TP-LINK®			**Name:** Description - conversation injector
№	**Number, version**	**Date**	**Author**
1	Version 2.0	10 September 2016	Konstantin Mauch, Engineer

Creating the executable of project “convinject-1”

To create the main executable file of project “convinject-1” the source code of project should be downloaded, prerequisites compiled and stored to right position in local folder, main executable to be compiled against libraries of prerequisites and - rest - against platform libraries.

Such architecture of “convinject-1” project makes it by certain degree portable across the different platforms and thus applicable in wider range of cases.

Installation of source code of project “convinject-1”

Download the project “convinject-1” from **Git** repository https://github.com/metallistov20/convinject-1.git . Ensure all the files were downloaded. To compile prerequisites run the command script

on Linux

“prereqs.linux” on Linux

on Cygwin (Windows)

“prereqs.cygwin” on Cygwin

Each of above scripts generates shared objects needed for further linkage of the project “convinject-1”. These shared objects are:

on Linux

```
./convinject-1/shared/libcurl.so4
./convinject-1/shared/libssh.so4
./convinject-1/shared/libml2.so.2
```

on Cygwin (Windows)

```
convinject-1/cygcurl-4.DLL
convinject-1/cygssh-4.dll
convinject-1/cygxml2-2.dll
```

Compilation of project “convinject-1”

After the above shared object created it’s time to make project “convinject-1” by running “make” command. If the file “./Makefile” was not produced on previous step, user may alternatively run:

on Linux

make -makefile=./Makefile.linux on Linux

on Cygwin (Windows)

make -makefile=Makefile.cygwin on Linux

On this the executable

on Linux

```
./convinject
```

on Cygwin (Windows)

```
convinject.exe
```

appears in same folder.

Executable inspection

Ensure that just compiled executable dynamically loads those shared objects enumerated in previous section.

on Linux

```
libssh.so.4 => /home/<USERNAME>/convinject-1/shared/libssh.so.4
libcurl.so.4 => /home/<USERNAME>/convinject-1/shared/libcurl.so.4
libxml2.so.2 => /home/<USERNAME>/convinject-1/shared/libxml2.so.2
libc.so.6 => /lib/i386-linux-gnu/libc.so.6
librt.so.1 => /lib/i386-linux-gnu/librt.so.1
libz.so.1 => /lib/i386-linux-gnu/libz.so.1
libgssapi_krb5.so.2 => /usr/lib/i386-linux-gnu/libgssapi_krb5.so.2
/lib/ld-linux.so.2
libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2
liblzma.so.5 => /lib/i386-linux-gnu/liblzma.so.5
libm.so.6 => /lib/i386-linux-gnu/libm.so.6
libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0
libkrb5.so.3 => /usr/lib/i386-linux-gnu/libkrb5.so.3
libk5crypto.so.3 => /usr/lib/i386-linux-gnu/libk5crypto.so.3
libcom_err.so.2 => /lib/i386-linux-gnu/libcom_err.so.2
libkrb5support.so.0 => /usr/lib/i386-linux-gnu/libkrb5support.so.0
libkeyutils.so.1 => /lib/i386-linux-gnu/libkeyutils.so.1
libresolv.so.2 => /lib/i386-linux-gnu/libresolv.so.2
```

on Cygwin (Windows)

```
kernel32.dll => /cygdrive/c/Windows/system32/kernel32.dll
cygwin1.dll => /usr/bin/cygwin1.dll
cygxml2-2.dll => /home/<USERNAME>/<WORK_DIR>/convinject-1/cygxml2-2.dll
cygz.dll => /usr/bin/cygz.dll
cyggcc_s-1.dll => /usr/bin/cyggcc_s-1.dll
cygiconv-2.dll => /usr/bin/cygiconv-2.dll
cygcurl-4.dll => /home/<USERNAME>/<WORK_DIR>/convinject-1/cygcurl-4.dll
cygssh-4.dll => /home/<USERNAME>/<WORK_DIR>/convinject-1/cygssh-4.dll
```

As soon as the above condition is satisfied the executable can be deployed.

