Dr.WEB®
с 1992 года

Расширение функционала Dr.Web для почтовых серверов Unix
Обработка спам-сообщений

Расширение функционала Dr.Web для почтовых серверов Unix

Обработка сообщений

Общая информация

Обработка сообщений электронной почты, поступивших на проверку от MTA через интерфейсы Milter, Spamd и Rspamd (в режиме фильтра), в Dr.Web для почтовых серверов Unix производится путем вызова специальной процедуры обработки (hook), написанной на языке Lua и заданной по умолчанию, что позволяет легко расширять функционал данного продукта.

Интерпретатор программ на языке Lua версии 5.3.4 поставляется совместно с продуктом.

В ходе работы этой процедуры по результатам анализа всей доступной информации о сообщении (отправитель, получатель, внутренняя структура, значения заголовков, балльная оценка спама) принимается решение отвергнуть или пропустить сообщение.

- Для интерфейса Milter процедура обработки возвращает действие («пропустить», «отвергнуть», «вернуть ошибку отправителю» и т. п.), которое MTA следует применить к сообщению. Для случая «пропустить» в рамках процедуры проверки сообщение может также подвергнуться таким изменениям, как добавление заголовков или их модификация, помещение вредоносных частей сообщения в архив, защищенный паролем.
- Для интерфейсов Spamd и Rspamd, не предполагающих модификацию проверяемого сообщения, в MTA возвращается вердикт в виде присвоенной сообщению «оценки спама» и порога для признания его спамом (чтобы сообщение электронной почты было отвергнуто MTA, оценка должна превышать порог).

Кроме оценки, в MTA возвращается также текстовый вердикт (report или action, в зависимости от протокола), который может быть проанализирован в настройках MTA.

Гибкость языка Lua и большой набор сведений о сообщении, доступных из процедуры обработки, позволяют реализовать не только типовые проверки сообщений на спам с получением балльной оценки от компонента Dr.Web ASE, поиск вложенных угроз или вредоносных URL, но и реализовать проверку произвольных условий с выработкой необходимых вердиктов для обработки проверенного сообщения сервером электронной почты.

Внимание! После внесения изменений в настройки следует перезапустить Dr.Web для почтовых серверов UNIX, используя команду **drweb-ctl** reload либо перезапустив демон управления конфигурацией Dr.Web ConfigD командой **service** drweb-configd restart.

Процедура обработки спам-сообщений, полученных через интерфейс Spamd, задается в секции [MailD] объединенного конфигурационного файла продукта Dr.Web для почтовых серверов UNIX параметром **SpamdReportHook**. Данному параметру в качестве значения может быть назначен путь к файлу или функция, написанная на языке Lua.

Если указан недоступный файл, то при загрузке компонента будет выдана ошибка.

Разберем пример скрипта для обработки спам-сообщений, полученных через интерфейс *Spamd*

```
local dw = require "drweb"

function spamd_report_hook(ctx)
	local score = 0
	local report = ""

	-- Add 1000 to the score for each threat found in the message
	for threat, path in ctx.message.threats{category = {"known_virus", "virus_
modification", "unknown_virus", "adware", "dialer"}} do
		score = score + 1000
		report = report .. "Threat found: " .. threat.name .. "\n"
		dw.notice(threat.name .. " found in " .. (ctx.message.part_at(path).
name or path))
	end
	-- Add 100 to the score for each unwanted found URL in the message
	for url in ctx.message.urls{category = {"infection_source",
	"not_recommended", "owners_notice"}} do
		score = score + 100
		report = report .. "Url found: " .. url .. "\n"
		dw.notice("URL found: " .. url .. "(" .. url.categories[1] .. ")")
	end
	-- Add the spam score
	score = score + ctx.message.spam.score
	report = report .. "Spam score: " .. ctx.message.spam.score .. "\n"
	if ctx.message.spam.score >= 100 then
		dw.notice("Spam score: " .. ctx.message.spam.score)
	end
	-- Return the check result
	return {
		score = score,
		threshold = 100,
		report = report
	}
end
```

Файл должен содержать глобальную функцию, являющуюся точкой входа в модуль проверки сообщений.

Функция обработки должна соответствовать следующим соглашениям о вызове:

1. *Имя функции* — spamd_report_hook.
2. *Единственный аргумент* — таблица SpamdContext (предоставляет из функции доступ к информации об обрабатываемом сообщении).
3. *Единственное возвращаемое значение* — заполненная таблица SpamdReportResult. Возвращаемое значение определяет ответ по протоколу *Spamd*.

Простейшая реализация сценария может выглядеть так:

```
function spamd_report_hook(ctx)
return {
      score = 200,
      threshold = 100,
      report = "The message was recognized as spam"
}
end
```

Данный сценарий будет безусловно возвращать вердикт о том, что сообщение электронной почты следует признать спамом (оценка баллов спама — 200, порог признания спамом — 100, уведомление отправителю. Аргумент ctx — экземпляр таблицы SpamdContext, описанный в документации. Таблица предоставляет доступ к информации об обрабатываемом сообщении (его поля, структура, заголовки, тело, информация об отправителе и получателях, информация SMTP-сессии).

