Learning Sliver C2 (01) - Tutorial / Installation

dominicbreuker.com/post/learning_sliver_c2_01_installation

Dominic Breuker

August 15, 2022

> This post is about how to install the Sliver C2 framework from BishopFox on a blank Kali Linux server. It is meant as the kickoff post for a series of tutorial posts on how to use Sliver, but targeting beginner users rather than experienced red team veterans.

Introduction

Recently, I developed some interest into red teaming and wanted to try out a few things. As you might know, all of that is no fun without a command and control (C2) framework. Unfortunately, most of the well-known ones are terribly expensive (Cobalt Strike or SCYTHE). Not good for a few personal experiments at home.

Fortunately though, there are plenty of open source solution out there. Check out the C2 matrix and you get more choices than you can handle. My short and slightly unstructured evaluation brought up sliver as a promising candidate. Build fully in Go, which compiles to all the platforms, actively maintained, tons of features. Sounds great overall.

It even provides a beginner's guide. Oh wait, the page says "TODO" (at the time of writing). If you are a beginner in those frameworks, the following series of blog posts may be for you. They'll be a detailed documentation of my endeavours with Sliver and may even become a tutorial.

Series Overview

Here is an overview of what's there:

Installation

This post covers basic installation from compiled binaries first, then demonstrates how to build from source. Building from source is what I recommend since it allows to modify things quickly if needed.

Sliver's architecture is client-server. Multiple clients, called "operators" or "players", connect to the same server and collaborate on an engagement. Accordingly, you'll have to set up the server, then distribute the client application along with credentials to all operators. To keep things easy, I'll use just one Kali VM. It runs the Sliver server as root. Each player can be set up as a dedicated Linux system user.

I recommend to install Sliver inside a virtual machine (VM). This will make it easy to later add other machines to the setup, such as a second VM serving as a target for attacks. All your VMs can run within a virtual network to simulate various environments.

Basic install from release binaries

Sliver provides an install script which gets you up and running in no time. You could run it with `curl https://sliver.sh/install|sudo bash`. If you don’t feel like piping remote scripts blindly to a root shell today then read on though. This installation was done manually on Kali Linux, but should work mostly the same on many other Linux distros.

First, you’ll want Sliver to cross-compile executables or DLLs for you. For it to be able to do that it needs MinGW. Install it with `apt-get install build-essential mingw-w64 binutils-mingw-w64 g++-mingw-w64`.

Second, get the client and server binaries. Sliver uses a client-server architecture to allow multiple operators to work together. The binaries can be found on GitHub. For example, this command will return download links for the latest versions:

```
┌──(root㉿kali)-[~]
└─# curl -s https://api.github.com/repos/BishopFox/sliver/releases/latest \
    | jq -r '.assets | .[] | .browser_download_url' \
    | grep -E '(sliver-server_linux|sliver-client_linux)$'
https://github.com/BishopFox/sliver/releases/download/v1.5.17/sliver-client_linux
https://github.com/BishopFox/sliver/releases/download/v1.5.17/sliver-server_linux
```

To download, I’ve used to following commands:

```
┌──(root㉿kali)-[~]
└─# wget -O /usr/local/bin/sliver-server \
    https://github.com/BishopFox/sliver/releases/download/v1.5.17/sliver-
server_linux && \
    chmod 755 /usr/local/bin/sliver-server
...
┌──(root㉿kali)-[~]
└─# wget -O /usr/local/bin/sliver \
    https://github.com/BishopFox/sliver/releases/download/v1.5.17/sliver-
client_linux && \
    chmod 755 /usr/local/bin/sliver
...
```

Assuming `/usr/local/bin/` is in your path, your sliver server should be available in the shell as `sliver-server` and the client as `sliver`.

Next, run `sliver-server unpack --force` to unpack all assets. This command installs a few files into the `~/.sliver` folder (see source code). Most importantly you get `~/.sliver/go`, a version of the Go toolchain dedicated to sliver. Go itself is used to compile implants, which are the C2 agents. Sliver also installs garble, a tool to obfuscate Go binaries.