Deployment of project “convinject-1” from command line

The executable to be deployed with file name of targets given on command line

on Linux

```
./convinject ./targets.xml
```

on Cygwin (Windows)

```
convinject.exe targets.xml
```

File 'targets.xml'

Let’s preview the internal structure of this file, here is an instance:

contents of file “targets.xml”

```
<?xml version="1.0"?>
<Devices>
        <Target>
                <Type>Network Switch</Type>
                <Name>TL-2218</Name>
                <Address>192.168.0.6</Address>
                <Login>admin</Login>
                <Passwd>admin</Passwd>
                <Datafile>./description_change.txt</Datafile>
                <Proto>ssh</Proto>
        </Target>
        <Target>
                <Type>Network Switch</Type>
                <Name>TL-SL5428E</Name>
                <Address>192.168.0.8</Address>
                <Login>admin</Login>
                <Passwd>admin</Passwd>
                <Datafile>./http_commands.txt</Datafile>
                <Proto>http</Proto>
        </Target>
</Devices>
```

This file has XML structure, and contains two sections named “Target”. This is to process two targets, first one is Network Switch “TL-2218” with IP address 192.168.0.6, the second one is Network Switch “TL-SL5428E” 192.168.0.8.

Besides this, the first Network Switch must be processes over “**ssh**” protocol, and exact instructions for such processing are given in “escription_change.txt” file, and second - over “**http**” protocol, and instructions are in “http_commands.txt”

What's being done when <Proto> field is 'ssh'

When processing protocol is “ssh” the project executable establishes the SSH session with corresponding Target (it’s IP address is “192.168.0.6”) with credentials <admin, admin>.

Scenario to execute, which is defined by “escription_change.txt” file, prescribes a series of CLI commands which are executed on Target automatically (i.e. without parsing an incoming response and analyzing altered state of Target) as if these CLI commands were entered by user during SSH/CLI dialog with Target.

What's being done when <Proto> field is 'http'

When processing protocol is “**http**” the project executable establishes the HTTPS session with corresponding Target (it’s IP address is “192.168.0.8”) with credentials <admin, admin>.

Scenario to execute, which is defined by “http_commands.txt” file, prescribes a series of specific commands, which description can be mastered from “trafinject-1” project, and it’s manual "Description - L7 traffic injector.pdf". And presently to such commands belong the following instances:

- "open" - to open the Target, to establish an HTTPS session with it
- "create" - to create an SNMP Community on the target
- "ACL" - to create ACL Group on Target
- "save" - to save changes done to Target, usually needed before reboot
- "reboot" - to reboot target
- "upgrade" - to upgrade firmware working on Target
- "close" - to close HTTPS with target.

(Attention: not all commands are tested; and each set of commands is unique to each instance of Target.) For more information refer to above mentioned “trafinject-1” project and its manual.

SSH sequence of commands

Under scenario to be executed on first Target (IP address “192.168.0.6”, protocol “ssh”) the CLI commands listed in this file:

contents of file “description_change.txt”

```
enable
configure
system-descript sysname SMB_TEST0
system-descript location UKRAINE_KYIV0
system-descript contact-info "TP-Link UA LTD MET 20 Build 0"
```

are executed. This means that first of all the secured CLI session is started, then advanced mode is enabled (“enable”) , the transition to configuration mode is done (“configure”), then CLI command to change system name is executed (“system-descript sysname SMB_TEST0”), then system command to change the location description is executed (“system-descript location UKRAINE_KYIV0”) , then system command to change contact information is executed (“system-descript contact-info "TP-Link UA LTD MET 20 Build 0").

If you wish to execute additional or alternative commands on given Target please alter the contents of above file. Attention: keep in mind that set of CLI commands for each type of Network Switch is different, so first enter CLI mode manually on this Target, and test the desired commands manually, and *only after* they’ve appeared to work correctly you put them into above file (this is because running unchecked commands in automated mode by means of “convinject-1” may put the switch into wrong operating state).

HTTP sequence of commands

Under scenario to be executed on second Target (IP address “192.168.0.8”, protocol “**http**”) the specific commands which are listed in this file

contents of file “http_commands.txt”

