

```
schedule at 1 startup * lua emfs:/O365_WU
```

```
embedded file O365_WU <<EOF
```

```
-----## 設定値  ##-----
```

```
-- コマンドパラメーター --
```

```
-- 経路
```

```
gateway_default = "tunnel 1"          -- ★ デフォルト経路に使用するインターフェース
```

```
gateway_offload = "pp 1"  -- ★ Office 365、Windows Update、G Suite に使用するインターフェース
```

```
-- FQDN フィルターに使用するフィルターID のプレフィックス
```

```
filter_prefix = 2000000
```

```
-- Office 365 インターネットブレイクアウトの SYSLOG のプレフィックス
```

```
offload_prefix = "[O365 Offload]"
```

```
-- 確認間隔 (秒)
```

```
idle_time = 3600          -- ★
```

```
-- HTTPS リクエスト --
```

```
-- ホスト名
```

```
office_host = "endpoints.office.com"
```

```
-- パス
```

```
path_version = "version/worldwide?clientrequestid=%s&Format=CSV"
```

```
path_endpoint = "endpoints/worldwide?clientrequestid=%s&Format=CSV&NoIPv6=true"
```

```
-- url
```

```
url_template = "https://%s/%s"
```

```
-- リクエストテーブル
```

```
req_tbl = {
```

```
    url = "",
```

```
    method = "GET"
```

```
}
```

-- Windows Update の接続先 URL リスト(テーブル)

```
urls_wu = {"update.microsoft.com", "*.update.microsoft.com", "download.windowsupdate.com", "*.download.windowsupdate.com",
"download.microsoft.com", "*.download.microsoft.com", "windowsupdate.com", "*.windowsupdate.com", "ntservicepack.microsoft.com",
"login.live.com", "mp.microsoft.com", "*.mp.microsoft.com", "*.do.dsp.mp.microsoft.com", "*.dl.delivery.mp.microsoft.com",
"*.emdl.ws.microsoft.com"}
```

-- インターネットブレイクアウトしない場合は、下の行のコメント"--"を外す ★

```
--urls_wu = {}
```

-- G Suite の接続先 URL リスト(テーブル)

```
urls_gs = {"accounts.google.com", "www.googleapis.com", "oauth2.googleapis.com", "*.googleapis.com", "www.google.com", "apps-
apis.google.com", "accounts.youtube.com", "fonts.gstatic.com", "ssl.gstatic.com", "www.gstatic.com", "classroom.googleapis.com",
"people.googleapis.com", "sheets.googleapis.com", "slides.googleapis.com", "gsuite.google.com", "mail.google.com", "calendar.google.com",
"chat.google.com", "meet.google.com", "drive.google.com", "cloud.google.com", "docs.google.com", "sites.google.com",
"developers.google.com", "keep.google.com", "jamboard.google.com", "admin.google.com", "ediscovery.google.com"}
```

-- インターネットブレイクアウトしない場合は、下の行のコメント"--"を外す ★

```
--urls_gs = {}
```

-- G Suite の接続先 URL リスト(テーブル)

```
urls_zoom = {"zoom.us", "*.zoom.us"}
```

-- インターネットブレイクアウトしない場合は、下の行のコメント"--"を外す ★

```
--urls_zoom = {}
```

-- 出力する SYSLOG のレベル (info, debug, notice)

```
log_level = "debug" -- ★
```

-- メッセージ --

```
msg_fail_ver = "Failed to get version from server"
```

```
msg_fail_server_http = "Failed to data transmission to server, HTTP STATUS: %d"
```

```
msg_fail_server_err = "Failed to data transmission to server, err: %s"
```

```
msg_fail_server = "Failed to data transmission to server"
```

```
msg_new_ver = "New version of service instance endpoints has been detected"
```

```
msg_cancel = "Service instance endpoints checking is canceled"
```

```
msg_update = "Service instance endpoints have been successfully updated"
```

