【译】Metasploit：搭建开发环境

王一航 / 2018-06-15 14:50:14 / 浏览数 5712 新手 入门资料 顶(0) 踩(0)

- 原文地址：https://github.com/rapid7/metasploit-framework/wiki/Setting-Up-a-Metasploit-Development-Environment
- 作者：Metasploit Community
- 译者：王一航 & Google 2018-06-15
- 校对：王一航 2018-06-15

译者注：

> 在国内建议使用 Gem 的 国内镜像源
>
> ```
> gem sources --add https://gems.ruby-china.org/ --remove https://rubygems.org/
> bundle config mirror.https://rubygems.org https://gems.ruby-china.org
> ```

本文将着重讲述如何搭建 Metasploit Framework 的开发环境以便于贡献者可以高效迅速地进行贡献。
如果你只是想对 Metasploit Framework 进行授权的合法的渗透测试，我们建议直接下载商业版本安装包或者
开源版本安装包，因为我们已经在这些包中为您解决了所有的依赖问题。商业版本安装包还包括可选的可以更新到 Metasploit Pro
的部分并且会每周更新两次，而开源版本则会在每天晚上进行更新。

如果你正在使用 Kali Linux，Metasploit 已经预先安装好了。可以参考这个指南，该指南提供了如何使用提供好的 Metasploit-Framework
软件包以及配置数据库。

如果你想参与开发或者对 Metasploit 进行贡献，那么本文很适合你，继续读下去吧！本文应该可以让你使用任何基于 Debian 的 Linux 操作系统进行工作。

那么废话少说，开始吧！

前提

- 你已经有了一个基于 Debian 的 Linux 环境
- 你有一个非 Root 的用户，本文中我们使用 `msfdev` 这个用户
- 你有一个 GitHub 账号，并且关联了一个 SSH 密钥 到你的 GitHub 账号。

安装基础开发软件包

```
sudo apt-get -y install \
 autoconf \
 bison \
 build-essential \
 curl \
 git-core \
 libapr1 \
 libaprutil1 \
 libcurl4-openssl-dev \
 libgmp3-dev \
 libpcap-dev \
 libpq-dev \
 libreadline6-dev \
 libsqlite3-dev \
 libssl-dev \
 libsvn1 \
 libtool \
 libxml2 \
 libxml2-dev \
 libxslt-dev \
 libyaml-dev \
 locate \
 ncurses-dev \
 openssl \
 postgresql \
 postgresql-contrib \
 wget \
 xsel \
```

```
zlib1g \
zlib1g-dev
```

请注意：目前我们还没有安装 Ruby，但是我们将会在后续步骤中进行安装。

Fork 并克隆 Metasploit-Framework 仓库

你可以参考这篇 GitHub 提供的关于 forking 的指南，但是其实你只需要点击仓库主页右上角的 “Fork” 按钮就可以完成 Fork 操作了。

克隆

一旦你已经在 GitHub 上 Fork 了该仓库，现在就可以将该仓库拉取到本地的开发机了。
同样，你也可以参考 GitHub 提供的这篇关于 cloning 的指南

```
mkdir -p $HOME/git
cd $HOME/git
git clone git@github.com:YOUR_USERNAME_FOR_GITHUB/metasploit-framework
cd metasploit-framework
```

设置 Git 仓库的 Upstream

首先，如果你打算使用最新的上游（upstream）来更新您的本地克隆，那么您需要跟踪（track）它
在 `metasploit-framework` Checkout 的目录中，运行如下命令：

```
git remote add upstream git@github.com:rapid7/metasploit-framework.git
git fetch upstream
git checkout -b upstream-master --track upstream/master
```

现在你已经有了一个指向了 upstream 的分支（也就是 GitHub 上的 `rapid7/Metasploit-Framework` 这个仓库），与你自己 Fork 得到的仓库不同（指向 `origin/master` 的源 `master` 分支）。
你或许会发现，拥有 `upstream-master` 和 `master` 两个不同的分支是非常方便的（尤其是当你是一个 Metasploit committer
的时候，因为这会将避免你意外地将代码直接推送到 `rapid7/master` 分支上）