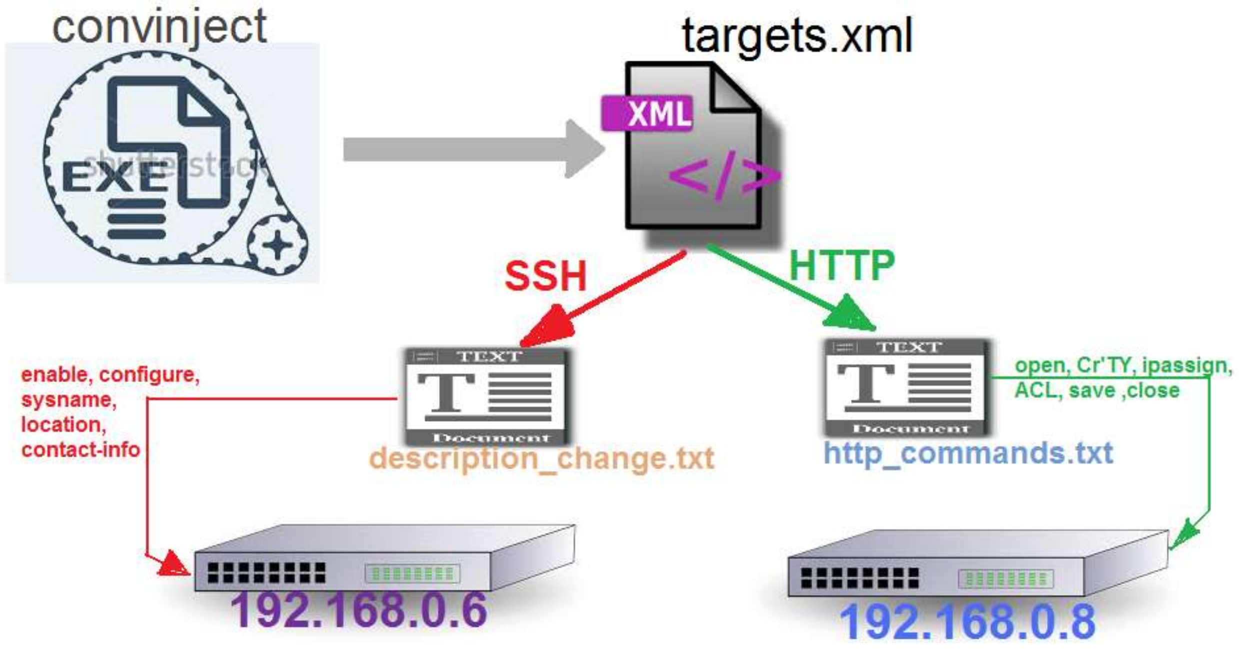
```
open
create community=NEW_SNMP_GROUP
ipassign ip-addr=192.168.0.166 ip-mask=255.255.0.0
ACL acl-data=NEW_ACL_GROUP
save
close
```

are executed.

During execution, firstly the HTTPS connection is opened (“open”) , then community with name “NEW_SNMP_GROUP” is created (“create community=NEW_SNMP_GROUP”), then assignment of new IP address “192.1698.0.166” is done (“ipassign ip-addr=192.168.0.166 ip-mask=255.255.0.0”), then - new ACL group “NEW_ACL_GROUP” created (“ACL acl-data=NEW_ACL_GROUP”), then recent changes are saved (“save”), and HTTPS connection is closed (“close”).

Refer to previously published project <trafinject-1> for appropriate list of commands for each instance of Network Switches. Testing them manually is not effective, so to request clarification it’s better to refer directly to maintainers of <trafinject-1>.

Schematically it looks like on next picture:



Two use cases instances, workflow, and acceptance

To force the project “convinject-1” to implement your scenario on your devices you may first alter the file “targets.xml” and then by iteration alter each file altered by it – such files as “description_change.txt”, “http_commands.txt”, whatever.

Let’s assume you with to reboot a series of switches with IP addresses 192.168.0.18, 192.168.0.19, and 192.168.0.166. In this case the file “reboot.txt” (name can be different, of course) should be placed into main folder of project, and contents of this file is

contents of file “reboot.txt”
open reboot

Afterwards, you should create and place beneath another file “reboot_targets.xml” (again, name can be different) – to enumerate IP addresses of Targets you wish to reboot 192.168.0.18, 192.168.0.19, and 192.168.0.166. The contents is this file is such:

contents if file “reboot_targets.xml”

```
<?xml version="1.0"?>
<Devices>
        <Target>
                <Type>Network Switch</Type>
                <Name>TL- xxXXXXX </Name>
                <Address>192.168.0.18</Address>
                <Login>admin</Login>
                <Passwd>admin</Passwd>
                <Datafile>./reboot.txt</Datafile>
                <Proto>http</Proto>
        </Target>
        <Target>
                <Type>Network Switch</Type>
                <Name>TL- xxXXXXX </Name>
                <Address>192.168.0.19</Address>
                <Login>admin</Login>
                <Passwd>admin</Passwd>
                <Datafile>./reboot.txt</Datafile>
                <Proto>http</Proto>
        </Target>
        <Target>
                <Type>Network Switch</Type>
                <Name>TL-xxXXXXX</Name>
                <Address>192.168.0.166</Address>
                <Login>admin</Login>
                <Passwd>admin</Passwd>
                <Datafile>./reboot.txt</Datafile>
                <Proto>http</Proto>
        </Target>
</Devices>
```

and to run the project :

on Linux

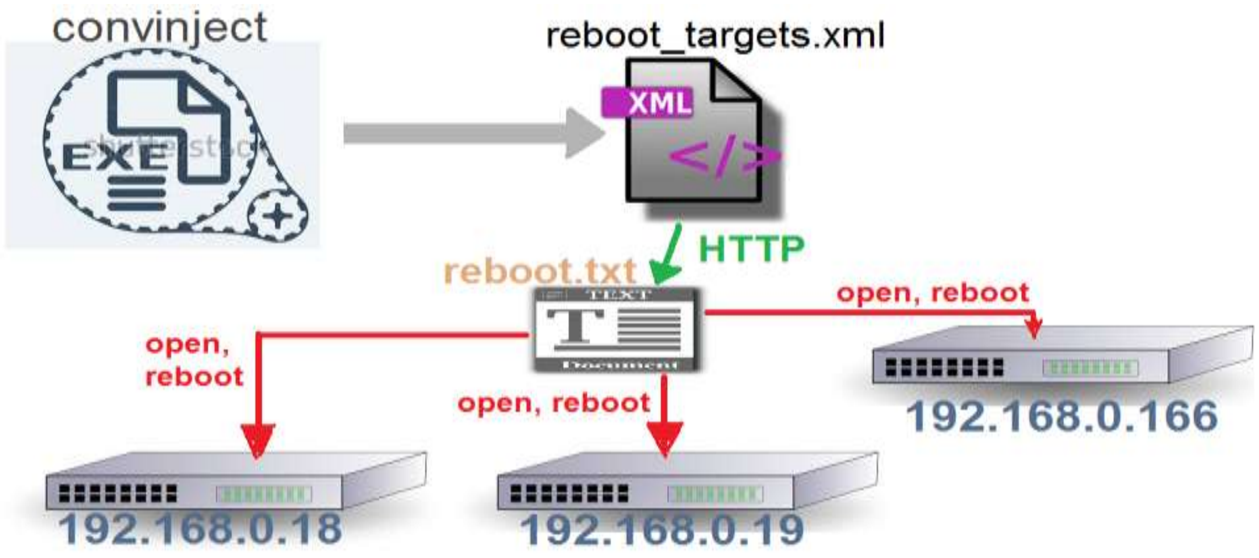
```
./convinject ./reboot_targets.xml
```

on Cygwin (Windows)

```
convinject.exe reboot_targets.xml
```

On this the HTTPS session with each of addresses 192.168.0.18, 192.168.0.19, and 192.168.0.166 will be established and command to reboot the Network Switch will be issued – each switch will be rebooted (and will appear in the network after reboot time).

Schematically it looks like on this picture:



Let’s now assume that instead of reboot you merely wish to alter the contact data of each of these switches. In this case, prepare (and place locally) the new file

contents of file “contact_data_change.txt”

```
enable
configure
system-descript contact-info "Contact Address has Changed to SAMPLE"
```

And the new file “new_contact_data_targets.xml” with following contents:

contents if file “new_contact_data_targets.xml”

```
<?xml version="1.0"?>
<Devices>
        <Target>
                <Type>Network Switch</Type>
                <Name>TL- xxXXXXX </Name>
                <Address>192.168.0.18</Address>
                <Login>admin</Login>
                <Passwd>admin</Passwd>
                <Datafile>./contact_data_change.txt</Datafile>
                <Proto>ssh</Proto>
        </Target>
        <Target>
                <Type>Network Switch</Type>
                <Name>TL- xxXXXXX </Name>
                <Address>192.168.0.19</Address>
                <Login>admin</Login>
                <Passwd>admin</Passwd>
                <Datafile>./contact_data_change.txt</Datafile>
                <Proto>ssh</Proto>
        </Target>
        <Target>
                <Type>Network Switch</Type>
                <Name>TL-xxXXXXX</Name>
                <Address>192.168.0.166</Address>
                <Login>admin</Login>
                <Passwd>admin</Passwd>
                <Datafile>./contact_data_change.txt</Datafile>
                <Proto>ssh</Proto>
        </Target>
</Devices>
```