Finally, I recommended to create a systemd service for the Sliver server. Put the following content into `/etc/systemd/system/sliver.service`, then `chmod 600 /etc/systemd/system/sliver.service`:

```
[Unit]
Description=Sliver
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=on-failure
RestartSec=3
User=root
ExecStart=/usr/local/bin/sliver-server daemon

[Install]
WantedBy=multi-user.target
```

Now, you can start the server with `systemctl start sliver`, which starts Sliver in daemon mode. If it worked, you should see it listening on port 31337 for connections from operators:

```
┌──(root㉿kali)-[~]
└─# netstat -antop | grep 31337
tcp6    0    0 :::31337    :::*    LISTEN    4706/sliver-server   off (0.00/0/0)
```

Operators authenticate to the server with mutual TLS. Client certificates can be issued by the server using the `operator` command. The following creates a new operator named `hacker1`, who connects to the server directly from localhost:

```
┌──(root㉿kali)-[~]
└─# sliver-server operator --name hacker1 --lhost localhost --save /tmp

┌──(root㉿kali)-[~]
└─# cat /tmp/hacker1_localhost.cfg  | jq
{
  "operator": "hacker1",
  "token": "4700ca860f06ed4e47ad50fc5a58ff22280272ccac437614667520ee31142c28",
  "lhost": "localhost",
  "lport": 31337,
  "ca_certificate": "-----BEGIN CERTIFICATE-----
\nMIIB2zCCAWGgAwIBAgIRAIvG5Z5ou/2uf+yHcTVoAgQwCgYIKoZIzj0EAwMwFDES\nMBAGA1UEAxMJb3
BlcmF0b3JzMB4XDTIyMDIwNzEwMzU0OFoXDTI1MDIwNjEwMzU0\nOFowFDESMBAGA1UEAxMJb3BlcmF0b3
JzMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAE\namtILty7fUUSnWetv5GTe+DCnIRw6S8Vn1DKYDrwDtXaF1
ZxLoX/etRDT4dpgfGi\nRam1Ful8BrbML5YjFzXQ5SAE1YA54RFd8QjWkjhhrQsBqBvRDnIJjp/xZU80V5
Hs\no3cwdTAOBgNVHQ8BAf8EBAMCAqQwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUF\nBwMCMA8GA1Ud
EwEB/wQFMAMBAf8wHQYDVR0OBBYEFCXcIOuKRs7lYrH6/BKEgfIK\n6PXDMBQGA1UdEQQNMAuCCW9wZXJh
dG9yczAKBggqhkjOPQQDAwNoADBlAjB4d7i1\nmYFSPlcTxZIJKuCym3Xy1pKwoRTrKdTMqeyzlfMu/Qcl
8DD+wXxslE5Hh3ECMQD0\nuMlQW3ponUaTXAKm/VpsPYQ97uUHz7BEdrz6uzUoWBBJVhiJXux1hgVKbr80
jXA=\n-----END CERTIFICATE-----\n",
  "private_key": "-----BEGIN EC PRIVATE KEY-----
\nMIGkAgEBBDBwNc+w1dtBeQ3Yc+bS5BC87os0ZWaqpiDvGapum1aBWGHysvZWqYad\nOcEvO0e1hPWgBw
YFK4EEACKhZANiAAS6JIiM9XB9RBFebA7CY6At7OSWhhzbiTso\nDltLDr4rbVkNO/9K08DY+TqZz13+eN
t7lADaHSkrNEok6E4MJ836Km0fRofwR7Sx\n8GNf+BTa7u6lHlnYDNwSHvpua4AK+RI=\n-----END EC
PRIVATE KEY-----\n",
  "certificate": "-----BEGIN CERTIFICATE-----
\nMIIBqjCCATCgAwIBAgIRAJsjV+qNzDit8smoBi15eSMwCgYIKoZIzj0EAwMwFDES\nMBAGA1UEAxMJb3
BlcmF0b3JzMB4XDTIxMDkyMzA5MzY1M1oXDTI0MDkyMjA5MzY1\nM1owEjEQMA4GA1UEAxMHaGFja2VyMT
B2MBAGByqGSM49AgEGBSuBBAAiA2IABLok\niIz1cH1EEV5sDsJjoC3s5JaGHNuJOygOW0sOvittWQ07/0
rTwNj5OpnPXf5423uU\nANodKSs0SiToTgwnzfoqbR9Gh/BHtLHwY1/4FNru7qUeWdgM3BIe+m5rgAr5Eq
NI\nMEYwDgYDVR0PAQH/BAQDAgWgMBMGA1UdJQQMMAoGCCsGAQUFBwMCMB8GA1UdIwQY\nMBaAFCXcIOuK
Rs7lYrH6/BKEgfIK6PXDMAoGCCqGSM49BAMDA2gAMGUCMEee92b/\nC4azRM1ZkKAlJtjIb3R1tGUNoIqI
jxr5JuwmnBiMZJPrGZaNWw/UFJVx1QIxAPHp\nwgtzb+uY3+XjB7yO9IrLeOV4sjdpGUjsj6lSTeWLn3c2
LSVXJm1KtL59+vLj5A==\n-----END CERTIFICATE-----\n"
}
```