安装 RVM（译者注：Ruby Version Manager）

大部分的 Linux 发行版并不会提供可预测频率的最新版本的 Ruby，因此我们使用 RVM （Ruby Version Manager）进行 Ruby
版本的管理，你可以参考官网获取相关信息，看起来很不错。当然也有些人比较青睐 rbenv，你也可以参考 rbenv 教程。

但是你就得自己保证你有合适版本的 Ruby。因为大部分和 Commiters 使用 RVM，所有本篇教程中，我们也会就 RVM 进行接下来的操作。

首先，你需要为获取 RVM 发行版的签名文件

```
curl -sSL https://rvm.io/mpapis.asc | gpg --import -
```

然后，安装 RVM：

```
curl -L https://get.rvm.io | bash -s stable
```

上面的命令是直接使用管道将所有的命令定向到了 Bash 中，但是这可能会引起一些比较敏感的问题。因此有一个更安全的方法如下：

```
curl -o rvm.sh -L https://get.rvm.io
less rvm.sh # Read it and see it's all good
cat rvm.sh | bash -s stable
```

当上面的操作都成功完成后，就可以修复你的终端（Terminal）来使用 RVM 版的 Ruby 了

```
source ~/.rvm/scripts/rvm
cd ~/git/metasploit-framework
rvm --install $(cat .ruby-version)
```

最后，安装 `bundler` 这个 gem 包来获取其他所有需要的的 gem 包：

```
gem install bundler
```

为 Gnome Terminal 配置使用 RVM

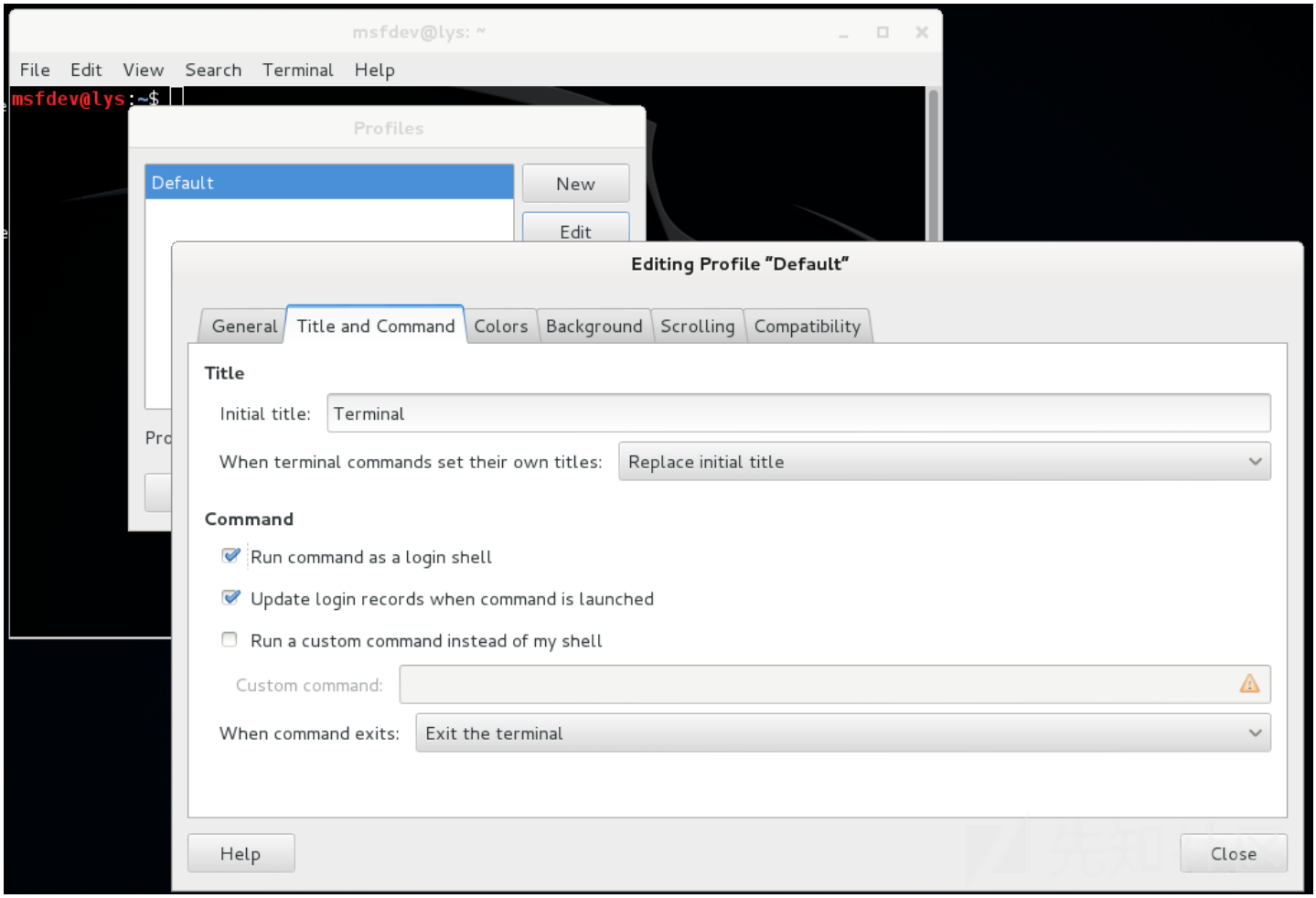
Gnome Terminal 真是个混蛋，默认情况下它并没有将你的 Shell 作为 Login shell，所以没有对 tweak 进行配置的话， RVM 将无法正常工作。如下：

