represent: DO_NOT_TRUST_FiddlerRoot... Do you want to install this certificate?". Select Yes.

Step 5. Manually add the Finesse publisher and subscriber certificates to the computer or browser certificate trust store. Ensure ports 8445, 7443, and (only for UCCE) 443. For example, on Firefox, this can be done simply without downloading certificates from the Finesse Operating System Administration page:

Options > Find in Options (search) > Certificates > Servers > Add Exception > Location
> Enter https://<Finesse server>:port for the relevant ports for both Finesse servers.
Step 6. Log into Finesse and see the http-bind messages leave the Finesse client to the Finesse Server via Fiddler.

In the example provided, the first 5 messages show http-bind messages that were responded to by the Finesse server. The first message contains 1571 bytes of data returned in the message body. The body contains an XMPP update regarding an agent event. The final http-bind message has been sent by the Finesse client, but has not gotten a response from the Finesse server. This can be determined by seeing that the HTTP result is null (-) and the number of bytes in the response body is null (-1).
Closer view of the data:

Response body for XMPP message:
Using Wireshark

Wireshark is a commonly used packet sniffing tool that can be used to sniff and decode HTTPS traffic. HTTPS traffic is HTTP traffic secured over Transport Layer Security (TLS). TLS provides integrity, authentication and confidentiality between to hosts. It is used commonly in web applications, but it can be used with any protocol that uses TCP as the transport layer protocol. Secure Sockets Layer (SSL) is the former version of the TLS protocol, which is no longer used as it is insecure. These names are often used interchangeably, and the Wireshark filter used for SSL or TLS traffic is `ssl`.

**Caution:** The example configuration provided is for Wireshark 2.6.6 (v2.6.6-0-gdf942cd8) and Mozilla Firefox 64.0.2 (32-bit) on Windows7 x64 in a lab environment. These procedures may not generalize to all versions of Fiddler, all browsers, or all computer operating systems. If your network is live, ensure that you understand the potential impact of any configuration. Reference the [official Wireshark SSL documentation](#) for more information. Wireshark 1.6 or greater is required.

**Note:** This method will only work for Firefox and Chrome. This method does not work for Internet Explorer.

**Step 1.** On the agent's Windows PC navigate to **Control Panel > System and Security > System > Advanced system settings Environmental Variables...**
Step 2. Navigate to User variables for user <username> > New...

Create a variable named SSLKEYLOGFILE.

Create a file to store the SSL pre-master secret in a private directory: SSLKEYLOGFILE=/path/to/private/directory/with/logfile

Note: Creating a system variable instead of a user variable and/or storing the file in a non-private directory will also work, but then all users on the system can access the pre-master secret, which is less secure.
Step 3. If Firefox or Chrome are open, close the applications. After they are reopened, they will start writing to the **SSLKEYLOGFILE**.

Step 4. On Wireshark, navigate to Edit > Preferences...

Navigate to Protocols > SSL.

Step 5. Enter the location of the pre-master secret log filename configured in Step 2.
Step 6. Using Wireshark filter `tcp.port==7443 && ssl`, the secured HTTP communication between the Finesse client and Finesse server (Notification Service) is seen decrypted.

Related Defects

- **CSCva72280** - Finesse Tomcat and Openfire Crash for invalid XML characters
- **CSCva72325** - UCCX: Finesse Tomcat and Openfire Crash for invalid XML characters
Related Information

- XMPP Specifications
- XEP-0124: BOSH
- XEP-0060: Publish-Subscribe
- Firefox web console
- Internet Explorer web console
- Chrome web console
- Windows PowerShell
- Windows Performance Monitor
- Troubleshooting Input Queue Drops and Output Queue Drops
- Windows Task Manager
- Mac Terminal
- Mac Activity Monitor
- Fiddler Download
- Fiddler Configuration
- Wireshark Download
- Wireshark SSL Decryption
- Technical Support & Documentation - Cisco Systems