WAF Security Rules

DLZP utilizes a URI based security protocol using either AWS WAF or F5 Big IP implementation. Our custom rules parse every HTTP request and break them down into their component parts based on your application. Our code will then filter the URI components at each level and either allow or deny access based on our custom whitelists.

PeopleSoft

Every PeopleSoft Portal request can be broken down into four type of requests:

- Basic portal URL format.
- URL format for PeopleSoft Pure Internet Architecture content types.
- URL format for unwrapped PeopleSoft Pure Internet Architecture content.
- Pagelet URLs.
- System URLs.
- Proxy architecture and relative URLs.

Basic Portal URL Format, Press Enter to collapse

URLs provide the addresses for content so that it can be located and correctly identified. The portal servlet needs three pieces of information to present a page to the user. These are integral parts of a PeopleSoft portal URL:

- The name of the portal through which content is being accessed.
- The node that hosts the content.
- The type and ID of the content.

The portal servlet uses the node name in the URL to determine the location of the servlet for that node (stored as part of the node object). If the content is hosted by the local database, the portal servlet communicates with the content servlet directly (Java method calls), not through HTTP (using the portal content servlet URL).

The ID format is different for each content type. Components are identified by menu, component, and market; scripts are identified by the program name, and so on. The portal registry determines the content reference for this content, and for the template and the pagelets that appear around with it.

Image: URL structure

The following diagram shows how the URL lists the required information from left to right: portal (EMPLOYEE), node (PSFT_HR), content type ©, content ID (MAINTAIN_SECURITY.USERMAINT.GBL) for this example URL:

http://sun09/psp/ps/EMPLOYEE/PSFT_HR/c/MAINTAIN_SECURITY.USERMAINT.GBL

This is consistent with the logical organization of content in portals and databases. At the left side of the figure, portals point to nodes and, ultimately, to content within those nodes.

Here's what a portal URL looks like:

http://server/servlet_name/sitename/portalname/nodename/content_type/content_id?content_parm

The following table describes the different sections of the URL:

Section	Description
http://server/	Scheme (HTTP / HTTPS) and web server name.
servlet name/	The name of the physical servlet that the web server invokes to handle the request.
sitename/	The site name specified during PeopleSoft Pure Internet Architecture setup. This enables you to set up multiple sites on one physical web server. The site name is ultimately mapped by the web server to the appropriate configuration.properties file.
portalname/	Name of the portal to use for this request. The portal definition contains metadata that describes how to present the content (template, pagelets, and so on).
nodename/	Name of the node that contains the content for this request.
content_type/	Type of the content for this request.
content_id	The identification of the content. The ID and type ensure that the correct content is retrieved.
?content parm	Query string parameters (name value pairs) for the content.

URL Format for PeopleSoft Pure Internet Architecture Content Types, Press Enter to collapse

This table lists the URL formats for each PeopleSoft Pure Internet Architecture content type:

Content Type	URL Format	Example
Component	/c/menu.component.market/?Page=page&Action=action &key_id=key_value. . .	http://sun09/psp/ps/EMPLOYEE/PSFT_HR/c/ MAINTAIN_SECURITY.USERMAINT.GBL?page=view&view=narrow
Script	/s/recordname. fieldname. event. function/?parm_id=parm_value. . .	http://sun09/psp/ps/EMPLOYEE/PSFT_HR/s/ WEBLIB_Portal.PORTAL_HEADER.FieldChange.Iscript_DoSomething
External	/e/?url=URL	http://sun09/psp/ps/EMPLOYEE/PSFT_HR/e/?url=http%2f%3a%3awww.example.com
Homepage	/h/?tab=homepage_tab	http://sun09/psp/ps/EMPLOYEE/PSFT_HR/h/?tab=HR homepage tab
		Note: This homepage URL tells the portal servlet to display the specified tab of the current user's homepage.
Query	/q/query	http://sun09/psp/ps/EMPLOYEE/PSFT_HR/q/my_query
Worklist	/w/worklist	http://sun09/psp/ps/EMPLOYEE/PSFT_HR/w/my_worklist
File	/f/filename	http://sun09/psp/ps/EMPLOYEE/PSFT_HR/f/myfile.html
		Note: The file URL for file content tells the servlet to retrieve the named file from the database and return it to the browser.
Disconnect	/?disconnect=y	http://sun09/psc/ps/EMPLOYEE/PSFT_HR/s/WEBLIB_QETEST.ISCRIPT1.FieldFormula.IScript_WhoAmI/?disconnect=y&postDataBin=y
		Note: This is an internal flag used by xmllink to process a request and to invalidate the HTTP session afterwards. In a normal browser, the HTTP session is maintained so that the subsequent requests are not creating new HTTP sessions.