По умолчанию параметру **RspamdHook**, отвечающему за обработку сообщений электронной почты, полученных через интерфейс *Rspamd*, аналогично **предыдущему** назначен следующий сценарий:

```
local dw = require "drweb"

function rspamd_hook(ctx)
      local score = 0
      local symbols = {}

      -- Add 1000 to the score for each threat found in the message
      for threat, path in ctx.message.threats{category = {"known_virus", "virus_
modification", "unknown_virus", "adware", "dialer"}} do
            score = score + 1000
            table.insert(symbols, {name = threat.name, score = 1000})
            dw.notice(threat.name .. " found in " .. (ctx.message.part_at(path).
name or path))
end

-- Add 100 to the score for each unwanted URL found in the message
for url in ctx.message.urls{category = {"infection_source", "not_recommended",
"owners_notice"}} do
            score = score + 100
            table.insert(symbols, {name = "URL " .. url, score = 100})
            dw.notice("URL found: " .. url .. "(" .. url.categories[1] .. ")")
end

-- Add the spam score
      score = score + ctx.message.spam.score
      table.insert(symbols, {name = "Spam score", score = ctx.message.spam.score})
      if ctx.message.spam.score >= 100 then
            dw.notice("Spam score: " .. ctx.message.spam.score)
      end
```

```
-- Return the check result
      return {
            score = score,
            threshold = 100,
            symbols = symbols
}
end
```

Правила написания данного сценария аналогичны описанным выше, но имя функции — rspamd_hook, единственный аргумент — таблица RspamdContext, а единственное возвращаемое значение — заполненная таблица RspamdResult.

Для взаимодействия с Dr.Web для почтовых серверов UNIX в пространство Lua-программы могут быть импортированы следующие специфические модули:

- Drweb — модуль предоставляет функции для записи сообщений из Lua-программы в журнал компонента Dr.Web для почтовых серверов UNIX, запустившего программу на Lua, а также средства асинхронного запуска Lua-процедур;
- drweb.lookup — модуль, предоставляющий инструменты для запроса данных из внешних источников путем обращения к модулю Dr.Web LookupD;
- drweb.dnsxl — модуль, предоставляющий инструменты для проверки вхождения адресов узлов в черные списки DNSxL;
- drweb.regex — модуль, предоставляющий интерфейс сопоставления строк с регулярными выражениями;
- drweb.subprocess — модуль, предоставляющий интерфейс запуска внешних приложений (процессов).

Описание возможностей модуля приведено в документации по продукту.

Пример реализации модуля обработки почтового сообщения по протоколу Spamd

```
local drweb = require "drweb"
local dns = require "drweb.dnsxl"
local lookup = require "drweb.lookup"

-- Entry point to check email message sent to the Dr.Web MailD by Spamd protocol
function spamd_report_hook(ctx)
    -- If message type is not multipart, output headers and body
    if #ctx.message.part == 0 then
        drweb.notice("Message HEADERS:")
        local headers = ctx.message.header.field
        for i =1, #headers do
            drweb.notice(" -> " .. headers[i].name .. ": " .. headers[i].value)
        end
        drweb.notice("Message BODY:")
        drweb.notice(" -> " .. ctx.message.body.raw)
    -- Else disassemble it in parts
    else
        drweb.notice("Message parts:")
        for index, part in ipairs(ctx.message.part) do
```

```
            drweb.notice("Part " .. index .. " HEADERS:")
            local headers = ctx.message.header.field
            for i =1, #headers do
                drweb.notice(" -> " .. headers[i].name .. ": " .. headers[i].value)
            end
            drweb.notice("Part " .. index .. " BODY:")
            drweb.notice(" -> " .. part.body.raw)
        end
    end

    -- If the message contains an URL from any of specified categories, reject it
(return the score that exceed the threshold)
    if ctx.message.has_url{category = {"adult_content"}} then
        return {
        score = 200,
        threshold = 100,
        report = "The message contains unwanted URL(s)"
    }
    end

    -- Check the message for spam and reject it, if spam score is exceed 100
(return the score that exceed the threshold)
    if ctx.message.spam.score > 100 then
        return {
            score = ctx.message.spam.score,
            threshold = 100,
            report = "The message was recognized as spam"
        }
    end

    -- The hook function must return to MTA: spam score, spam threshold and string
report.
    return {
        score = ctx.message.spam.score,
        threshold = 100,
        report = "The message is clean"
    }
end
```

Данная реализация начисляет баллы оценки в случае выявления порнографического контента и возвращает как число начисленных баллов, так и порог срабатывания на спам.