Pay attention that the communication protocol now is “**ssh**” and scenario filename is “./contact_data_change.txt“. Run this instance as:

on Linux

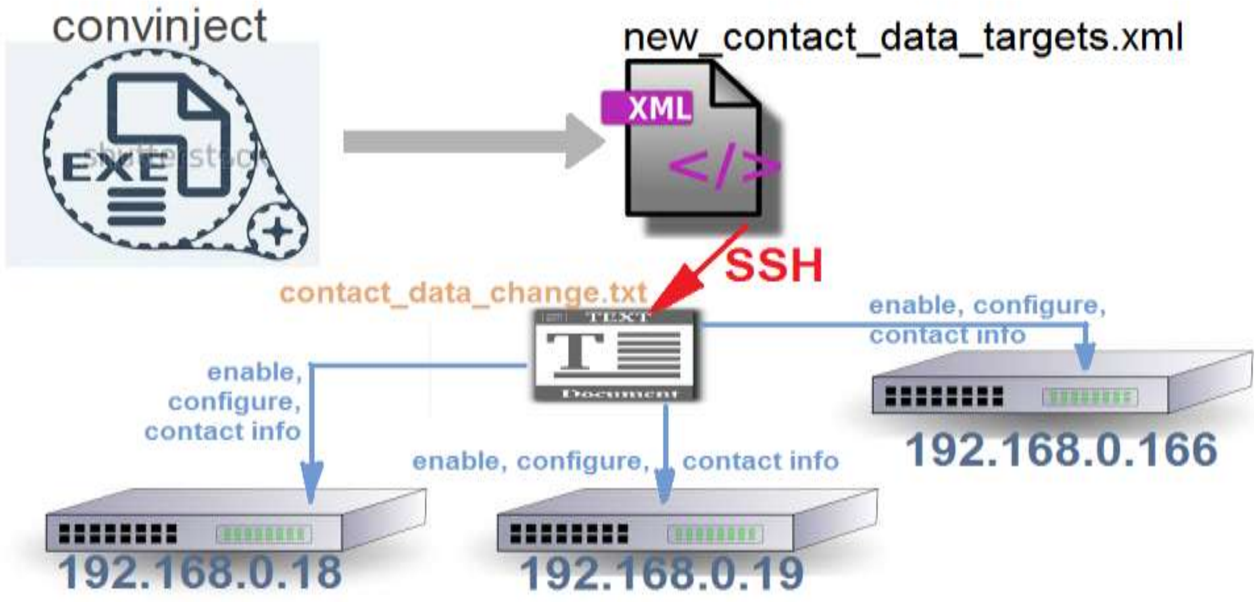
```
./convinject ./new_contact_data_targets.xml
```

on Cygwin (Windows)

```
convinject.exe new_contact_data_targets.xml
```

During run, the SSH connection (with credentials <admin, admin>) will be established with each of addresses 192.168.0.18, 192.168.0.19, and 192.168.0.166, the “configuration” mode will be established on each of them, then - string “*Contact Address has Changed to SAMPLE*” assigned into location information field on each of them.

Schematically it looks like on this picture:



convinject
EXE
new_contact_data_targets.xml
XML
</>
SSH
contact_data_change.txt
TEXT
Document
enable, configure, contact info
enable, configure, contact info
enable, configure, contact info
192.168.0.166
192.168.0.18
192.168.0.19

According to common sense, acceptance criteria in first case is such: for a duration of reboot time (30-90 seconds) the switches should become unavailable (for instance not ping'able), and in seconds case: on each of Targets (192.168.0.18, 192.168.0.19, and 192.168.0.166) location information field should be changes to new value "*Contact Address has Changed to SAMPLE*".

Feel free to evaluate these simple text (*.TXT) and extended markup (*.XML) files, and to use the project under your network administrating tasks.

TP-LINK®			**Name:** Description - conversation injector
№	**Number, version**	**Date**	**Author**
1	Version 2.0	10 September 2016	Konstantin Mauch, Engineer

Table of contents