| TP-LINK® | **Name:** Description - L7 traffic injector | | |
|---|---|---|---|
| **№** | **Number, version** | **Date** | **Author** |
| 1 | Version 1.0 | 2015-08-07 | Konstantin Mauch, Senior Software Engineer |

# Preamble

Using L7-traffic injections to automate remote setting on L2/L3 network hardware instead of clicking on web-interface buttons (of internal web-server of this hardware) can save a lot of admin's time.

# 1. Introduction

Current library implements hypertext traffic injections for automation of remote setting on network-switching and -routing hardware.

This library was tested on TL-SL2218, TL-SL2428, TL-SL5428E, and most of below operations have successfully passed the testing.

Library is still under development and its up-to-date revision is permanently stored at https://github.com/metallistov20/trafinject/ . Feel free to deploy under your tasks and to contribute.

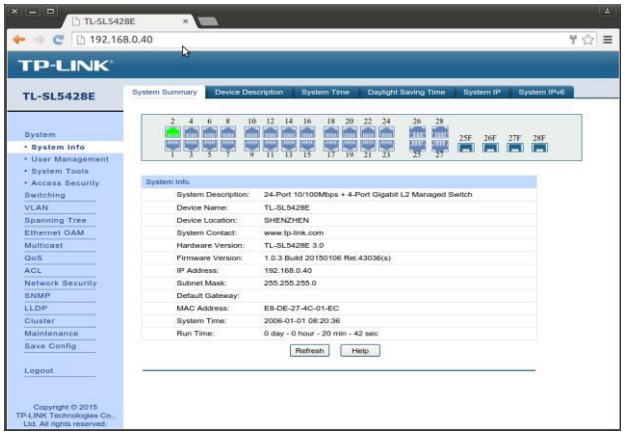
Library's prospective is: to get stable edition and then to get ported onto L3 hardware (routers, ADSL modems).

# 2. Short description of library functions

## 2.1. iOpenSite( )

Function int iOpenSite( ) performs 'opening' of web-interface of switch. I.e. simulates passing authentication data (username and password) to a authentication form (applet) and receives response with session id ( tID hexadecimal number needed for further L7 requests).


2.1.1. Before opening site




2.1.2. After site was opened

Two-step operation. In first message it does referencing to <http://<TARGET_IP>/logon/LogonRpm.htm> with POST payload containing user authentication credentials, on second actually opens the site – references page <http://<TARGET_IP>/> with POST payload containing user ID and level, on this the HTTPS packet with response arrives to STDOUT of shell. This response contains session ID, which is 32-digit hexadecimal, and which is valid for all other operations till current session is closed by iCloseSite operation.

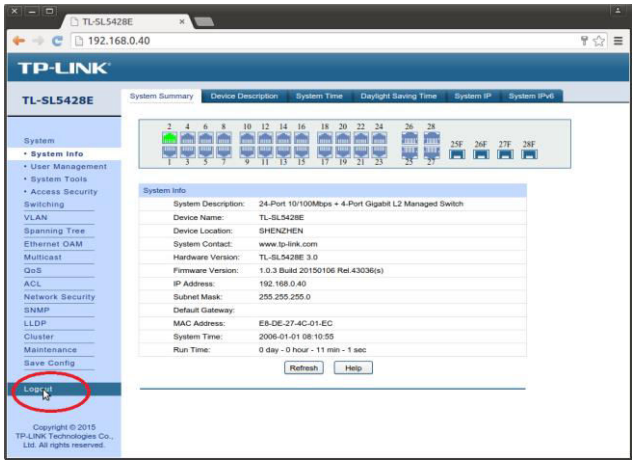
Let's assume we want to open site located at 192.168.0.40, then from command line we must issue:

```
#./inject --open --target=192.168.0.40
```

The site becomes opened and response HTTP traffic contains tID. How this tID can be parsed out we will know from section 3.1.

## 2.2. int iCloseSite( )

Function int iCloseSite( ) simulates clicking onto 'Logout' button (menu entry) in web-interface of switch.





2.2.1. Before closing the site

2.2.2. After site was closed

Two-step operation. In first message it performs referencing to <http://<TARGET_IP>/userRpm/Logout.htm> without payload, and in second – to <http://<TARGET_IP>/>. On this the finalizing HTTPS response arrives to STDOUT of shell. After this the session ID created on iOpenSite and used by other functions during previous steps is not valid anymore, and dynamically generated HTM-pages are destroyed ( not available for referencing ).