URL Format for Unwrapped PeopleSoft Pure Internet Architecture Content, Press Enter to collapse

PeopleSoft Pure Internet Architecture content is accessible with no template wrapping by means of the content servlet. This enables portals to implement a proxied architecture and enables you to include PeopleSoft Pure Internet Architecture content in other portal products and websites.

URLs for unwrapped PeopleSoft Pure Internet Architecture content are similar to URLs for wrapped content. Unwrapped PeopleSoft Pure Internet Architecture content has not gone through the portal servlet template process. In the URL, the content servlet is specified rather than the portal servlet. The following table shows sample URLs for a component and an iScript.

Note: The content servlet ignores the portal and node name, but they are still necessary as placeholders. Omitting them causes a runtime error because the psc servlet checks that the URL contains a portal and node name, even though it doesn't use them.

Unwrapped Content Type	URL Example
Component	http://sun09/psc/ps/EMPLOYEE/PSFT_HR/c/E_PRO.CheckOut.GBL
Script	http://sun09/psc/ps/EMPLOYEE/PSFT_HRm/s/WEBLIB_Portal.PORTAL_HEADERFieldChange.Iscrip_DoSomething

Pagelet URLs, Press Enter to collapse

Pagelets are snippets of HTML content that appear in one section of a template. Unlike target content, they are referenced by name within the template. This sample HTML refers to a pagelet:

<Pagelet Name="UniversalNavigation"> <Source Node ="LOCAL_NODE" Pagelet="MyPagelet"/> </Pagelet>

When the servlet resolves this tag, it generates the URL using:

- The name of the portal containing this template.
- The node name specified in the Source tag (for example, Node = "SomeNode").
- Content type and content name specified in the pagelet definition for the specified pagelet.

The resolved URL for this example is (assume this template is in the Employee portal):

http://sun09/psc/ps/EMPLOYEE/PSFT_HR/s/WEBLIB_TEST.ISCRIPT1.FieldFormula.IScript_WhoAmI

System URLs, Press Enter to collapse

System URLs do not have content or query strings; instead, they contain system commands, such as Login or Expire. System URLs can be issued to both the content and portal servlets. This table lists system URLs:

URL	Description
http://sun09/psp/ps/?cmd=expire	Closes the current session and returns the expire page.
	Note: The system detects whether the user is still active during the cmd=expire process on the server. The system ignores the session after detecting that the user is still active so that ongoing transactions in the active window are not disturbed. However, if another session is open simultaneously that is inactive, the inactive session will be closed.
http://sun09/psp/ps/?cmd=logout	Closes the current session and returns to the sign-in page.
http://sun09/psp/ps/?cmd=login	Tells the servlet to return to the sign-in page.

Proxy Architecture and Relative URLs, Press Enter to

collapse

A relative URL is written to an HTML document without some portion of the scheme, server, or path. When the browser downloads the document containing the relative URL, it makes the relative URL absolute by adding the scheme, server, and path of the downloaded document. Relative URLs simplify moving static documents around on web servers because you don’t have to change the URLs embedded within the documents that you move.

One portal servlet can proxy content from several other content servlets. The portal servlet acts as an intermediary in the conversation between the browser and the various content services, relaying HTTP requests and responses from the content servlet to the browser.

The portal servlet acts as a reverse proxy server by ensuring that all URL references on portal pages point back to the portal servlet itself. The portal servlet does this by rewriting all content retrieved through the portal to contain relative URLs in appropriate URL formats.

To increase performance, you can include the custom header UsesPortalRelativeURL with the value True to indicate that the URL is already set in the correct format. All content from databases using PeopleTools 8.42 and later generates URLs with the correct format and uses this command in the header.

Examples of Relative PeopleSoft URLs

For example, assume that the MAINTAIN_SECURITY.USERMAINT_SELF.GBL component is in the PSFT_HR node, and it is being accessed by the EMPLOYEE portal.

If you wanted a navigation iScript within the PSFT_HR node to construct a link to the MAINTAIN_SECURITY.USERMAINT_SELF.GBL component, you would add the following HTML to the response:

<a href=“../../../EMPLOYEE/PSFT_HR/c/MAINTAIN_SECURITY.USERMAINT_SELF.GBL” . . . >

When this HTML is downloaded to the browser, the absolute URL includes the scheme, server, and servlet directory of the proxying portal servlet, even though the iScript may have run on a content servlet on a different web server. The absolute URL continues with the portal, node, service type, and component name, as specified by the iScript.

Here is what the final URL looks like:

http://sun0server/psp/ps/EMPLOYEE/PSFT_HR/c/MAINTAIN_SECURITY.USERMAINT.GBL

Note: The content services always specify the portal, node, and content type (with the “../../..) even if those values are the same as the current page.