-----## 設定値ここまで ##-----

```

-----

-- ログメッセージを SYSLOG に出力する関数

-- [引数]

--     msg_fmt ... メッセージフォーマット(書式指定文字列)

--     #2 以降 ... 書式指定子に入れる値

-- [戻り値]

--     なし

-----

function write_to_log(msg_fmt, ...)

    local log_msg

    log_msg = string.format(msg_fmt, ...)

    log_msg = string.format("%s %s", offload_prefix, log_msg)

    rt.syslog(log_level, log_msg)

end

-----

-- UUID を生成する関数

-- [引数]

--     なし

-- [戻り値]

--     UUID(文字列)

-----

function generate_uuid()

    local template = "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx"

    math.randomseed(os.time())

    return string.gsub(template, "[xy]", function (c)

        local v = (c == "x") and math.random(0, 0xf) or math.random(8, 0xb)

        return string.format("%x", v)

    end)

end

end

```

```
-----  
-- Office サーバーからバージョンを取得する関数
```

```
-- [引数]
```

```
--      uuid ... UUID(文字列)
```

```
-- [戻り値]
```

```
--      バージョン(数値)  
-----
```

```
function get_version_from_server(uuid)
```

```
    local path, i, v, res_tbl
```

```
    local lines = {}
```

```
    local line = {}
```

```
    path = string.format(path_version, uuid)
```

```
    req_tbl.url = string.format(url_template, office_host, path)
```

```
    res_tbl = rt.httprequest(req_tbl)
```

```
    local cur_ver = 0
```

```
    local num = 0
```

```
    if (res_tbl.rtn1) and (res_tbl.code) and (res_tbl.code == 200) then
```

```
        -- 送信成功
```

```
        lines = { string.split(res_tbl.body, /%s+/) }
```

```
        for i, v in ipairs(lines) do
```

```
            line = { string.split(v, ",") }
```

```
            if (i == 1) then
```

```
                for i, v in ipairs(line) do
```

```
                    if (v == "latest") then
```

```
                        num = i
```

```
                    end
```

```
                end
```

```
            else
```

```
                if (num > 0) and (line[num] ~= nil) and (string.len(line[num]) > 0) then
```

```
                    cur_ver = tonumber(line[num])
```

```
                    if (cur_ver == nil) then
```

```
                        cur_ver = 0
```

```
                    end
```

```
                end
```

```

                                break
                        end
                    end
                end
            if (cur_ver == 0) then
                write_to_log(msg_fail_ver)
            end
        else
            if (res_tbl.code) then
                write_to_log(msg_fail_server_http, res_tbl.code)
            elseif (res_tbl.err) then
                write_to_log(msg_fail_server_err, res_tbl.err)
            else
                write_to_log(msg_fail_server)
            end
        end
    end

    return cur_ver
end

```

```

-----
-- Office サーバーからエンドポイントを取得する関数
-- [引数]
--     uuid ... UUID(文字列)
-- [戻り値]
--     Office サーバーから受信したレスポンスデータ(文字列) / nil
-----

function get_endpoints_from_server(uuid)
    local res_tbl, path

    path = string.format(path_endpoint, uuid)
    req_tbl.url = string.format(url_template, office_host, path)
    res_tbl = rt.httprequest(req_tbl)

    if (res_tbl.rtn1) and (res_tbl.code) and (res_tbl.code == 200) then
        -- 送信成功
        return res_tbl.body
    end
end