Полный текст реализации выложен по адресу:

https://github.com/DoctorWebLtd/drweb-lua-examples/blob/master/maild/spamd/api_example.lua

Пример реализации модуля обработки почтового сообщения по протоколу Rspamd

```
local drweb = require "drweb"
local dns = require "drweb.dnsxl"
local lookup = require "drweb.lookup"

-- Entry point to check email message sent to the Dr.Web MailD by Rspamd protocol
function rspamd_hook(ctx)

  -- Now we can see the information we are interested in about the message.

    -- Output message data to log of Dr.Web MailD on level "notice"
    drweb.notice("SMTP HELO/EHLO: " .. ctx.helo)
    drweb.notice("SMTP MAIL FROM: " .. ctx.from)

    drweb.notice("Sender info:")
    drweb.notice(" -> hostname: " .. ctx.sender.hostname)
    drweb.notice(" -> ip: " .. ctx.sender.ip)

    -- Iterate throw array of recipients
    drweb.notice("Message rcpts:")
    for _, rcpt in ipairs(ctx.to) do
        drweb.notice(" -> " .. rcpt)
    end

    -- If message type is not multipart, output headers and body
    if #ctx.message.part == 0 then
        drweb.notice("Message HEADERS:")
        local headers = ctx.message.header.field
        for i =1, #headers do
            drweb.notice(" -> " .. headers[i].name .. ": " .. headers[i].value)
        end
        drweb.notice("Message BODY:")
        drweb.notice(" -> " .. ctx.message.body.raw)
    -- Else disassemble it in parts
    else
        drweb.notice("Message parts:")
        for index, part in ipairs(ctx.message.part) do
            drweb.notice("Part " .. index .. " HEADERS:")
            local headers = ctx.message.header.field
            for i =1, #headers do
                drweb.notice(" -> " .. headers[i].name .. ": " .. headers[i].value)
            end
            drweb.notice("Part " .. index .. " BODY:")
            drweb.notice(" -> " .. part.body.raw)
        end
    end

  -- Then we can check message for a legit consistence
```

```
    -- If the message contains an URL from any of specified categories, reject it
(return the score exceed the threshold)
    if ctx.message.has_url{category = {"adult_content", "social_networks"}} then
        return {
        score = 200,
        threshold = 100,
        symbols = {
                {
                    name = "The message contains unwanted URL(s)",
                    score = 200
                }
            }
        }
    end

    -- If the message contains any threats, reject it (return the score exceed the
threshold)
    if ctx.message.has_threat() then
        return {
        score = 900,
        threshold = 100,
        symbols = {
                {
                    name = "The message contains threat(s)",
                    score = 900
                }
            }
        }
    end

    -- Check the message for spam and reject it, if spam score is exceed 100
(return the score exceed the threshold)
    if ctx.message.spam.score > 100 then
        return {
            score = ctx.message.spam.score,
            threshold = 100,
            symbols = {
                {
                    name = "The message was recognized as spam",
                    score = ctx.message.spam.score
                }
            }
        }
    end

    -- The hook function must return report to SMTP server.
    return {
```

```
        score = ctx.message.spam.score,
        threshold = 100,
        symbols = {
                {
                    name = "The message is clean",
                    score = 0
                }
            }
        }
end
```

Данная реализация функционала существенно более интересна. Скрипт состроит из следующих частей:

1. Запись в лог информации о полученном сообщении. В данном примере записывается информация обо всех проверенных письмах, что позволяет строить систему отчетности о полученных сообщениях. Но путем использования начисленных баллов можно записывать в лог информацию только о спам-сообщениях — или собирать статистику об их отправителях и включать ее, например, в черные списки.
2. Далее в скрипте проводится анализ различных признаков спама — например, подозрительных ссылок, и в зависимости от этого начисляются баллы. Ничего не мешает также блокировать эти ссылки.
3. В конце письма заполняется структура, возвращаемая в результате работы скрипта.

Как видим, ничего сложного.

Полный текст реализации выложен по адресу:
https://github.com/DoctorWebLtd/drweb-lua-examples/blob/master/maild/rspamd/api_example.lua

Пример реализации модуля обработки почтового сообщения по протоколу Milter

Требования к сценарию

Файл должен содержать глобальную функцию, являющуюся точкой входа в модуль проверки сообщений (эту функцию Dr.Web MailD будет вызывать для обработки вновь поступившего сообщения). Функция обработки должна соответствовать следующим соглашениям о вызове:

1. *Имя функции* — milter_hook;
2. *Единственный аргумент* — таблица MilterContext (предоставляет из функции доступ к информации об обрабатываемом сообщении);
3. *Единственное возвращаемое значение* — заполненная таблица MilterResult.

Возвращаемое значение определяет вердикт относительно проверяемого сообщения: принять, отвергнуть, изменить или отбросить, а также какие действия (возможно) с ним следует совершить в случае его принятия.