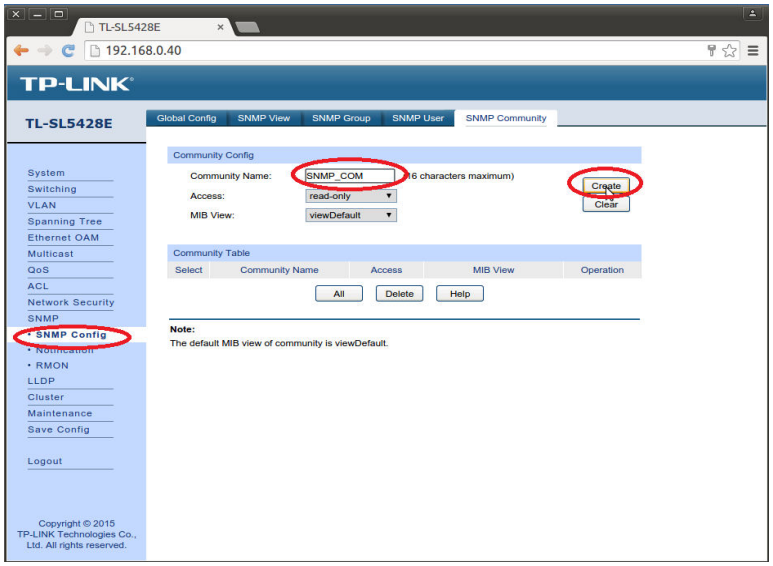
Let's assume we want to close site located at 192.168.0.40, then from command line we must issue:

```
#./inject --close --target=192.168.0.40
```

The site get's closed, after which any reference to it will not be responded.

## 2.3. int iCreateSnmp( )

Function int iCreateSnmp( ) creates SNMP group with read-write access rights with given name. Such group is necessary if once wants to access SNMP strings of switch via SNMP protocol.




2.3. Create SNMP

Five-step operation. In first message it references to <http://<TARGET_IP>> with no extra payload, in second – to <http://<TARGET_IP>/userRpm/SNMPv3CommunityConfigRpm.htm>, in third – to <http://<TARGET_IP>/userRpm/SNMPv3CommunityConfigRpm.htm/?s_userlevel=1&_tid_=<tID>>, in fourth – to to <http://<TARGET_IP>/userRpm/SNMPv3CommunityConfigRpm.htm/?txt_comname=<SNMP_GROUP>&comRight=2&comView=0&button=Add&_tid_=<tID>>, in fifth - <http://<TARGET_IP>/userRpm/SNMPv3GlobalConfigRpm.htm?snmpState=1&button=stateSubmit&_tid_=<tID>> .

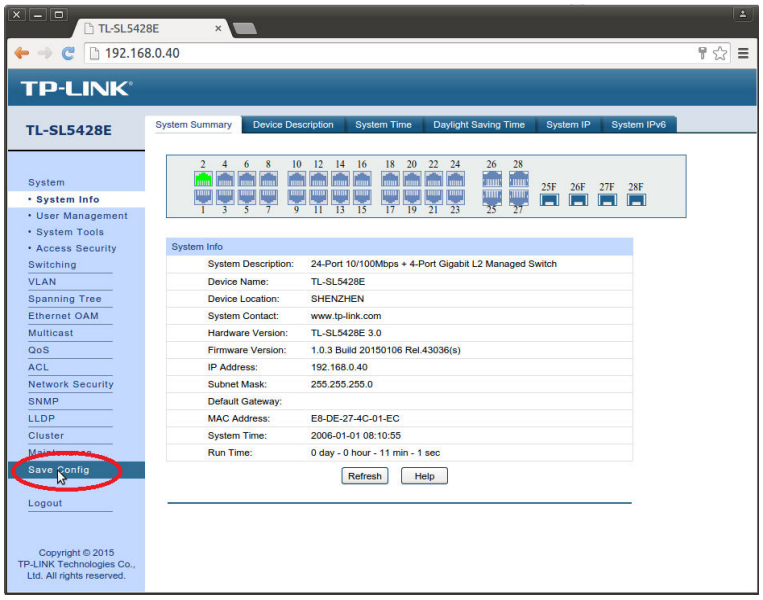
Let's assume we want to create SNMP group named 'SNMP_COM1' at site located at 192.168.0.40, then from command line we must issue:

```
#./inject --create --id=58b10980f6f32c2f --target=192.168.0.40 --community=SNMP_COM1
```

Attention: instead of "`58b10980f6f32c2f` " you must put the tID received inside response to iOpenSite( ) command, see above, and section 3.1. This command creates require SNMP group with read-write right, so you may use SNMP-Tools to access server via SNMP protocol.