```

```

else

    if (res_tbl.code) then

        write_to_log(msg_fail_server_http, res_tbl.code)

    elseif (res_tbl.err) then

        write_to_log(msg_fail_server_err, res_tbl.err)

    else

        write_to_log(msg_fail_server)

    end

end

return nil

end

-----

-- CSV 形式の 1 行をテーブルに変換する関数
-- [引数]
--     line ... CSV 形式の 1 行(文字列)
-- [戻り値]
--     テーブルに変換したもの(テーブル)
-----

function ParseCSVLine(line)

    local res = {}

    local pos = 1

    local sep = ','

    while (true) do

        local c = string.sub(line,pos,pos)

        if (c == "") then break end

        if (c == '"') then

            -- quoted value (ignore separator within)

            local txt = ""

            repeat

                local startp,endp = string.find(line,'^%b"',pos)

                txt = txt..string.sub(line,startp+1,endp-1)

                pos = endp + 1

                c = string.sub(line,pos,pos)

                if (c == '"') then txt = txt..'"' end

            until (c == ',')

            res[#res+1] = txt

            pos = pos + 1

        else

            res[#res+1] = c

            pos = pos + 1

        end

    end

    return res

end

```

```

-- check first char AFTER quoted string, if it is another
-- quoted string without separator, then append it
-- this is the way to "escape" the quote char in a quote. example:
-- value1,"blub""blip""boing",value3 will result in blub"blip"boing for the

middle

until (c ~= '"')
table.insert(res,txt)
assert(c == sep or c == '"')
pos = pos + 1
else
-- no quotes used, just look for the first separator
local startp, endp = string.find(line, sep, pos)
if (startp) then
table.insert(res, string.sub(line, pos, startp-1))
pos = endp + 1
else
-- no separator found -> use rest of string and terminate
table.insert(res, string.sub(line, pos))
break
end
end
end

return res
end

-----
-- レスポンスデータから URL リストを取得する関数
-- [引数]
-- body ... Office サーバーから受信したレスポンスデータ(文字列)
-- [戻り値]
-- URL のリスト(テーブル)
-----

function get_urls(body)
local lines, line
local i, v

```

```

local url_num = 0

local cat_num = 0

local ip_num = 0

local urls_list = {}

if (body ~= nil) then

    lines = { string.split(body, /%v+/) }

    for i, v in ipairs(lines) do

        line = ParseCSVLine(v)

        -- 1 行目のパラメーター名から、urls と category の位置を取得

        if (i == 1) then

            for i, v in ipairs(line) do

                if (v == "urls") then

                    url_num = i

                elseif (v == "category") then

                    cat_num = i

                elseif (v == "ips") then

                    ip_num = i

                end

            end

            if (url_num == 0) or (cat_num == 0) or (ip_num == 0) then

                break

            end

        else

            if (line[cat_num] == "Allow") or (line[cat_num] == "Optimize") then

                -- urls を保存

                if (line[url_num] ~= nil) and (string.len(line[url_num]) > 0) then

                    table.insert(urls_list, line[url_num])

                elseif (line[ip_num] ~= nil) and (string.len(line[ip_num]) > 0) then

                    table.insert(urls_list, line[ip_num])

                end

            end

        end

    end

end

end

```



```

        return urls_list
    end

-----

-- URL リストから重複を削除、ソートする関数
-- [引数]
--     urls_list ... URL のリスト重複あり(テーブル)
-- [戻り値]
--     URL のリスト重複なし(テーブル)
-----

function del_duplicate_urls(urls_list)

    local url_splited = {}

    local flist = {}

    local flist_temp = {}

    local urls, url

    local k, v

    local str_urls = ""

    for urls in each(urls_list) do

        url_splited = { string.split(urls, ",") }

        for url in each(url_splited) do

            flist_temp[url] = 1

        end

    end

    for k, v in pairs(flist_temp) do

        table.insert(flist, k)

    end

    table.sort(flist)

    return flist

end

-----

-- URL のリストを一定の条件で連結する関数