Пример корректного определения сценария, который будет безусловно возвращать в Dr.Web MailD вердикт Accept (принять) для всех сообщений, поступивших на проверку через интерфейс *Milter* (здесь и далее — аргумент ctx — экземпляр таблицы *MilterContext*):

```
function milter_hook(ctx)
return {action = "accept"}
end
```

Пример внесения в сообщение электронной почты следующих изменений при обработке:

- добавление названий обнаруженных в сообщении угроз в виде значений заголовка XFound;
- добавление к теме сообщения (значение заголовка Subject) префикса "[SPAM]", если его оценка спама составит более 100 баллов;

```
function milter_hook (ctx)
-- Добавить в заголовок имена найденных угроз
for threat in ctx.message.threats() do
ctx.modifier.add_header_field("X-Found", threat.name)
end
-- Изменить значение заголовка Subject, если сообщение
-- набрало более 100 баллов оценки спама
if ctx.message.spam.score > 100 then
local old_value = ctx.message.header.value("Subject") or ""
local new_value = "[SPAM] " .. old_value
ctx.modifier.change_header_field("Subject", new_value)
end
-- Передать сообщение получателю, применив отложенные изменения
return {
action = "accept",
modifications = ctx.modifier.modifications()
}
end
```

Пример перепаковки сообщения при обработке:

- перемещение обнаруженных в сообщении угроз в защищенный архив;
- перемещение сообщения целиком в защищенный архив, если его оценка спама составит более 100 баллов;

```
function milter_hook(ctx)
ctx.modifier.repack_password = "xxx"
ctx.modifier.repack_message = ""
-- Поместить в защищенный паролем архив все части
-- сообщения, в которых найдены угрозы
for threat, path in ctx.message.threats() do
                ctx.modifier.repack(path)
                local msg = " Threat found: " .. threat.name
                ctx.modifier.repack_message = ctx.modifier.repack_message .. msg
end
-- Перепаковать сообщение целиком, если оно набрало
-- более 100 баллов оценки спама
if ctx.message.spam.score > 100 then
                ctx.modifier.repack()
                local msg = " Spam score: " .. ctx.message.spam.score
                ctx.modifier.repack_message = ctx.modifier.repack_message .. msg
end
```

```
-- Передать сообщение получателю, применив отложенные изменения
-- Обратите внимание, что если таблица модификаций не указана,
-- она будет автоматически возвращена
return {action = "accept"}
end
```

Архив, содержащий все нежелательные части, изъятые из сообщения, будет отправлен получателю в виде вложения в новое сформированное сообщение (сообщение, подвергнутое перепаковке). Архив будет защищен паролем, который в данном случае принудительно будет установлен в строку "xxx".

Более интересный скрипт

```
-- Provided auxiliary modules
local drweb = require "drweb"
local dns = require "drweb.dnsxl"
local lookup = require "drweb.lookup"

-- Entry point to check email message sent to the Dr.Web MailD by Milter protocol
function milter_hook(ctx)
    -- Output message data to log of Dr.Web MailD on level "notice"
    drweb.notice("SMTP HELO/EHLO: " .. ctx.helo)
    drweb.notice("SMTP MAIL FROM: " .. ctx.from)

    drweb.notice("Sender info:")
    drweb.notice(" -> hostname: " .. ctx.sender.hostname)
    drweb.notice(" -> family: " .. ctx.sender.family)
    drweb.notice(" -> port: " .. ctx.sender.port)
    drweb.notice(" -> ip: " .. ctx.sender.ip)

    -- Iterate through array of recipients
    drweb.notice("Message rcpts:")
    for _, rcpt in ipairs(ctx.to) do
        drweb.notice(" -> " .. rcpt)
    end

    -- If message type is not multipart, output headers and body
    if #ctx.message.part == 0 then
        drweb.notice("Message HEADERS:")
        local headers = ctx.message.header.field
        for i =1, #headers do
            drweb.notice(" -> " .. headers[i].name .. ": " .. headers[i].value)
        end
        drweb.notice("Message BODY:")
        drweb.notice(" -> " .. ctx.message.body.raw)
    -- Else disassemble it in parts
    else
```

```
        drweb.notice("Message parts:")
        drweb.notice("Part 0 HEADERS:")
        local headers = ctx.message.header.field
        for i =1, #headers do
            drweb.notice(" -> " .. headers[i].name .. ": " .. headers[i].value)
        end
        for index, part in ipairs(ctx.message.part) do
            drweb.notice("Part " .. index .. " HEADERS:")
            local headers = part.header.field
            for i =1, #headers do
                drweb.notice(" -> " .. headers[i].name .. ": " .. headers[i].value)
            end
            drweb.notice("Part " .. index .. " BODY:")
            drweb.notice(" -> " .. part.body.raw)
        end
    end

  -- Then we can check message for a legit consistence

    -- If the message contains an URL from any of specified categories, reject it
    if ctx.message.has_url{category = {"adult_content", "social_networks"}} then
        return {action = "reject", message = "Detected url with bad content!"}
    end

    -- If attachs extensions matchs rar|zip|7z accept it, else reject
    for file, path in ctx.message.attachments{name_re_not = [[.*\.(rar|zip|7z)]]} do
        return {action = "reject", message = "Only archieve attaches
allowed(zip|rar|7z)" }
    end

  -- Then we can analyze and modify ('repack') the message

    -- Set the modifier variable containing the functions for implementing
the modifications
    local modifier = ctx.modifier
    -- Set the password for protected archive containing malicious parts of the
message
    modifier.repack_password = "qwerty"
    -- Set the text to be added to repacked message
    modifier.repack_message = ""

    -- Place all parts containing threats into the password-protected archive
    for threat, path in ctx.message.threats() do
        modifier.repack(path)
        local msg = " Threat found: " .. threat.name
        modifier.repack_message = modifier.repack_message .. msg
    end
```

```
    -- Check the message for spam and modify it, if spam score exceeds 100
    if ctx.message.spam.score > 100 then
        -- Modify value of Subject header
        local old_value = ctx.message.header.value("Subject") or ""
        local new_value = "[SPAM] " .. old_value
        -- Plan to set new value for Subject header
        modifier.change_header_field("Subject", new_value)
        -- Plan to add new header with spam score
        modifier.add_header_field("X-Spam-Score", ctx.message.spam.score)
        modifier.repack_message = "The message was recognized as spam"
        modifier.repack()
    end

  -- The hook function must return response to MTA.
  -- If the response is 'accept' and there are scheduled modifications,
  -- the hook function should return them in order to they are applied.
    return {action = "accept"}

    -- Available responses are:
    -- return {action = "accept"}
    -- return {action = "reject"}
    -- return {action = "discard"}
    -- return {action = "tempfail"}
    -- return {action = "replycode", code = "450", text = "response: Are you
serious?"}
End
```