> Navigate to Edit > Profiles > Highlight Default > Edit > Title and Command

选中如下复选框：

[√] Run command as a login shell.

看起来应该像下图这样，具体可能根据不同的 Gnome 版本有微小差异：



最后，你就可以运行在 .ruby-version 这个文件中指定的特定的 Ruby 版本了。

```
ruby -v
```

如果你发现当前正在运行的 Ruby 版本仍然不是在 `.ruby-version` 这个文件里面定义的版本的话，你或许需要重启你的终端。如果你初始安装的 RVM 不具备如下功能的话，那么请确保你已经将 rvm 添加到了你的 Termianl 的启动脚本中。

```
echo '[[ -s "$HOME/.rvm/scripts/rvm" ]] && source "$HOME/.rvm/scripts/rvm"' >> .bashrc
```

安装打包组件 Gems

Metasploit 依赖许多 gems（Ruby 库）。因为你正在使用 RVM，因此你可以直接在本地安装它们而不用考虑与 Debian 打包好的 Gems 产生冲突，感谢 Bundler 的黑科技。

先进入你的 Checkout 的根目录，然后运行：

```
cd ~/git/metasploit-framework/
bundle install
```

一两分钟后，你就可以开始你的 Metasploit 之旅了，在你的 Checkout 目录，输入：

```
./msfconsole
```

沐浴在 Metasploit 启动带来的光辉中吧！（译者注：原文 And bask in the glory that is a functioning source checkout）
第一次运行的时候，将会顺便创建一个 `~/.msf4` 的目录

```
msfdev@lys:~/git/metasploit-framework$ ./msfconsole
   [*] Starting the Metasploit Framework console.../

                  _---------.
              .' #######   ;."
```

```
     .---,.    ;@             @@`;   .---,..
   ." @@@@@'.,'@@            @@@@@',.'@@@@ ".
   '-.@@@@@@@@@@@@          @@@@@@@@@@@@@ @;
     `.@@@@@@@@@@@@        @@@@@@@@@@@@@@ .'
       "--'.@@@  -.@        @ ,'-   .'--"
             ".@' ; @       @ `.  ;'
               |@@@@ @@@     @    .
               ' @@@ @@   @@    ,
                 `.@@@@    @@   .
                   ',@@     @   ;           _____________
                   (   3 C    )     /|___ / Metasploit! \
                   ;@'. __*__,."    \|--- \_____________/
                     '(.,...."/

      =[ metasploit v5.0.0-dev-26bf96b                  ]
+ -- --=[ 1744 exploits - 1001 auxiliary - 302 post       ]
+ -- --=[ 536 payloads - 40 encoders - 10 nops            ]
+ -- --=[ ** This is Metasploit 5 development branch **   ]

   msf > ls ~/.msf4
   [*] exec: ls ~/.msf4

   history
   local
   logos
   logs
   loot
   modules
   plugins
   msf > exit
```