## 2.4. int iSaveSite( )

Function int iSaveSite( ) simulates clicking 'Save Config' button (menu entry) in web-interface of switch.




2.4. Save site

Four-step operation. In first message it references to <http://<TARGET_IP>> with no extra payload, in second – to <http://<TARGET_IP>/userRpm/ConfigsaveRpm.htm?s_userlevel=1&_tid_=<tID>&_tid_=<tID>>, in third – to <http://<TARGET_IP>/userRpm/ConfigsaveImg.htm>, and in fourth – to <http://<TARGET_IP>/userRpm/ConfigsaveFileRpm.htm?_tid_=<tID>>.

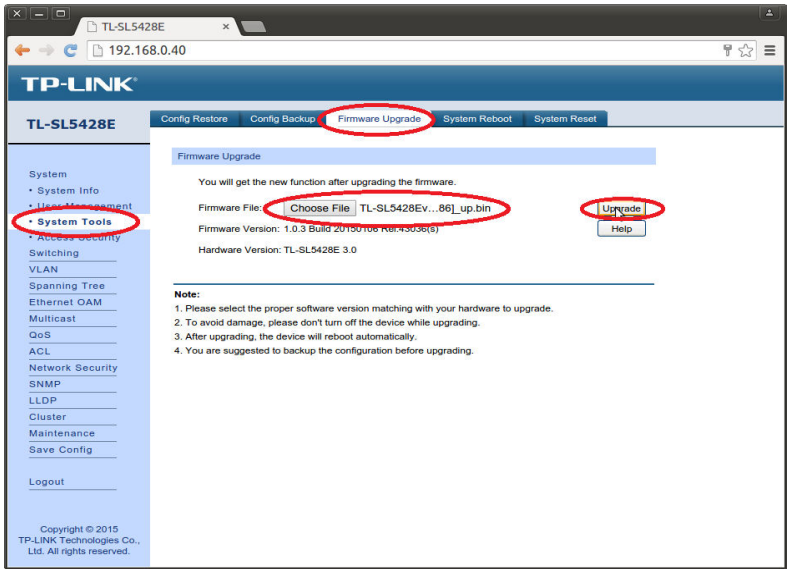
Let's assume we want to save changes done to server located at 192.168.0.40, then from command line we must issue:

```
#./inject --save  --id=58b10980f6f32c2f --target=192.168.0.40
```

Attention: instead of "`58b10980f6f32c2f` " you must put the tID received inside response to iOpenSite( ) command, see above, and section 3.1. This saves any changes you've done during recent session into NVRAM of server, so the will be actual after reboot.

## 2.5. int iUpgradeFirmware( )

Function int iUpgradeFirmware( ) uploads binary file with firmware image on switch, and initiates firmware writing into flash ( on switch side ) .




2.5. Upgrade FW procedure

Five-step operation. In first message it references to <http://<TARGET_IP>/userRpm/FirmwareRpm.htm?s_userlevel=1&_tid_=<tID>> , in second – to <http://<TARGET_IP>/help/FirmwareHelpRpm.htm>. TODO: Presumably this message can be omitted (this presumption is being checked currently). Then the program passes main flow to a shell and form within shell executes two processes: first - < /usr/bin/curl -F filedata=@<LOCAL_FILENAME_OF_FIRMWARE_TO_UPLOAD> http://<TARGET_IP>/ > - on this the necessary fields in http-form get filled;

and second - < /usr/bin/curl --form submit=@<LOCAL_FILENAME_OF_FIRMWARE_TO_UPLOAD> --form submit=upgrade --form _tid_=<tID> http://<TARGET_IP>/userRpm/FirmwareAdRpm.htm > - on this the form gets submitted.

Finally, the main flow returns back to a program , and program references fifth page <http://<TARGET_IP>/userRpm/FirmwareUpdateTempRpm.htm?_tid_=<tID>> - on this the server side starts procedure of uploading via HTTP protocol and writing down to a flash the firmware defined as <LOCAL_FILENAME_OF_FIRMWARE_TO_UPLOAD> . Note that after this operation server side gets not responding till flashing down is finished amd server side is rebooted. Once the new firmware contains

specific settings (such as different MAC or different IP, etc) then following calls of current program should encounter this.