```

```
-- [引数]

--      urls_list ... URL のリスト (テーブル)

-- [戻り値]

--      一定の条件で連結した URL のリスト (テーブル)

-- [条件]

--      連結した文字列の長さが指定数を超えたところで、1 つの要素とする
```

```
function concatenate_urls(urls_list)

    local url

    local str_urls = ""

    local flist = {}

    local urls_len = 256 -- 文字列の長さの閾値(0～)

    for url in each(urls_list) do

        if (str_urls == "") then

            str_urls = url

        else

            str_urls = str_urls .. "," .. url

        end

        if (string.len(str_urls) > urls_len) then

            table.insert(flist, str_urls)

            str_urls = ""

        end

    end

    if (str_urls ~= "") then

        table.insert(flist, str_urls)

    end

    return flist

end
```

```
-- FQDN フィルターを削除する関数

-- [引数]
```

```

--      num ... 削除するフィルター数(数値)
-- [戻り値]
--      なし
-----

function exec_no_filter_cmd(num)

    local cmd, i

    for i = 0, (num - 1) do

        cmd = string.format("no ip filter %d", filter_prefix + i)

        rtn, str = rt.command(cmd)

        if (rtn == false) then

            if (str) then

                write_to_log(str)

            end

        end

    end

end

end

```

```

-----

-- 2つのリストを結合する関数
-- [引数]
--      table_1 ... 1 つめのリスト(テーブル)
--      table_2 ... 2 つめのリスト(テーブル)
-- [戻り値]
--      結合した新しいリスト(テーブル)
-----

```

```

function joint_tables(table_1, table_2)

    local table_new = {}

    for v in each(table_1) do

        table.insert(table_new, v)

    end

    for v in each(table_2) do

        table.insert(table_new, v)

    end

end

```

```

        end

        return table_new
    end

-----

-- FQDN フィルターを設定する関数
-- [引数]
--     urls_list ... URL のリスト(テーブル)
-- [戻り値]
--     フィルター番号のリスト(テーブル)
-----

function exec_filter_cmd(urls_list)

    local v, cmd, rtn, str

    local i = 0;

    local flist = {}

    for v in each(urls_list) do

        cmd = string.format("ip filter %d pass * %s", filter_prefix + i, v)

        rtn, str = rt.command(cmd)

        if (rtn == false) then

            if (str) then

                write_to_log(str)

            end

            break

        else

            table.insert(flist, tostring(filter_prefix + i))

            i = i + 1;

        end

    end

    return flist

end

-----

-- ip route コマンドを設定する関数

```

```
-- [引数]
--      flist ... フィルター番号のリスト(テーブル)
```

```
-- [戻り値]
```

```
--      なし
```

```
-----
function exec_ip_route_cmd(flist)
```

```
    local cmd, v, rtn, str
```

```
    local buf = ""
```

```
    if (#flist == 0) then
```

```
        return
```

```
    end
```

```
    for v in each(flist) do
```

```
        buf = buf .. " " .. v
```

```
    end
```

```
    cmd = string.format("ip route default gateway %s filter %s gateway %s", gateway_offload, buf, gateway_default)
```

```
    rtn, str = rt.command(cmd)
```

```
    if (rtn == false) then
```

```
        if (str) then
```

```
            write_to_log(str)
```

```
        end
```

```
    end
```

```
end
```

```
-----
-- メインルーチン
```

```
--
```

```
local buf, cur_ver
```

```
local urls
```

```
local filters = {}
```

```
local pre_ver = 0
```

```
urls = joint_tables(urls_zoom, urls)
```

```

-- URL のリストをフィルター単位に分割する

urls = concatenate_urls(urls)

-- FQDN フィルターを削除

exec_no_filter_cmd(#filters)

-- FQDN フィルターを設定

filters = exec_filter_cmd(urls)

if (filters ~= nil) then
    -- 静的経路を設定
    exec_ip_route_cmd(filters)

    -- コンフィグを保存
    rt.command("save")

    err = 0
end

end

end

end

end

if (cur_ver == 0) or (err == 1) then
    write_to_log(msg_cancel)

elseif (cur_ver > pre_ver) then
    pre_ver = cur_ver
    write_to_log(msg_update)
end

end

rt.sleep(idle_time)

end

EOF

lua emfs:/O365_WU

```