Now assume that you want the navigation iScript to create a link to the MAINTAIN_SECURITY.USERMAINT_SELF.GBL component in the HRMS node. Also assume that the component is being accessed by the employee portal. The navigation iScript adds the following HTML to the response:

<a href=“../../../EMPLOYEE/HRMS/c/MAINTAIN_SECURITY.USERMAINT_SELF.GBL” . . .> The absolute

URL looks like this:

http://sunserver/psp/ps/EMPLOYEE/HRMS/c/MAINTAIN_SECURITY.USERMAINT.GBL

The URL correctly points to the appropriate content without any HTML parsing or URL rewriting.

Finally, assume that you want a navigation iScript running within the EMPLOYEE portal to construct a link to the MAINTAIN_SECURITY.USERMAINT_SELF.GBL component within the e_benefits portal. To construct this link, the iScript generates the following HTML:

<a href="../../../E_BENEFITS/HRMS/c/MAINTAIN_SECURITY.USERMAINT_SELF.GBL" . . . >

Note: The HREF tag with a relative URL can be used to change a portal or node only if the HTML is being accessed through an HTML template. It won't work with a frame template because the base URI of the frame points to the content servlet, which ignores the portal and node names. Use the PeopleCode transfer function to specify a target portal and node.

F5 Implementation

Using the F5 BigIP Security platform DLZP has developed custom iRules that are portable, customizable, and robust. Our iRules inspect every HTTP request and parse the URL into each of its component parts as documented above. The iRules will then evaluate the host, site, portal, node, component, pagelet, script, worklist, and homepage. Each of these individuals parts of the URL and URI will be allowed only if the request satisfies every portion of our customizable whitelist. All other requests will be denied. Additionally, detailed logs will be generated on the F5 BigIP where each part of the request will report whether it was allowed or denied and additions or deletions to the whitelist can be easily made. Logging can be disabled if so desired. Additionally, we have built in a promiscuous mode to allow all traffic and simply evaluated logs for testing purposes.

```
when HTTP_REQUEST timing on {
# set environment
set env "tst"
#portal site variable for login redirect
set portal_site "portst"

#set logging on = 1, off = 0
set logging 1
# set promiscuos mode (all traffic allowed) on = 1, off = 0
set promiscuous 0
# Set denial message
set denialmsg "Denied"

# set datagroups
set ps_hosts_dg "ps_hosts_$env"
set ps_sites_dg "ps_sites_$env"
set ps_logins_dg "ps_logins_$env"
set ps_portals_dg "ps_portals_$env"
set ps_nodes_dg "ps_nodes_$env"
set ps_pagelets_dg "ps_pagelets_$env"
set ps_root_urls_dg "ps_root_urls_$env"
```

```
set ps_c_components_dg "ps_c_components_$env"
set ps_c_scripts_dg "ps_c_scripts_$env"
set ps_c_homepages_dg "ps_c_homepages_$env"
set ps_c_worklists_dg "ps_c_worklists_$env"
set host [HTTP::host]
set uri [HTTP::uri]
set servlet [getfield $uri "/" 2]

# Check for valid host
if { [class match $host equals $ps_hosts_dg] }{
    # redirect / to signin page
    if { [HTTP::uri] equals "/" } {
        if { $logging > 0 } {log local0. "Redirect -
https://$host/psp/$portal_site/?cmd=login"}
        HTTP::redirect "https://$host/psp/$portal_site/?cmd=login"
        return
    }
    #redirect from EMPLOYEE portal to EXTERNAL portal
    if { [HTTP::uri] contains "/EMPLOYEE/"}{
        set uri [string map  {"/EMPLOYEE/" "/EXTERNAL/"} [HTTP::uri]]
        HTTP::redirect "https://$host$uri"
        if { $logging > 0 } {log local0. "Redirect - https://$host$uri"}
        return
    }

# Allow all Cache, Images, JavaScript servlets
    if { ($servlet equals "cs") }{
      set site [getfield $uri "/" 3]
      set portal [getfield $uri "/" 4]
      set file [getfield $uri "/" 5]
      if { $logging > 0 } {log local0. "Allowed - Static Content - servlet:
$servlet site: $site portal: $portal file: $file"}
      return
# System URLs
    } elseif { [class match $servlet equals $ps_sites_dg] }{
      set portal [getfield $uri "/" 3]
      set content [getfield $uri "/" 4]
      # Check for Login URL strings
      if { [class match $portal equals $ps_logins_dg] }{
        if { $logging > 0 } {log local0. "Allowed - System URL - servlet:
$servlet cmd: $portal"}
        return
      } elseif { [class match $portal equals $ps_portals_dg] }{
        if { $logging > 0 } {log local0. "Allowed - System URL - servlet:
$servlet portal: $portal content: $content"}
        return
      } elseif { $portal equals "images" }{
        if { $logging > 0 } {log local0. "Allowed - System URL - servlet:
$servlet portal: $portal file: $content"}
        return
      } else {
```

```
        if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
        if { $logging > 0 } {log local0. "Denied - System URL - servlet:
$servlet portal: $portal content: $content"}
        return
      }
# URL format for PeopleSoft Pure Internet Architecture content types
    } elseif { ($servlet equals "psp") }{
      set site [getfield $uri "/" 3]