Скрипт состроит из следующих частей:

1. Запись в лог информации об отправителе сообщения и его получателях. Также в лог сохраняются заголовки письма и его тело. В том числе когда письмо состоит из нескольких частей. В данном примере записывается информация обо всех проверенных письмах, что позволяет строить систему отчетности о полученных сообщениях. Но путем использования начисленных баллов можно записывать в лог информацию только о спам-сообщениях — или собирать статистику об их отправителях и включать ее, например, в черные списки.
2. Далее в скрипте в случае наличия в письме подозрительных ссылок или заархивированных вложений определенных форматов определяется действие на письмо (reject).
3. Если вложения не являются подозрительными, то они перепаковываются в архив с паролем.
4. Если письмо признано спамом, модифицируется тема письма и добавляются соответствующие служебные заголовки.
5. В конце письма возвращается действие, которое необходимо применить к письму.

Как видим, ничего сложного.

Полный текст реализации выложен по адресу:

https://github.com/DoctorWebLtd/drweb-lua-examples/blob/master/maild/milter/api_example.lua

Пример работы с белыми и черными списками ключевых слов

```
-- Auxiliary Dr.Web Lua module providing common utilities
local drweb = require "drweb"

-- Auxiliary Dr.Web Lua module providing regexp checks
local regex = require "drweb.regex"
    -- regex.search(pattern, text [, flags])
    -- regex.match(pattern, text [, flags])

-- Load regexp patterns from files
local whitelist = drweb.load_array("/opt/drweb.com/lists/whitemails.txt")
local blacklist = drweb.load_array("/opt/drweb.com/lists/blackmails.txt")

-- Entry point to check email message sent to the Dr.Web MailD by Rspamd protocol
function milter_hook(ctx)

    -- Stop checks if mail_from matchs one of the patterns loaded from file
    for _, pattern in ipairs(whitelist) do
        if regex.match(pattern, ctx.from, regex.ignore_case) then
            return {action = "accept"}
        end
    end

    -- Stop checks if mail_from matchs one of the patterns loaded from file
    for _, pattern in ipairs(blacklist) do
        if regex.match(pattern, ctx.from, regex.ignore_case) then
            return {action = "reject", message = "Blacklist"}
        end
    end

    -- regex.match and regex.search can also taking arrays of patterns
    -- and the code above will be pretty simple:

    -- -- Stop checks if mail_from matchs one of the patterns loaded from file
    -- if regex.match(whitelist, ctx.from, regex.ignore_case) then
    --     return {action = "accept" }
    -- end
    -- -- Stop checks if mail_from matchs one of the patterns loaded from file
    -- if regex.match(blacklist, ctx.from, regex.ignore_case) then
    --     return {action = "reject", message = "Blacklist" }
    -- end

    return {action = "accept"}
end
```

Скрипт состроит из следующих частей:

1. Проверка наличия ключевых слов из списка в проверяемом сообщении.
2. В зависимости от наличия данных ключевых слов выбирается действие, которое необходимо совершить с письмом.

Полный текст реализации выложен по адресу:
https://github.com/DoctorWebLtd/drweb-lua-examples/blob/master/maild/milter/black_white_lists.lua