This file contains the CA certificate used by the client to authenticate the server, a private key and certificate used for client authentication, and additionally a token, which is also required for client authentication (see source code for details about the authentication process). The CA certificate of the server uses `CN = operators` and accordingly, clients do not verify the host name (source).

You must copy this file into the folder `~/.sliver-client/configs` of the operator. In my case, I used the system user `kali` as the operator. Thus, the following accomplished the setup:

```
┌──(root㉿kali)-[~]
└─# mkdir -p /home/kali/.sliver-client/configs

┌──(root㉿kali)-[~]
└─# mv /tmp/hacker1_localhost.cfg /home/kali/.sliver-client/configs/

┌──(root㉿kali)-[~]
└─# chown -R kali:kali /home/kali/.sliver-client/ && chmod 600 /home/kali/.sliver-
client/configs/hacker1_localhost.cfg

┌──(root㉿kali)-[~]
└─# ls -la /home/kali/.sliver-client/configs
total 12
drwxr-xr-x 2 kali kali 4096 Jun 30 11:45 .
drwxr-xr-x 3 kali kali 4096 Jun 30 11:44 ..
-rw------- 1 kali kali 1845 Jun 30 11:36 hacker1_localhost.cfg
```

Now switch to user `kali` and run the client. If everything worked, you should be able to connect and see a screen that looks roughly as below. Executing the command `version` in the interactive prompt should print the server and client version:

```
┌──(kali㉿kali)-[~]
└─$ sliver
Connecting to localhost:31337 ...

    ███████╗██╗     ██╗██╗   ██╗███████╗██████╗
    ██╔════╝██║     ██║██║   ██║██╔════╝██╔══██╗
    ███████╗██║     ██║██║   ██║█████╗  ██████╔╝
    ╚════██║██║     ██║╚██╗ ██╔╝██╔══╝  ██╔══██╗
    ███████║███████╗██║ ╚████╔╝ ███████╗██║  ██║
    ╚══════╝╚══════╝╚═╝  ╚═══╝  ╚══════╝╚═╝  ╚═╝

All hackers gain vigilance
[*] Server v1.5.17 - 814670dc6d023f290fefd3e0fd7e0c420f9bb2e8
[*] Welcome to the sliver shell, please type 'help' for options

sliver > version

[*] Client v1.5.17 - 814670dc6d023f290fefd3e0fd7e0c420f9bb2e8 - linux/amd64
    Compiled at 2022-06-28 20:49:18 +0200 CEST
    Compiled with go version go1.18.3 linux/amd64


[*] Server v1.5.17 - 814670dc6d023f290fefd3e0fd7e0c420f9bb2e8 - linux/amd64
    Compiled at 2022-06-28 20:49:17 +0200 CEST
```

Build from source

Building sliver from source is easy. You only have to follow the steps described in their wiki. For the sake of completeness, here is a description of the process at the time of writing.