      set portal [getfield $uri "/" 4]
      set node [getfield $uri "/" 5]
      set type [getfield $uri "/" 6]
      set content [getfield $uri "/" 7]
      if { [class match $site starts_with $ps_sites_dg] }{
        # Login URL
        if { [class match $portal equals $ps_logins_dg] }{
          if { $logging > 0 } {log local0. "Allowed - PIA Content - servlet:
$servlet site: $site cmd: $portal"}
        } elseif { [class match $portal equals $ps_portals_dg] }{

          if { [class match $node equals $ps_nodes_dg] }{
            if { $type equals "s" }{
              if { [class match $content starts_with $ps_c_scripts_dg] }{
                if { $logging > 0 } {log local0. "Allowed - PIA Content -
script - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                return
              } else {
                if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
                if { $logging > 0 } {log local0. "Denied - PIA Content -
script - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                return
              }
            } elseif { $type equals "c" } {
              if { [class match $content starts_with $ps_c_components_dg] }{
                if { $logging > 0 } {log local0. "Allowed - PIA Content -
component - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                return
              } else {
                if { $logging > 0 } {log local0. "Denied - PIA Content -
component - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
                return
              }
            } elseif { $type equals "h" } {
              if { [class match $content contains $ps_c_homepages_dg] }{
```

```
                if { $logging > 0 } {log local0. "Allowed - PIA Content -
homepage - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                return
              } else {
                if { $logging > 0 } {log local0. "Denied - PIA Content -
servlet: $servlet site: $site portal: $portal node: $node type: $type
content: $content"}
                if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
                return
              }
            } elseif { $type contains "w" } {
              if { [class match $content starts_with $ps_c_worklists_dg] }{
                if { $logging > 0 } {log local0. "Allowed - PIA Content -
worklist - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                return
              } else {
                if { $logging > 0 } {log local0. "Denied - PIA Content -
servlet: $servlet site: $site portal: $portal node: $node type: $type
content: $content"}
                if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
                return
              }
            } elseif { $type contains "pageletname=" } {
              if { [class match $type contains $ps_pagelets_dg] }{
                if { $logging > 0 } {log local0. "Allowed - PIA Pagelet
Content - servlet: $servlet site: $site portal: $portal node: $node cmd:
$type"}
                return
              } else {
                if { $logging > 0 } {log local0. "Denied - PIA Pagelet
Content - servlet: $servlet site: $site portal: $portal node: $node cmd:
$type"}
                if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
                return
              }
            } elseif { [class match $type equals $ps_logins_dg] } {
              if { $logging > 0 } {log local0. "Allowed - PIA Content -
servlet: $servlet site: $site portal: $portal node: $node content: $type"}
              return
            } else {
              if { $logging > 0 } {log local0. "Denied - PIA Content -
Failed Content Type - servlet: $servlet site: $site portal: $portal node:
$node type: $type content: $content"}
              if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
              return
```

```
            }
          } else {
            if { $logging > 0 } {log local0. "Denied - PIA Content - Failed
Node - servlet: $servlet site: $site portal: $portal node: $node type: $type
content: $content"}
            if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
            return
          }
        } else {
          if { $logging > 0 } {log local0. "Denied - PIA Content - Failed
Portal - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
          if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
          return
        }
      } else {
        if { $logging > 0 } {log local0. "Denied - PIA Content - Failed Site
- servlet: $servlet site: $site portal: $portal node: $node type: $type
content: $content"}
        if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
        return
      }
# URL format for unwrapped PeopleSoft Pure Internet Architecture content
    } elseif { ($servlet equals "psc") }{
      set site [getfield $uri "/" 3]
      set portal [getfield $uri "/" 4]
      set node [getfield $uri "/" 5]
      set type [getfield $uri "/" 6]
      set content [getfield $uri "/" 7]
      if { [class match $site starts_with $ps_sites_dg] }{
        # Login URL