Пример работы с черными списками адресов и отправителей

```
local drweb = require 'drweb'
local dwxl = require 'drweb.dnsxl'

-- Function checks ip addresses on dnsxl server
local function check_ip(ip, server)
    local log_str = '[' .. ip .. ']'
    local result = dwxl.ip(ip, server)
    -- Return true if ip in blacklist else return false
    if result then
        log_str = 'Bad ip ' .. log_str .. ': '
        for _, record in ipairs(result) do
            log_str = log_str .. record .. ', '
        end
        -- Output message data to log of Dr.Web MailD on level "debug"
        drweb.debug(log_str)
        return true
    else
        log_str = 'Legit ip ' .. log_str
        -- Output message data to log of Dr.Web MailD on level "debug"
        drweb.debug(log_str)
        return false
    end

end

-- Function checks urls on surbl server
local function check_url(url, server)
    local log_str = '[' .. url .. ']'
    local result = dwxl.url(url, server)
    -- Return true if ip in blacklist else return false
    if result then
        log_str = 'Bad ip ' .. log_str .. ': '
        for _, record in ipairs(result) do
            log_str = log_str .. record .. ', '
        end
        -- Output message data to log of Dr.Web MailD on level "debug"
        drweb.debug(log_str)
        return true
    else
```

```
        log_str = 'Legit ip ' .. log_str
        -- Output message data to log of Dr.Web MailD on level "debug"
        drweb.debug(log_str)
        return false
    end
end

-- Entry point to check email message sent to the Dr.Web MailD by Rspamd protocol
function milter_hook(ctx)
    local surbl_server = 'multi.surbl.org'
    local dnsxl_server = 'zen.spamhaus.org'

    -- Reject message if sender ip matches the dnsxl server blacklists
    if check_ip(ctx.sender.ip, dnsxl_server) then
        return {action = "reject", message = "Blocked by blacklists of " ..
dnsxl_server}
    end
    -- Reject message if sender hostname matches the surbl server blacklists
    if check_url(ctx.sender.hostname, surbl_server) then
        return {action = "reject", message = "Blocked by blacklists of " ..
surbl_server}
    end

    return {action = "accept"}
end
```

Скрипт состоит частей, последовательно проверяющих наличие IP-адреса, URL, данных отправителя в черных списках.

Дополнительно в скрипте показывается вывод сообщений в режиме отладки (использование функции drweb.debug).

Полный текст реализации выложен по адресу:

https://github.com/DoctorWebLtd/drweb-lua-examples/blob/master/maild/milter/dnsxl_surbl_lists.lua

Пример работы с базой данных SQLite3

```
local drweb = require "drweb"
local sqlite3 = require "lsqlite3"
local database = '/tmp/drweb.db'

-- Function to add record about a threat to SQLite database
local function db_threat_add(date, host, ip, mail_from, mail_to, threat_name,
threat_type)
    local db = sqlite3.open(database)

    local sql_create = "CREATE TABLE IF NOT EXISTS threats (id integer PRIMARY KEY
AUTOINCREMENT,\
```

```
                    date text,\
                    host text,\
                    ip text,\
                    mail_from text,\
                    mail_to text,\
                    threat_name text,\
                    threat_type text);"

    local sql_add = string.format("INSERT INTO threats(date, host, ip, mail_from,
mail_to, threat_name, threat_type) values\
    ('%s', '%s', '%s', '%s', '%s', '%s', '%s');", date, host, ip, mail_from,
mail_to, threat_name, threat_type)
    local result = assert(db:execute(sql_create))
    -- drweb.notice("SQLite3: create_table result: " .. tostring(result))
    local result = assert(db:execute(sql_add))
    -- drweb.notice("SQLite3: insert_row result: " .. tostring(result))
    db:close()
end

-- Function to add record about spam to SQLite database
local function db_spam_add(date, host, ip, mail_from, mail_to, spam_score)
    local db = sqlite3.open(database)

    local sql_create = "CREATE TABLE IF NOT EXISTS spam (id integer PRIMARY KEY
AUTOINCREMENT,\
                    date text,\
                    host text,\
                    ip text,\
                    mail_from text,\
                    mail_to text,\
                    spam_score text);"

    local sql_add = string.format("INSERT INTO spam(date, host, ip, mail_from,
mail_to, spam_score) \
    values ('%s', '%s', '%s', '%s', '%s', '%s');", date, host, ip, mail_from,
mail_to, spam_score)
    local result = assert(db:execute(sql_create))
    -- drweb.notice("SQLite3: create_table result: " .. tostring(result))
    local result = assert(db:execute(sql_add))
    -- drweb.notice("SQLite3: insert_row result: " .. tostring(result))
    db:close()
end

-- Entry point to check email message sent to the Dr.Web MailD by Milter protocol
function milter_hook(ctx)
    local datetime = os.date()
    local mail_from = ctx.from
    local host = ctx.sender.hostname
    local ip = ctx.sender.ip
```

```
    local mail_to = table.concat(ctx.to, ", ")

    -- Insert info about each found threat into database and reject the message
    if ctx.message.has_threat() then
        for threat, path in ctx.message.threats() do
            db_threat_add(datetime, host, ip, mail_from, mail_to, threat.name,
threat.type)
        end
        return {action = "reject"}
    end

    -- Insert info about spam into database (if spam score exceeds 100) and reject
the message
    if ctx.message.spam.score >= 100 then
        db_spam_add(datetime, host, ip, mail_from, mail_to, ctx.message.spam.score)
        return {action = "reject"}
    end

    -- Accept, if the message is not spam and there are no threats found
    return{action = "accept"}
end
```

Скрипт содержит пример работы с базой данных — добавления в нее таблиц с нужными параметрами и записи информации о проверенных сообщениях.