To build from source, you need Go installed. Do that with `apt-get install golang`. Then get the source from GitHub. Clone the repository anywhere onto your machine with `git clone https://github.com/BishopFox/sliver` and cd into its root folder.

Now you have to download all assets using the `go-assets.sh` script. It produces a lot of output while downloading all static Go assets. A successful run looks roughly like this:

```
┌──(root㉿kali)-[~/github/sliver]
└─# ./go-assets.sh
-----------------------------------------------------------------
/tmp/tmp.tch6xLauE8 (Output: /root/github/sliver/server/assets/fs)
-----------------------------------------------------------------
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
 76  137M   76  105M    0     0  11.3M      0  0:00:12  0:00:09  0:00:03 11.6

...

clean up: /tmp/tmp.tch6xLauE8

[*] All done
```

Now just run `make` to build server and client binaries for Linux. Cross-compiling to other operating systems is also supported (see the wiki). After a successful build, you find the files and `./sliver-server` and `./sliver-client`. To demonstrate building a custom version, I've checked out commit `140c47e163541340295d3f2b530fe800eccf7156` here:

```
┌──(root㉿kali)-[~/github/sliver]
└─# git checkout 140c47e163541340295d3f2b530fe800eccf7156
Note: switching to '140c47e163541340295d3f2b530fe800eccf7156'.

You are in 'detached HEAD' state. You can look around, make experimental
...

┌──(root㉿kali)-[~/github/sliver]
└─# make
...

┌──(root㉿kali)-[~/github/sliver]
└─# file sliver-server
sliver-server: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=3c42b0bbc6d3077929e87f3c1c170d680eb6f8b2, for GNU/Linux 3.2.0,
stripped

┌──(root㉿kali)-[~/github/sliver]
└─# file sliver-client
sliver-client: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=7add139c2d119c72b533778f08897ed575b3f15e, for GNU/Linux 3.2.0,
stripped
```

Now we can put the new binaries in place. Make sure to stop the server before that, then start again after moving binaries:

```
┌──(root㉿kali)-[~/github/sliver]
└─# systemctl stop sliver

┌──(root㉿kali)-[~/github/sliver]
└─# mv sliver-client /usr/local/bin/sliver

┌──(root㉿kali)-[~/github/sliver]
└─# mv sliver-server /usr/local/bin/sliver-server

┌──(root㉿kali)-[~/github/sliver]
└─# systemctl start sliver
```

Connecting from the `kali` user with the new client works like a charm. In my case, it was not necessary to create new operator configs. Of course, this is not generally true and has to be tested case by case. Connecting to the server and running the `version` command confirms the new version is used:

```
┌──(kali㉿kali)-[~]
└─$ sliver
Connecting to localhost:31337 ...

.------..------..------..------..------..------.
|S.--. ||L.--. ||I.--. ||V.--. ||E.--. ||R.--. |
| :/\: || :/\: || (\/) || :(): || (\/) || :(): |
| :\/: || (__) || :\/: || ()() || :\/: || ()() |
| '--'S|| '--'L|| '--'I|| '--'V|| '--'E|| '--'R|
`------'`------'`------'`------'`------'`------'

All hackers gain renown
[*] Server v1.5.16 - 140c47e163541340295d3f2b530fe800eccf7156
[*] Welcome to the sliver shell, please type 'help' for options

sliver > version

[*] Client v1.5.16 - 140c47e163541340295d3f2b530fe800eccf7156 - linux/amd64
    Compiled at 2022-06-30 12:07:30 +0200 CEST
    Compiled with go version go1.18.3 linux/amd64


[*] Server v1.5.16 - 140c47e163541340295d3f2b530fe800eccf7156 - linux/amd64
    Compiled at 2022-06-30 12:07:30 +0200 CEST
```