        if { $portal equals "view" }{
          set path [getfield $uri "/" 5]
          set file [getfield $uri "/" 6]
          if { $logging > 0 } {log local0. "Allowed - unwrapped PIA Content
- servlet: $servlet site: $site path: $path file: $file"}
          return
        } elseif { $portal equals "viewredirect" }{
          set path [getfield $uri "/" 5]
          set file [getfield $uri "/" 6]
          if { $logging > 0 } {log local0. "Allowed - unwrapped PIA Content
- servlet: $servlet site: $site path: $path file: $file"}
          return
        } elseif { [class match $portal equals $ps_logins_dg] }{
          if { $logging > 0 } {log local0. "Allowed - unwrapped PIA Content
- servlet: $servlet site: $site cmd: $portal"}
          return
        } elseif { [class match $portal equals $ps_portals_dg] }{
```

```
          if { [class match $node equals $ps_nodes_dg] }{
            if { $type equals "s" }{
              if { [class match $content starts_with $ps_c_scripts_dg] }{
                if { $logging > 0 } {log local0. "Allowed - unwrapped PIA
Content - script - servlet: $servlet site: $site portal: $portal node: $node
type: $type content: $content"}
                return
              } else {
                if { $logging > 0 } {log local0. "Denied - unwrapped PIA
Content - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
                return
              }
            } elseif { $type equals "c" } {
              if { [class match $content starts_with $ps_c_components_dg] }{
                if { $logging > 0 } {log local0. "Allowed - unwrapped PIA
Content - component - servlet: $servlet site: $site portal: $portal node:
$node type: $type content: $content"}
                return
              } else {
                if { $logging > 0 } {log local0. "Denied - unwrapped PIA
Content - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
                return
              }
            } elseif { $type contains "w" } {
              if { [class match $content starts_with $ps_c_worklists_dg] }{
                if { $logging > 0 } {log local0. "Allowed - unwrapped PIA
Content - worklist - servlet: $servlet site: $site portal: $portal node:
$node type: $type content: $content"}
                return
              } else {
                if { $logging > 0 } {log local0. "Denied - unwrapped PIA
Content - servlet: $servlet site: $site portal: $portal node: $node type:
$type content: $content"}
                if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
                return
              }
            } elseif { [class match $type equals $ps_logins_dg] } {
              if { $logging > 0 } {log local0. "Allowed - PIA Content -
login - servlet: $servlet site: $site portal: $portal node: $node content:
$type"}
              return
            } else {
              if { $logging > 0 } {log local0. "Denied - unwrapped PIA
Content - Failed Content Type - servlet: $servlet site: $site portal:
```

```
$portal node: $node type: $type content: $content"}
              if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
              return
              }
          } else {
            if { $logging > 0 } {log local0. "Denied - unwrapped PIA Content
- Failed Node - servlet: $servlet site: $site portal: $portal node: $node
type: $type content: $content"}
            if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
            return
            }
        } else {
          if { $logging > 0 } {log local0. "Denied - unwrapped PIA Content -
Failed Portal - servlet: $servlet site: $site portal: $portal node: $node
type: $type content: $content"}
          if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
          return
        }
      } else {
        if { $logging > 0 } {log local0. "Denied - unwrapped PIA Content -
Failed Site - servlet: $servlet site: $site portal: $portal node: $node
type: $type content: $content"}
        if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
        return
      }
# Check for WebSocket URL
    } elseif { ($servlet equals "ws") }{
      set site [getfield $uri "/" 3]
      set cmd [getfield $uri "/" 4]
      if { $logging > 0 } {log local0. "Denied - WebSocket Content -
servlet: $servlet site: $site cmd: $cmd"}
      if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
      return
#Check for Root url strings
    } elseif { [class match $uri equals $ps_root_urls_dg] }{
      if { $logging > 0 } {log local0. "Allowed - Root URL Content - $uri"}
      return
# Unfiltered URL denial
    } else {
      if { $logging > 0 } {log local0. "$uri failed URL filter"}
      if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg"
noserver Content-Type "text/html" Connection Close}
      return
    }
# Uncaught Host string denial
  } else {
```

```
    if { $logging > 0 } {log local0. "Failed Host - host: $host uri: $uri"}
    if { $promiscuous < 1 } {HTTP::respond 200 content "$denialmsg" noserver
Content-Type "text/html" Connection Close}
    return
  }
}
```