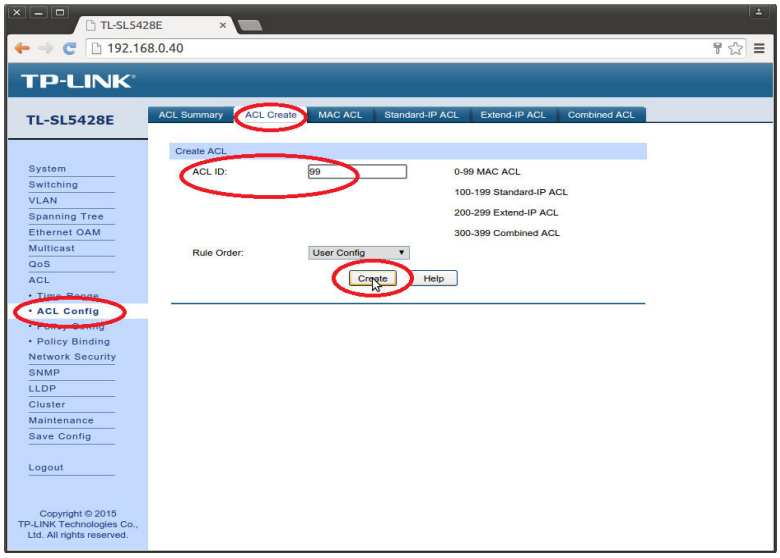
Let's assume we want to upload and flash firmware onto site located at 192.168.0.40, then from command line we must issue:

```
#./inject --upgrade --target=192.168.0.40  --id=58b10980f6f32c2f --filename=TL-SL5428Ev3.bin
```

Attention: instead of "58b10980f6f32c2f " you must put the tID received inside response to iOpenSite( ) command, see above, and section 3.1., instead of "TL-SL5428E.bin" you must put proper firmware name ( with full path) you want to upload onto server.

## 2.6. int iAclGroup( )

Function int iAclGroup( ) creates an ACL group with given ID.




2.6. ACL group creation

Three-step operation. In first message it references to <http://<TARGET_IP>> without extra payload, then it references <http://<TARGET_IP>/userRpm/ACLRuleCreateRpm.htm?s_userlevel=1&_tid_=<tID>> page, and finally - <http://<TARGET_IP>/userRpm/ACLRuleCreateRpm.htm?submit=Submit&aclId=<cACL>&ruleOrder=1&_tid

_=<tID>>, where <cACL> is an index of ACL group, which is decimal from range [0 ..299]. in this case the ACL group name is blank.

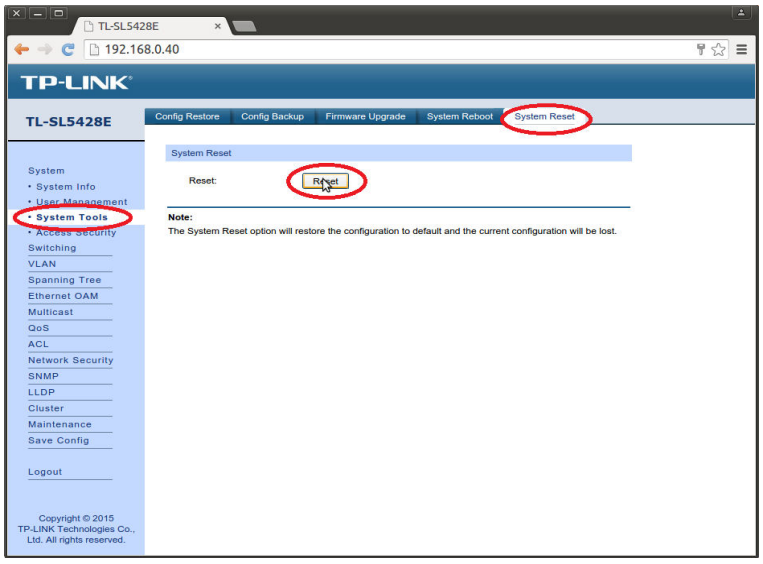
Let's assume we want to create ACL group with ID 98 on site located at 192.168.0.40, then from command line we must issue:

```
#./inject --ACL --target=192.168.0.40  --id=58b10980f6f32c2f --acl-data=98
```

Attention: instead of "58b10980f6f32c2f " you must put the tID received inside response to iOpenSite( ) command, see above, and section 3.1.