唉，我们似乎还没有成功配置数据库来使用这些狂暴的 Hacking，要解决？简单！

配置 PostgreSQL

Kali Linux 已经自带了 PostgreSQL，所以我们可以直接使用。在 Ubuntu 和其他基于 Debian 的发行版上，所有的配置应该都是类似的，并且可以很好的工作。我们假设使用别的发行版的人已经安装好了 `PostgreSQL`，TLDR 也确保数据库能在系统启动时启动。

启动数据库

TLDR（以 msfdev 用户的身份运行）

```
echo 'YOUR_PASSWORD_FOR_KALI' | sudo -kS update-rc.d postgresql enable &&
echo 'YOUR_PASSWORD_FOR_KALI' | sudo -S service postgresql start &&
cat <<EOF> $HOME/pg-utf8.sql
update pg_database set datallowconn = TRUE where datname = 'template0';
\c template0
update pg_database set datistemplate = FALSE where datname = 'template1';
drop database template1;
create database template1 with template = template0 encoding = 'UTF8';
update pg_database set datistemplate = TRUE where datname = 'template1';
\c template1
update pg_database set datallowconn = FALSE where datname = 'template0';
\q
EOF
sudo -u postgres psql -f $HOME/pg-utf8.sql &&
sudo -u postgres createuser msfdev -dRS &&
sudo -u postgres psql -c \
 "ALTER USER msfdev with ENCRYPTED PASSWORD 'YOUR_PASSWORD_FOR_PGSQL';" &&
sudo -u postgres createdb --owner msfdev msf_dev_db &&
sudo -u postgres createdb --owner msfdev msf_test_db &&
cat <<EOF> $HOME/.msf4/database.yml
# Development Database
development: &pgsql
 adapter: postgresql
 database: msf_dev_db
 username: msfdev
```

```
 password: YOUR_PASSWORD_FOR_PGSQL
 host: localhost
 port: 5432
 pool: 5
 timeout: 5

# Production database -- same as dev
production: &production
 <<: *pgsql

# Test database -- not the same, since it gets dropped all the time
test:
 <<: *pgsql
 database: msf_test_db
EOF
```

在 Kali Linux
上，PostgreSQL（或者其他需要监听端口的服务）默认情况下不会开启。这是一个很好的安全资源防御措施，但是如果你想要让他们随时运行的话（译者注：开机启动），

```
update-rc.d postgresql enable
```

然后，切换到 Postgres 用户来执行一些小型数据库的运维工作来修复默认编码错误（该有用的信息由 @ffmike's gist 提供）

```
sudo -sE su postgres
psql
update pg_database set datallowconn = TRUE where datname = 'template0';
\c template0
update pg_database set datistemplate = FALSE where datname = 'template1';
drop database template1;
create database template1 with template = template0 encoding = 'UTF8';
update pg_database set datistemplate = TRUE where datname = 'template1';
\c template1
update pg_database set datallowconn = FALSE where datname = 'template0';
\q
```

创建一个数据库用户 **`msfdev`**

还是在 postgres 这个用户的权限下：

```
createuser msfdev -dPRS              # Come up with another great password
createdb --owner msfdev msf_dev_db   # Create the development database
createdb --owner msfdev msf_test_db  # Create the test database
exit                                 # Become msfdev again
```

创建 database.yml

现在切换到你自己的用户，按照如下模板创建文件 `$HOME/.msf4/database.yml`：

```
# Development Database
development: &pgsql
 adapter: postgresql
 database: msf_dev_db
 username: msfdev
 password: YOUR_PASSWORD_FOR_PGSQL
 host: localhost
 port: 5432
 pool: 5
 timeout: 5

# Production database -- same as dev
production: &production
 <<: *pgsql

# Test database -- not the same, since it gets dropped all the time
test:
 <<: *pgsql
 database: msf_test_db
```