iRules Logic

Our F5 Implementation utilizes custom if-else loops which example each part of the URI and denies any incoming request which does not satisfy every one of our conditions. The getfield command delimits each part of the URI and evaluates the contents against our data file groups which contain a list of acceptable strings. For instance, *the set portal [getfield $uri "/" 3]* code will parse the URI with "/" as a delimiter and set the third string as the variable portal.

```
      } elseif { [class match $servlet equals $ps_sites_dg] }{ ##System URLs
          set portal [getfield $uri "/" 3]
          set content [getfield $uri "/" 4]
          # Login URL
          if { [class match $portal equals $ps_logins_dg] }{
              if { $logging > 0 } {log local0. "Allowed - System URL -
servlet: $servlet cmd: $portal"}
              return
          } elseif { [class match $portal equals $ps_portals_dg] }{
              if { $logging > 0 } {log local0. "Allowed - System URL -
servlet: $servlet portal: $portal content: $content"}
              return
          } elseif { $portal equals "images" }{
              if { $logging > 0 } {log local0. "Allowed - System URL -
servlet: $servlet portal: $portal file: $content"}
              return
          } else {
              if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
              if { $logging > 0 } {log local0. "Denied - System URL -
servlet: $servlet portal: $portal content: $content"}
              return
          }
```

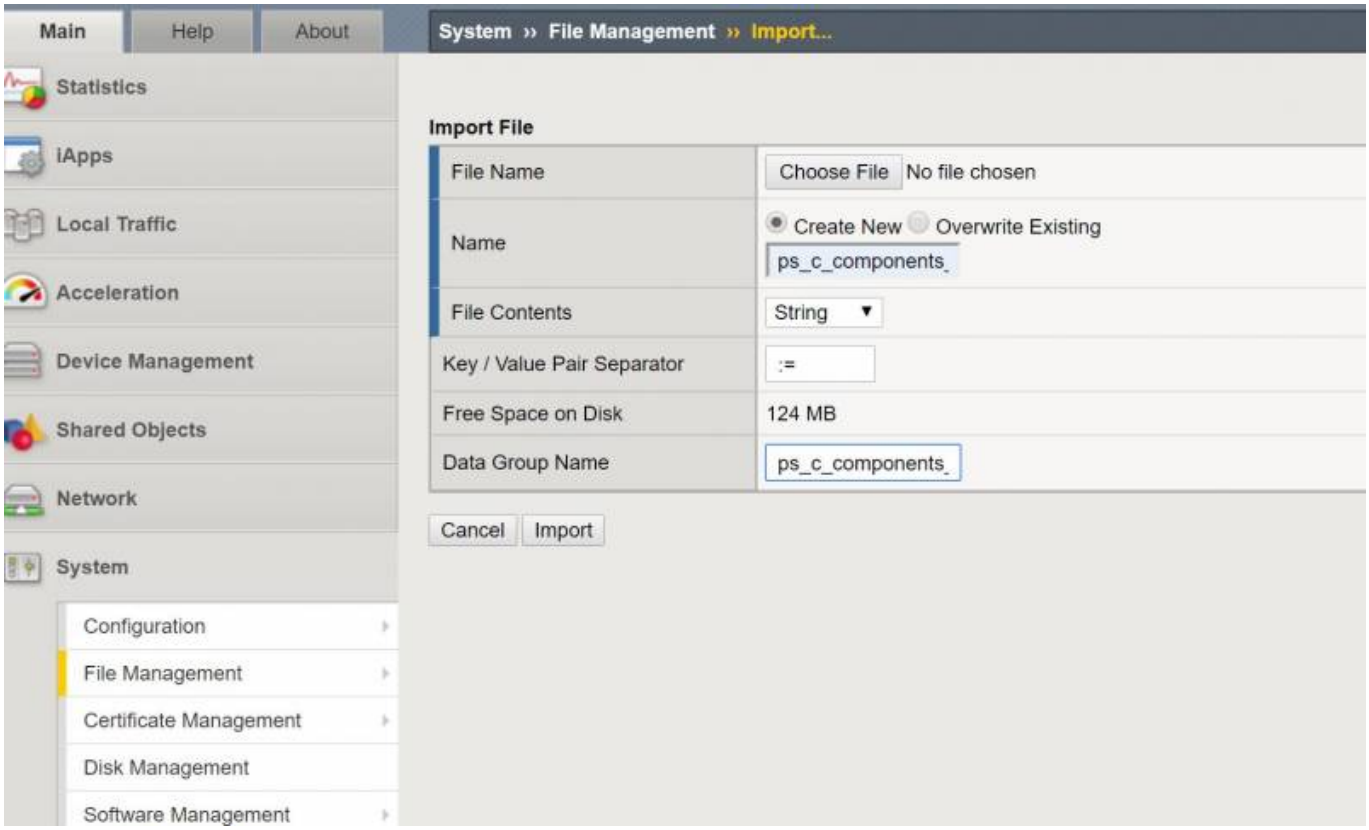
As you can see in this code snippet we are comparing the servlet part of the URI to our ps_sites data group. If the servelet matches one of the strings in the data group we will the examine the portal and content parts of the URI. We will then check the URI for a login string. If the login string is found we exit the loop and allow the request. Otherwise, we will examine the portal to see if it either matches a string in our data group, or contains the string "images". In all other cases where the HTTP request does not satisfy all of our requirements the server will return our denial message.