Полный текст реализации выложен по адресу:

https://github.com/DoctorWebLtd/drweb-lua-examples/blob/master/maild/milter/drweb_sqlite.lua

Пример модификации сообщения

```
function milter_hook(ctx)
    local subject = ""
    -- Устанавливаем в переменной modifier таблицу для осуществления модификаций
    local modifier = ctx.modifier

    -- Получаем тему письма из соответствующего заголовка
    local subject = ctx.message.header.value('Subject') or ''
    -- Проверяем, содержит ли сообщение угрозы, и отвергаем его, если это так
    if ctx.message.has_threat() then
        return {action = "reject"}
    end
    -- Проверяем письмо на спам, и отвергаем его, если число баллов спама превышает
100
    if ctx.message.spam.score > 100 then
        return {action = "reject"}
    end
    -- Если угрозы не найдены и сообщение не признано спамом, модифицируем тему
письма, добавляя к ней наш текст
    modifier.change_header_field("Subject", subject .. " (Checked with Dr.Web Anti-
Virus)")
```

```
    --modifier.change_header_field("Subject", subject .. " (Письмо проверено
антивирусом Dr.Web)")

    -- Пропускаем сообщение, применив все сделанные модификации
    return {action = "accept"}
end
```

Скрипт добавляет в письмо строку с информацией о том, что письмо проверено и письмо не является мошенническим или вредоносным.

Полный текст реализации выложен по адресу:

https://github.com/DoctorWebLtd/drweb-lua-examples/blob/master/maild/milter/modifications_subject.lua

Сценарии, написанные на Lua, могут быть использованы компонентом Dr.Web Firewall для Linux для предварительной проверки соединения перед тем, как направить его на анализ компоненту SpIDer Gate.

Анализ соединения при помощи сценария на языке Lua выполняется, если в настройках компонента Dr.Web Firewall для Linux (в параметре **InterceptHookPath**) задан путь к файлу, содержащему сценарий проверки соединения, написанный на языке Lua. В противном случае обработка соединения производится с использованием настроек по умолчанию и правил обработки, заданных в настройках компонента (параметры **RuleSet***).

Требования к сценарию обработки соединений

Файл должен содержать глобальную функцию, являющуюся точкой входа в модуль проверки соединения (эту функцию Dr.Web Firewall для Linux будет вызывать для обработки вновь поступившего соединения). Функция обработки должна соответствовать следующим соглашениям о вызове:

1. *Имя функции* — intercept_hook;
2. *Единственный аргумент* — таблица Lua InterceptContext (предоставляет из функции доступ к информации об обрабатываемом соединении; см. описание таблицы ниже);
3. *Единственное возвращаемое значение* — строковое значение из таблицы ниже:

Значение Описание вердикта

- pass Пропустить соединение, минуя его проверку в компоненте SpIDer Gate
- check Проверить соединение с помощью компонента SpIDer Gate
- drop Блокировать соединение, организуя потерю пакета
- reject Отвергнуть соединение (клиент, инициирующий соединение, получит TCP-пакет с флагом RST)

Примеры

1. Сценарий, безусловно возвращающий в Dr.Web Firewall для Linux вердикт pass (пропустить) для всех устанавливаемых соединений:

```
-- Функция проверки соединения, написанная пользователем
function intercept_hook(ctx)
return "pass" -- не проверять соединение
end
```

2. Сценарий, предписывающий Dr.Web Firewall для Linux направить на проверку все устанавливаемые соединения, кроме исходящих локальных соединений от приложений, исполняемых с правами пользователя из группы drweb, либо соединений, инициированных с привилегированных портов (вне зависимости от владельца соединения и его направления), либо соединений, исходящих с IP-адреса из локальной подсети:

```
function intercept_hook(ctx)/
-- Не проверять соединения, инициированные с локального
-- узла (divert == "output") приложением от имени группы
-- "drweb" (group == "drweb")
if ctx.divert == "output" and ctx.group == "drweb" then
return "pass"
end
-- Не проверять соединения, инициированные с
-- привилегированных портов (от 0 до 1024)
if ctx.src.port >= 0 and ctx.src.port <= 1024 then
return "pass"
end
-- Не проверять соединения с адресов из локальной подсети
-- (диапазона IP-адресов 127.0.0.1/8)
if ctx.src.ip.belongs("127.0.0.0/8") then
return "pass"
end
-- По умолчанию соединение проверяется
return "check"
end
```