当你下次启动 `./msfconsole`，一个用于开发的数据库将会被创建，可以使用以下命令检查：

```
./msfconsole -qx "db_status; exit"
```

运行规格说明

我们使用 rspec 来进行大部分的框架测试。首先要确保下面的命令可以正常工作：

```
rake spec
```

上述命令运行完成后，你应该会看到总共运行了超过 9000 个测试用例，其中大部分被标记为绿色，很少一部分为黄色，不会出现红色的错误结果。

配置 Git

TLDR（以 msfdev 用户的身份运行）

```
cd $HOME/git/metasploit-framework &&
git remote add upstream git@github.com:rapid7/metasploit-framework.git &&
git fetch upstream &&
git checkout -b upstream-master --track upstream/master &&
ruby tools/dev/add_pr_fetch.rb &&
ln -sf ../../tools/dev/pre-commit-hook.rb .git/hooks/pre-commit &&
ln -sf ../../tools/dev/pre-commit-hook.rb .git/hooks/post-merge &&
git config --global user.name   "YOUR_USERNAME_FOR_REAL_LIFE" &&
git config --global user.email  "YOUR_USERNAME_FOR_EMAIL" &&
git config --global github.user "YOUR_USERNAME_FOR_GITHUB"
```

配置 Pull Ref

如果你想要在命令行上轻松访问上游（upstream）的 Pull Requests，那么您需要添加相关的 fetch reference
到`.git/config`中。可以通过以下方式轻松完成：

```
tools/dev/add_pr_fetch.rb
```

这将会添加所有相关的引用到你所有的 Git Remotes 中，包括你自己的。
现在你就可以按照如下命令来进行操作了：

```
git checkout fixes-to-pr-1234 upstream/pr/1234
git push origin
```

- GitHub 提供了一种 更简单的方法

上面所做的一切可以让你查看别人的 Pull Request（PR），并进行修改，然后发布到你自己的 Fork
的某一个分支上。这反过来会让你帮助其他人的PR进行修复或补充。

保持同步

请绝对 不要 直接直接提交代码到 Master
分支。请将你所有的改动放置在一个新的分支，然后对分支进行合并。这样的做法可以很容易就能保持代码同步，并且你的本地修改永远也不会丢失。

同步到 upstream/master

再简单不过了。

```
git checkout master
git fetch upstream
git push origin
```

这可以让 Pull Requests 与 Master 分支保持同步。除非你遇到合并冲突，否则不需要经常这样做。
而且当你最终解决合并冲突之后，你可能会想使用 `--force` 参数来将本地修改重新同步到分支，因为你的提交历史将会在 rebase 之后变得不同。

> 对 rapid7/master 进行强制 Push 是绝对不可以的。但是可以在别的正在开发的分支，重写一些历史并不会构成联邦犯罪。

Msftidy 工具

为了对任何你正在编写的新模块进行检查，你需要在 Commit 之前，以及 Merge 之后进行 HOOK 以使用我们的代码检查工具 `msftity.rb`
进行检查，所以按照如下命令配置 Git HOOK：

```
cd $HOME/git/metasploit-framework
ln -sf ../../tools/dev/pre-commit-hook.rb .git/hooks/pre-commit
ln -sf ../../tools/dev/pre-commit-hook.rb .git/hooks/post-merge
```

配置你的身份

最后，如果你想为 Metasploit 做贡献的话（译者注：事实上所有开源项目都应该这样做），你至少需要按照如下命令配置你的名字与邮箱：

```
git config --global user.name   "YOUR_USERNAME_FOR_REAL_LIFE"
git config --global user.email  "YOUR_USERNAME_FOR_EMAIL"
git config --global github.user "YOUR_USERNAME_FOR_GITHUB"
```