## 2.7. int iRebootSwitch( )

Function int iRebootSwitch( ) issues request to reboot the switch immediately.




2.7. Reboot switch

Three-step operation. In first message it references to <http://<TARGET_IP>> without extra payload, in second one – to <http://<TARGET_IP>/userRpm/ReiniRstAdRpm.htm?restore=Reset&_tid_=<tID> > page, on third – to <http://<TARGET_IP>//userRpm/ReiniRstAdTempRpm.htm?_tid_=<tID> > page. This operation reboots the server side as is and immediately, so please take care to have you recent changes saved, and your further references to a server be issued not earlier than 30-200 seconds (depends on instance of network switch).

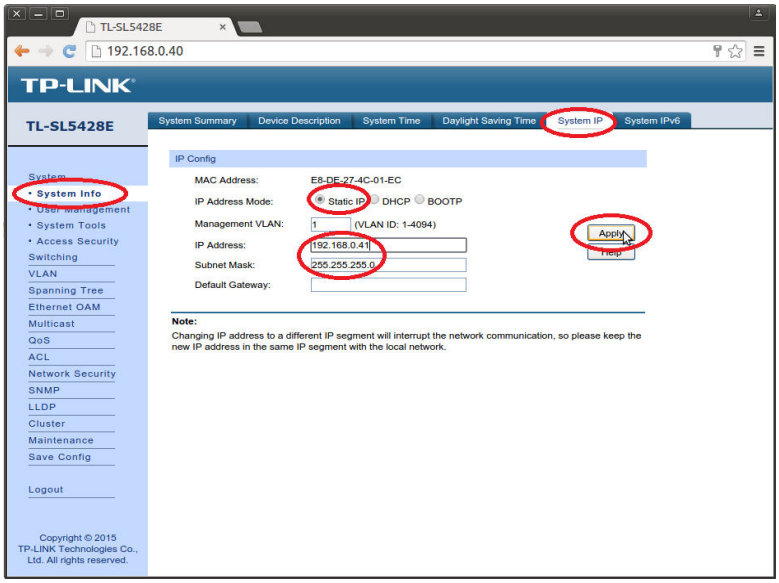
Let's assume we want to reboot network switch located at 192.168.0.40, then from command line we must issue:

```
#./inject --reboot --target=192.168.0.40  --id=58b10980f6f32c2f
```

Attention: instead of "58b10980f6f32c2f " you must put the tID received inside response to iOpenSite( ) command, see above, and section 3.1.

## 2.8. int iAssignIp( )

Function int iAssignIp( ) assigns static IP with network mask, and receives confirmation from switch – in response packet the IP-address is already altered.




2.8. Assigning static IP address

Five-step operation. This operation assigns static IP to a server side. IP must be given along with subnet mask, and without DNS server address. Beware that in the middle of procedure the IP get changes so the second half of messages should be “addressed” to a server with already changed target IP. In first message it references to <http://<TARGET_IP>/SystemInfoRpm.htm?s_userlevel=1&_tid_=<tID>&_tid_=<tID>> without extra payload, in second – to <http://<TARGET_IP>/userRpm/SystemIpRpm.htm?s_userlevel=1&_tid_=<tID>>, in third – to <http://<TARGET_IP>/userRpm/SystemIpRpm.htm?ip_mode=0&ip_mgmt_vlanid=1&ip_address=<ADDDRESS_BEING_ASSIGNED>&ip_mask=<MASK_BEING_ASSIGNED>&ip_gateway=&submit=Apply&_tid_=<tID> >. In fourth we need to reference a front page at already changed target IP address, like this: <http://<ADDDRESS_BEING_ASSIGNED>/ >. TODO: Presumably this message can be omitted (this presumption is being checked currently). On fifth step – we referencing page at at already changed target IP address but still with same tID, like this: <http://<ADDDRESS_BEING_ASSIGNED>/userRpm/SystemInfoRpm.htm?s_userlevel=1&_tid_=<tID>>. It's

clear that after this moment we need to reference any page at already changed target IP address , including “Save Config” and “Close Site” pages. Once parameters saving was done (with “Save Config”) the changed target IP address will be actual for this server after reboot, and of course during all following sessions.