Пример реализации необходимого функционала для модуля Dr.Web Firewall

```
local drweb = require "drweb"

-- Entry point to check network connections
function intercept_hook(ctx)
    -- 1. Do not check Dr.Web processes' outgoing connections
    if ctx.divert == "output" and ctx.group == "drweb"
    then
        -- Pass the connection
        return "pass"
    end

    -- 2. Other connections should be checked
    -- The rule below allows connection if current time satisfies the specified con-
dition

    -- Condition for time is specified using cron record format (from 8 to 18 hours,
from Mon to Fri)
    local datetime_rules = {
        worktime = "* 8-18 * * 1-5"
    }
```

```
    if ctx.divert == "forward" or ctx.divert == "input" then
        -- If current time satisfies the condition, connection will be checked for
threats
        if rules_processor(datetime_rules.worktime) then
            return "check"
        end
    end

    -- 3. Reject connection if all conditions above are false
    return "reject"
end

-- Auxiliary function.
-- The function gets condition for date and time in cron record format and returns
true if the current time satisfies the condition.
-- Condition should be specified as a string using simplified cron format allows
absolute values (10 14 * * *) and ranges (0 12-14 * * 1-5).
-- Fields of the string are as follows:
-- * * * * *
-- | | | | |
-- | | | | ----- Day of week (0 - 6, where 0 is Sunday)
-- | | | ------- Month (1 - 12)
-- | | --------- Day (1 - 31)
-- | ----------- Hours (0 - 23)
-- ------------- Minutes (0 - 59)

function rules_processor(cron)

    local function check(now, rule)
        if rule == "*" then
            return true
        elseif string.find(rule, "-") then
            local thresholds = string.gmatch(rule, '([^-]+)')
            local min_th, max_th = tonumber(thresholds()), tonumber(thresholds())
            if tonumber(now) >= min_th and tonumber(now) <= max_th then
                return true
            else
                return false
            end
        elseif rule == now then
            return true
        else
            return false
        end
    end
```

```
    local now_stamp = string.gmatch(os.date("%M %H %d %m %w"), '([^%s]+)')
    local now_min, now_hour, now_day, now_month, now_dow = now_stamp(),
now_stamp(), now_stamp(), now_stamp(), now_stamp()

    local rule_stamp = string.gmatch(cron, '([^%s]+)')
    local rule_min, rule_hour, rule_day, rule_month, rule_dow = rule_stamp(),
rule_stamp(), rule_stamp(), rule_stamp(), rule_stamp()

    if check(now_min, rule_min) and check(now_hour, rule_hour) and check(now_day,
rule_day) and check(now_month, rule_month) and check(now_dow, rule_dow) then
        return true
    else
        return false
    end

end
```

Полный текст реализации выложен по адресу:

https://github.com/DoctorWebLtd/drweb-lua-examples/blob/master/firewall/time_conditions.lua

О компании «Доктор Веб»

«Доктор Веб» — российский производитель антивирусных средств защиты информации под маркой Dr.Web. Продукты Dr.Web разрабатываются с 1992 года. Все права на технологии Dr.Web принадлежат компании «Доктор Веб». «Доктор Веб» — один из немногих антивирусных вендоров в мире, владеющих собственными уникальными технологиями детектирования и лечения вредоносных программ, имеет свою антивирусную лабораторию, глобальную службу вирусного мониторинга и службу технической поддержки, которые расположены в России.

Компания «Доктор Веб» — ключевой игрок на российском рынке программных средств обеспечения базовой потребности бизнеса — безопасности информации. Свой выбор в пользу продуктов Dr.Web сделали Государственная Дума Федерального Собрания РФ, ЦИК России, Минобороны России, Верховный Суд РФ, Совет Федерации Федерального Собрания РФ, Центральный банк Российской Федерации, многие другие государственные учреждения и крупнейшие компании.

Вот только некоторые клиенты Dr.Web: https://customers.drweb.com.

Dr.Web внесен в «Единый реестр российских программ для электронных вычислительных машин и баз данных» **Министерства цифрового развития, связи и массовых коммуникаций Российской Федерации.**

Использование отечественного антивирусного ПО Dr.Web обеспечивает нашим клиентам защиту от рисков, связанных с изменением международной обстановки, таких как отказ в использовании, продлении, поставке или получении обновлений, а также от угроз, созданных для целенаправленных атак на предприятия и граждан России.

Со знаком качества

«Доктор Веб» имеет сертификаты, позволяющие использовать ПО Dr.Web в организациях с повышенными требованиями к уровню безопасности.

Dr.Web сертифицирован на отсутствие недекларированных возможностей – по 2 уровню контроля, на соответствие требованиям документа «Требования к средствам антивирусной защиты», утв. приказом ФСТЭК России № 28 от 20.03.2012 г., на соответствие требованиям ФСБ России к антивирусным средствам.

Продукты Dr.Web применяются для защиты информации, содержащейся в различных информационных системах, в том числе информации ограниченного доступа (государственная тайна, персональные данные и т. д.).

Использование ПО Dr.Web позволяет обеспечить надлежащее выполнение требований норм законодательства РФ о применении мер для защиты:

– информации с ограниченным доступом (государственная тайна, персональные данные и т. д.);
– отдельных категорий граждан от информации, причиняющей вред.

Сертификаты ФСТЭК России	**Сертификаты Минобороны России**	**Сертификаты ФСБ России**	**Все сертификаты и товарные знаки**

Государственные сертификаты и награды, а также география пользователей Dr.Web свидетельствуют о высоком качестве продуктов, созданных талантливыми российскими программистами.

125040, Россия, Москва, 3-я улица Ямского поля, вл. 2, корп. 12а
Тел.: +7 495 789–45–87 (многоканальный)
Факс: +7 495 789–45–97
https://антивирус.рф | https://www.drweb.ru | https://curenet.drweb.ru | https://free.drweb.ru