注意你的邮箱地址必须和你的 GitHub 注册的邮箱相互匹配。

对 Commits 签名

我们热爱对 Commits 进行签名，主要是因为我们 [害怕另一种选择（译者注：原文 we're terrified of the alternative）]()。关于如何签名的详细信息可以[在此获取]()。请注意姓名和邮件地址必须完全匹配上签名密钥上的信息。我们鼓励贡献者签名自己的 Commits，因为 Metasploit 的 Committers 在 [合并 Pull Request（译者注：Land Pull Request）]() 的时候也需要进行签名。

有用的别名

如果不再配置一些有用的让生活变得更美好的别名的话，那么配置开发环境就不能算完成。

覆盖已经安装过的 `msfconsole`

作为一个开发者，你可能会一不小心使用到已经安装过的 Metasploit `msfconsole`，由于 RVM 会处理不同的 Ruby 版本和 gemsets 包的原因，可能会出现一些非预期的问题，所以你可以配置一下 Shell 的别名：

```
echo 'alias msfconsole="pushd $HOME/git/metasploit-framework && ./msfconsole && popd"' >> ~/.bash_aliases
```

如果你既想使用开发版的 Metasploit 也想使用安装包，那么使用两个不同的用户可能是一个更好的选择（译者注：感觉这样很麻烦，还不如使用 `./msfconsole` 和 `msfconsole` 来的方便）

修改命令提示符使其支持显示 Ruby/Gemset/Branch

下面的脚本可以很容易让你知道你目前所在的位置/环境（译者注：例如 Ruby 版本，Git 仓库的分支等），你可以将如下脚本放在 `~/.bash_aliases`

```
function git-current-branch {
   git branch 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/(\1) /'
}
export PS1="[ruby-\$(~/.rvm/bin/rvm-prompt v p g)]\$(git-current-branch)\n$PS1"
```

Git 命令别名

与传统 Shell 别名很相似，Git 自己本身也有别名功能，既可以在 `$HOME/.gitconfig` 中被配置，也可以在 `repo-name/.git/config` 中被配置。
如下为一些很有用的别名配置：

```
[alias]
# ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
nicelog = log --pretty=format:'%Cred%h%Creset -%Creset %s %Cgreen(%cr) %C(bold blue)<%aE>%Creset [%G?]'

# ■■■■■
# ■ merge ■ commit ■■■■■■■■■■■■ commit message
m = merge -S --no-ff --edit
c = commit -S --edit

# ■■■■■ Git Blame ■■■■■■■■■■■■■■■■■■
#■■■■■git blame ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ blame■■■■■■■■■■■■ BUG ■■■■■■
b= blame -w

# ■■■■■■■■■■■■■ Git Stash ■■■■■■■■■■■■■■■■■■■■■■■
temp = !"git branch -D temp; git checkout -b temp"

# ■■■■■■■■■■■■■■■ Pull Request■■■■■■■■■■■■■■■■■■■■■■■■
# ■■ @kernelsmith ■■■■■■■
pr-url =!"xdg-open https://github.com/$(git config github.user)/$(basename $(git rev-parse --show-toplevel))/pull/new/$1:$2...
```

以上就是全部内容啦！如果你比较关心 Pull Request 的话，那么你看还需要配置 [Aliases]() 和 PGP key 来 [签名你的 Commits]()。
但是除此之外，你已经准备好啦！

如果您在操作的时候遇到任何错误，遗漏，或者发现了更好的方法，无论如何，请开启一个 [GitHub Issue]())，我们将会看到您的反馈并更新 HOWTO 文档。

特别鸣谢 @kernelsmith 与 @corelanc0d3r ，感谢他们对此开发环境文档指南的宝贵帮助和反馈！

点击收藏 | 1 关注 | 1

上一篇：一次红队之旅 下一篇：APT 27攻击中亚政府数据中心，...

1. 1 条回复

不能忍 2018-08-27 00:10:31

您好,我在配置RVM环境的时候出了点问题,能不能帮我看一看,
谢谢
Error running 'requirements_debian_update_system ruby-2.5.1',
please read /usr/local/rvm/log/1535299632_ruby-2.5.1/update_system.log
Requirements installation failed with status: 100.
Gemset '' does not exist, 'rvm ruby-2.5.1 do rvm gemset create ' first, or append '--create'.

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录