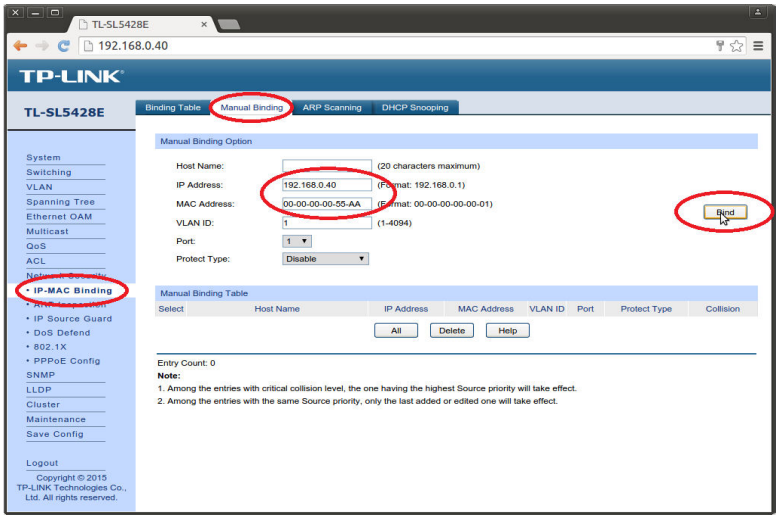
Let's assume we want to assign an IP address 192.168.0.44 to a network switch located at 192.168.0.40, then from command line we must issue:

```
#./inject --ipassign --target=192.168.0.40  --id=58b10980f6f32c2f --ip-addr=192.168.0.44 --ip-mask=255.255.255.0
```

Attention: instead of "58b10980f6f32c2f " you must put the tID received inside response to iOpenSite( ) command, see above, and section 3.1. Switch obtains new IP in the middle if this operation and with this new IP until nearest reboot.

## 2.9. int iBindMacIp( )

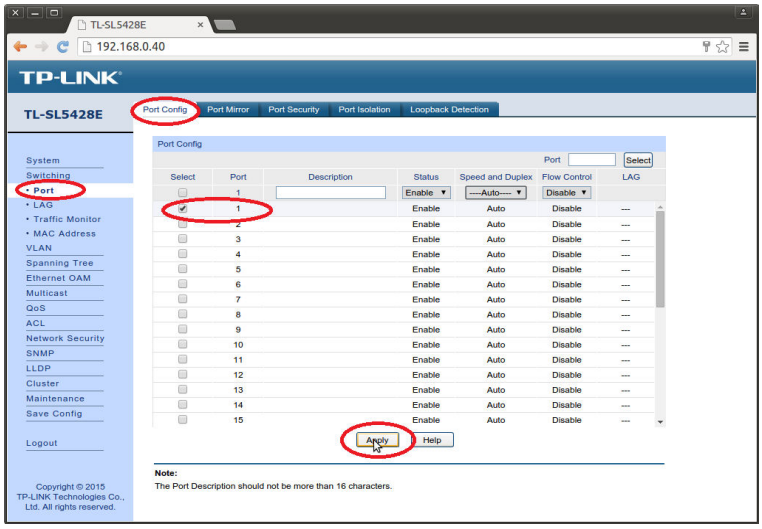
Function int iBindMacIp( ) performs manual binding of given IP and given MAC address .Four-step operation.This operation provides IP-MAC binding in “Network security” applet. *Being prepared currently*.




2.9. Binding MAC and IP

## 2.10. int iEnablePort( )

Three-step operation to implement port enabling or disabling in “Switching” applet. *Being prepared currently*.




2.10. Enable (*disabling* - optional) the port

# 3. Usage from within BASH script framework

In following set of examples we work with library from within upper-layer software wrapper represented by few BASH items such as "inject.sh" (wrapper itself), "functions.sh" (functions to check correctness of parameters passed to 'inject.sh') . To deploy wrapper the executable 'inject' must be compiled and stored near (in current directory).