Customization

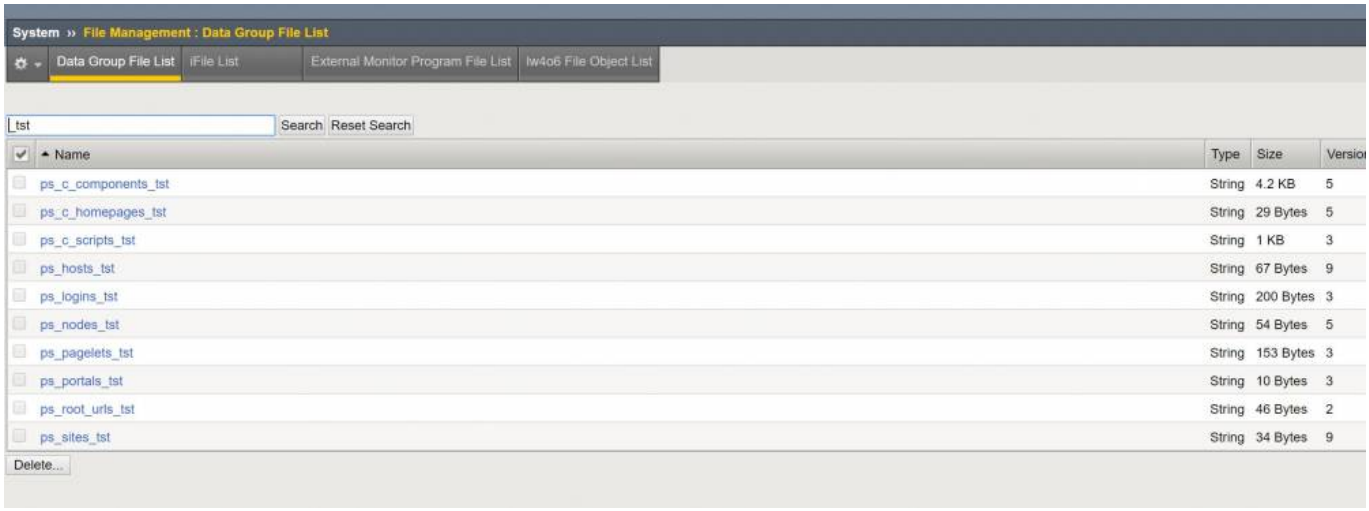
Our implementation utilizes data groups to make customization of your environment simple. Any changes to whitelisted nodes, sites, hosts, components, scripts, or homepages can be easily added or removed from a data group file which is comprised of a list of strings within double quotes. There is no need to modify the irules code.

Data Groups

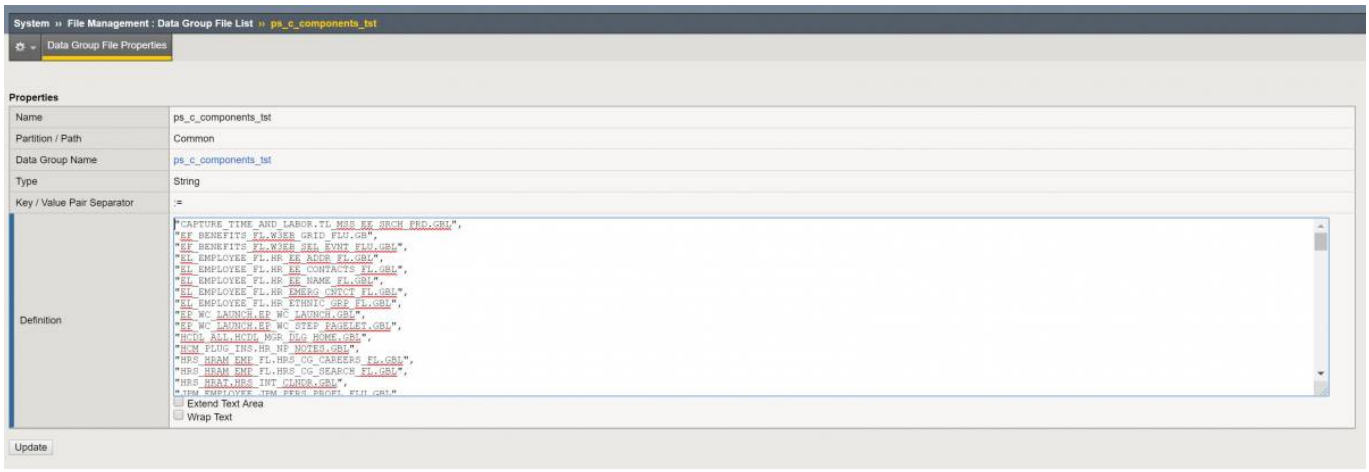
Data Groups can be added to F5 very simply. We can create new Data Groups using a text file, then add and remove strings as needed. To add a new data group navigate to System → File Management → Data Group File List → Import.