## 3.1. Parsing out session ID

Let's study the following BASH fragment. In the first step it perform opening the site < ./inject --open . . . > and stored STDOUT response into file <$TMP_NAME >. In the second step it goes across the entire file < while read line; do . . . . done < ./$TMP_NAME > and inspects each string for correspondence to some string < if [[ $line =~ $tID_STR ]]; then > and once found corresponding performs truncating of aux symbols thus getting bare tID as long hexadecimal without 0x prefix. The value is stored in < tID > variable , and parsing file is stopped.

```
./inject --open --target="$IP">$TMP_NAME

while read line; do
 if [[ $line =~ $tID_STR ]]; then
        tID_complete_ENTRY=($(echo $line));
            tID_dirty_ENTRY=${tID_complete_ENTRY[3]}
        tID_exact_ENTRY=${tID_dirty_ENTRY#"""}
        tID=${tID_exact_ENTRY%%??}
        echo "$tID" && break
 fi
done < ./$TMP_NAME
```

## 3.2. Upgrading firmware

Let's study the following BASH fragment. On the first step it performs system upgrade with new firmware < ./inject --upgrade . . . >, and on the second it closes the session <./inject --close . . . > . Although second seems to be useless because the switch closes the session automatically on system reboot (which is necessary part of system upgrade procedure), this step is necessary because when the system upgrade fails we don't have automated closing the session and need to do it manually ( to provide 'clean environment' for other references to a server which can follow after current system upgrade operation) .

```
if  [[ $OPERATION == "upgrade" ]]; then
                ./inject --upgrade --target="$IP" --id="$tID" --filename=$FILENAME
                # 1. no use to call --save because normally it will reboot right now)
                # 2. but in case smth went wrong (and it is not being rebooted now) way we need close site
                ./inject --close --target="$IP"
                exit 0
```

## 3.3. Rebooting switch

Let's study the following BASH fragment. On the first step it performs system reboot < ./inject --reboot . . . >, and on the second it closes the session <./inject --close . . . > . The second step <./inject --close . . .> is placed here for same reason as in p. 3.2. - if the reboot for some reason fails we need to proceed with 'clean' session, so we close present session and assume that newly opened session will percept any further operations correcly.

```
elif  [[ $OPERATION == "reboot" ]]; then
                echo "<$0>: Performing remote reboot of (TL-SL$MODEL) switch";
                ./inject --reboot --id="$tID" --target="$IP"
                ./inject --close --target="$IP"
                exit 0
```

## 3.4. Assigning static IP address

Let's study the following BASH fragment. On the first step it assigns new IP address to a system < ./inject -- ipassign . . . >, and on the second it closes the session <./inject --close . . . > . It is important to note that on second step we are closing the site with already changed ( i.e. already new ) IP address.

```
        elif  [[ $OPERATION == "static" ]]; then
                echo "<$0>: Assigning static IP address ( --ip-addr="$4" --ip-mask="$5")";
                ./inject --ipassign --id="$tID" --target="$IP" --ip-addr="$4" --ip-mask="$5"
                # The adress on switch has already been changed. Beware what you pass as <--target> .
                ./inject --close --target="$4"
                exit 0
```

## 3.5. Saving changes on the switch into NVRAM

From the following BASH fragment we see that to save changes to a switch we require only target IP of the switch and current session ID.

```
        elif  [[ $OPERATION == "save" ]]; then
                echo "<$0>: To save changes allegedly done earlier in web interface of switch "
                ./inject --save --id="$tID" –target="$IP"
```

## 3.6. Processing not implemented operations (closing session)

From the following BASH fragment demonstrates that if something went wrong way and we don't know which operation we should execute on switch within current session we better to close the session and leave framework reporting an error. It is vital in order to not put the switch into unpredictable state.

```
        else
                echo "<$0>: strange case, exiting (nothing to do here)"
                # odd case. the faster we get away from here the better. opened site must be closed before
                ./inject --close --target="$IP"
                exit -1
        fi
```

# 4. Usage from within CMD script framework

Now let' s observe how we can use above wrapper in order to implement some administration of switches. In current examples we work with CMD batches, they call wrapper ( and such accessories as 'enroll.sh' ) and store the STDOUT into log files for further studying/evaluation.

## 4.1. Assigning serial number

Assigning of Serial Number is a three-stage operation, which firstly opens the site, then by means of SNMP-TOOLS (see 'enroll.sh') takes MAC address from switch, and searches which Serial Number corresponds to the MAC address (see 'dev-mac-id.base.txt') , and then writes this Serial Number down to the switch. On third stage ( changed Serial Number should be somehow memorized in the switch) it save the recent changes into NVRAM.

```
echo "1. Creating SNMP on the switch. (Supressing STDOUT output is obvious.)"
./inject.sh 192.168.0.137 create 2218 SNMP_COM15 >./_create.log
echo "1. Created or found existing"

echo "2. Writing S/N into switch"
./enroll.sh 192.168.0.137 2218 SNMP_COM15
echo "2. Writen (or left intact)"

echo "3. Saving altered S/N in the switch. (Supressing STDOUT output is obvious.)"
./inject.sh 192.168.0.137 save 2218 >./_save.log
echo "3. Saved"
```