Editing data groups can be done from the Data Group File List Menu.



To edit the data group, click on one of the links in the list. You will be brought to a text editor. Here you may add or remove strings contained in "".



Note how we are setting data groups in the beginning of the iRules. Here we can set the environment at the top. Then, all the data groups that we set will have the correct suffix. The data groups should follow the naming convention shown in the iRules code. For example, the PeopleSoft Components Data Group would be names as follows: <application>_<URI part>_c_<env> ps_components_c_tst.

```
# set environment
set env "tst"

# set datagroups
set ps_hosts_dg "ps_hosts_$env"
set ps_sites_dg "ps_sites_$env"
set ps_logins_dg "ps_logins_$env"
set ps_portals_dg "ps_portals_$env"
set ps_nodes_dg "ps_nodes_$env"
set ps_pagelets_dg "ps_pagelets_$env"
set ps_root_urls_dg "ps_root_urls_$env"
set ps_c_components_dg "ps_c_components_$env"
set ps_c_scripts_dg "ps_c_scripts_$env"
set ps_c_homepages_dg "ps_c_homepages_$env"
set ps_c_worklists_dg "ps_c_worklists_$env"
```

iRules Speed

Every time you save your configuration all of your iRules are pre-compiled into what is referred to as “byte code”. Byte code is mostly compiled and has the vast majority of the interpreter tasks already performed, so that the Traffic Management Microkernel can directly interpret the remaining object. This makes for far higher performance and as such, increase scalability and the speed at which traffic is evaluated.

URL Redirects

Utilizing the following code, you can easily provide redirects to your root URL and/or redirect from the PeopleSoft portal to another portal of your choosing

```
# redirect root to signin page
    if { [HTTP::uri] equals "/" } {
        if { $logging > 0 } {log local0. "Redirect -
https://$host/psp/$portal_site/?cmd=login"}
        HTTP::redirect "https://$host/psp/$portal_site/?cmd=login"
        return
    }
# redirect from EMPLOYEE portal to EXTERNAL portal
    if { [HTTP::uri] contains "/EMPLOYEE/"}{
        set uri [string map  {"/EMPLOYEE/" "/EXTERNAL/"} [HTTP::uri]]
        HTTP::redirect "https://$host$uri"
        if { $logging > 0 } {log local0. "Redirect - https://$host$uri"}
        return
    }
```

Logging

Our implementation includes an advanced logging feature which records every part of the URL and whether it was Allowed or Denied. This allows the user to easily identify which part of the URL and/or URI was denied and be added or removed from the whitelist. Whitelists are managed by data group files where strings can be added or removed from a simple text editor. The core iRules code does not need to be changed. If you choose to disable logging you can do so in the iRules code by simply changing the logging value at the top to 0.

```
         if { $logging > 0 } {log local0. "Allowed - Static Content -
servlet: $servlet site: $site portal: $portal file: $file"}
         return
```

Notice the log local0. statement on the first line. Here we will log an allowed message and record the $servlet, $portal, and $file variables.

Promiscuous Mode

Our implementation includes a Promiscuous Mode which will allow all traffic through the F5, but will continue to log all allow and deny messages for the URL. This can be especially useful if you want to rapidly visit your allowed pages and then collect the components, scripts, pagelets, etc that you wish to whitelist.

```
          } else {
              if { $promiscuous < 1 } {HTTP::respond 200 content
"$denialmsg" noserver Content-Type "text/html" Connection Close}
```

Notice in the second line we check our $promiscuous variable to see if promiscuous mode is enabled. If promiscuous mode is enabled, we will disregard the denial statement. If promiscuous mode is disabled we will deny the HTTP request.

!!! Note Regarding iRules Updates !!!

Please note that it is always best practice to remove iRules from ANY Virtual Servers before making any updates. Updating iRules while attached to Virtual Servers may cause large amounts of latency on the F5 appliance and in some cases crash the F5 device entirely.

YOU HAVE BEEN WARNED!

AWS WAF Implementation

Our AWS WAF implementation utilizes regex string matching to whitelist components, scriptes, pagelets and homepages.