## 4.2. Assigning static IP address

Assigning a static IP address to a switch is two-stage operation, first one - assigning IP address with subnet mask, second one - saving changes into NVRAM

```
echo "1. Assigining static IP settings"
./inject.sh 192.168.0.137 static 2218 192.168.0.138 255.255.255.0 >./_static.log
echo "1. IP address settings issued"

echo "2. Saving changed IP adddress settings. (Supressing STDOUT output is obvious.)"
./inject.sh 192.168.0.138 save 2218 >./_save.log
echo "2. Saved"
```

## 4.3. Creating ACL group

Creation of ACL group is two-stage operation, on first stage we create ACL group , on second one we save these changes down to the NVRAM of switch.

```
echo "1. Creating ACL group"
./inject.sh 192.168.0.1 acl 2218 99 >./_acl.log
echo "1. ACL group creation command issued"

echo "2. Saving created ACL in the switch. (Supressing STDOUT output is obvious.)"
./inject.sh 192.168.0.1 save 2218 >./_save.log
echo "2. Saved"
```

## 4.4. Upgrading firmware

Upgrading firmware is single-stage operation because in case it was successful the switch goes down and reboots after some time ( this time operates under new firmware ), and in unsuccessful case care of closing session is done in underlying wrapper ( 'inject.sh' ).

```
echo "1. Upgrading FW on the switch. (Supressing STDOUT output is obvious.)"
./inject.sh 192.168.0.1 upgrade 2218 TL-SL2218v1_underTest.bin >./_upgrade.log
echo "1. Upgraded in case was different"
```

## 4.5. Rebooting switch remotely

Rebooting switch is single-stage operation for the same reason as in case of 'Upgrade firmware', see p. 4.4.

```
echo "1. Remote reboot of the switch. (Supressing STDOUT output is obvious.)"
./inject.sh 192.168.0.1 reboot 2218 >./_reboot.log
echo "1. Reboot command issued"
```

# 5. References

## 5.1. http://curl.haxx.se/docs/features.html

## 5.2. http://www.tp-link.ua/resources/simulator/TL-SL5428E_V3/Index.htm

### 5.3. https://dl.packetstormsecurity.net/papers/attack/Aspect_File_Download_Injection.pdf

## 5.4. https://www.owasp.org/index.php/HTTP_Request_Smuggling

## 5.5. https://en.wikipedia.org/wiki/Session_ID

## 5.6. https://github.com/metallistov20/trafinject

| TP-LINK® | **Name:** Description - L7 traffic injector | | |
|---|---|---|---|
| **№** | **Number, version** | **Date** | **Author** |
| 1 | Version 1.0 | 2015-08-07 | Konstantin Mauch, Senior Software Engineer |

# Table of contents


Preamble 1

1. Introduction 1

2. Short description of library functions 2

2.1. iOpenSite( ) 2

2.2. int iCloseSite( ) 3

2.3. int iCreateSnmp( ) 3

2.4. int iSaveSite( ) 5

2.5. int iUpgradeFirmware( ) 6

2.6. int iAclGroup( ) 7

2.7. int iRebootSwitch( ) 8

2.8. int iAssignIp( ) 9

2.9. int iBindMacIp( ) 10

2.10. int iEnablePort( ) 10

3. Usage from within BASH script framework 12

3.1. Parsing out session ID 12

3.2. Upgrading firmware 12

3.3. Rebooting switch 13

3.4. Assigning static IP address 13

3.5. Saving changes on the switch into NVRAM 13

3.6. Processing not implemented operations (closing session) 14

4. Usage from within CMD script framework 15

4.1. Assigning serial number 15

4.2. Assigning static IP address 15

4.3. Creating ACL group 16

4.4. Upgrading firmware 16

4.5. Rebooting switch remotely 16

5. References 17

5.1. http://curl.haxx.se/docs/features.html 17

5.2. http://www.tp-link.ua/resources/simulator/TL-SL5428E_V3/Index.htm 17

5.3. https://dl.packetstormsecurity.net/papers/attack/Aspect_File_Download_Injection.pdf 17

5.4. https://www.owasp.org/index.php/HTTP_Request_Smuggling 17

5.5. https://en.wikipedia.org/wiki/Session_ID 17

5.6. https://github.com/metallistov20/trafinject 17

Table of contents 18