SSH, GDB and Version Control Systems

Tools for the modern programmer

James Swaro
Department of Computer Science
Ohio University

April 13, 2012

Outline

Outline

Overview

- **S**ecure **SH**ell.
- Replaces rtools suite(insecure, plaintext)
- Provides a terminal, etc. . .

```
usage: ssh [-1246AaCfgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-e escape_char] [-F configfile]
           [-I pkcs11] [-i identity_file]
           [-L [bind_address:]port:host:hostport]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-R [bind_address:]port:host:hostport] [-S ctl_path]
           [-W host:port] [-w local_tun[:remote_tun]]
           [user@]hostname [command]
```

Outline

Aspects of the tool

Things you can do with SSH

- remote execution of applications and scripts
- port forwarding/tunneling
- secure transfer of information

Common flags:

- -X (allows X11 forwarding, allowing GUI programs to be run remotely and displayed locally)
- -C (compression, decreasing the size of packets)
- -D,-L (port forwarding, allowing for connections to originate from the source or destination, tunneling traffic)

Aspects of the tool

How can I make it better?

- Public key authentication
- Configuration files

Outline

Public key authentication

Public key authentication is the act of using a preshared RSA or DSA key to authenticate an SSH session instead of using a password.

- More secure than a password.
- Can be used with password as well
- Easy to do . . . I'll show you

First you need to generate a private and public key. . .

```
test@ds9 ~ $ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/test/.ssh/id_rsa):
Created directory '/home/test/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/test/.ssh/id_rsa.
Your public key has been saved in /home/test/.ssh/id_rsa.pub.
```

Passphrases are not required but recommended for additional security.

To log in with the key, the contents of the public key file,

`id_rsa.pub`

must be added to the

`/.ssh/authorized_keys`

file on any computer you want to log in to using the public key.
Private key stays on the local machine.
Why bother?

- Easy
- Passwordless (when public key is on the receiving machine and PKA is allowed)
- Secure (unlikely that someone will crack your password or your private key)

Also useful when you don't want to allow password based login to a secure system.

Outline

sshconfig

The ssh config file, */.ssh/config*, is a catch-all local config file. Containing:

- Host specific configurations
- User configurations for hosts
- Host aliases
- -o default options per host
- and more. . .

Simple Config

```
cat ~/.ssh/config

Host p1
HostName p1.eecs.ohio.edu
User jswaro

Host leo
HostName wolf359.it.cx
Port 5800
User jamesswaro

Host morpheus
HostName morpheus.dtnbone.ocp.ohiou.edu
User dtnbone

Host bin00001
HostName bin00001.eecs.ohio.edu
User jswaro

Host bin00010
HostName bin00010.eecs.ohio.edu
User jswaro
```

What is the point?

Without /.ssh/config ...

```
jswaro@ds9 ~ $ ssh -p 5800 jamesswaro@wolf359.it.cx
```

With

```
jswaro@ds9 ~ $ ssh leo
```

Outline

Overview

GDB, the GNU Project debugger, allows you to see what is going on ‘inside’ another program while it executes – or what another program was doing at the moment it crashed.

To debug a program, like vdump

gdb vdump

Getting Help

- Type:

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
```

Typing to GDB

- Uses emacs command line editing

```
control-p  previous command
control-n  next command
control-f  forward a character
control-b  back a character
control-k  kill the rest of the line
control-a  beginning of line
control-e  end of line
control-d  delete a character
control-s  forward search for command
control-r  backward search for command
```

Other characters “auto insert”

Outline

Running GDB

To run the program:

```
run
```

To run the program with args:

```
run -f thisfile thatarg
```

To run the program with i/o redirection:

```
run < infile > outfile
```

Runtime arguments persist between commands so it isn't necessary to retype them if you want to retype the program. Infact, if a command is not supplied to gdb, it will run the last command when the user presses return/enter.

Printing

You can use most C/C++ formats for printing

- print x
- print x+2
- print ptr
- print *ptr
- print ptr->next
- print/x ptr->next
- print func(2,3,4)
- print func(2,3,4)+5

You can also change variables:

set x = 5

Listing the program

List “next screen”

```
Program received signal SIGSEGV, Segmentation fault.
0x00000000004006b3 in badStuff (arr=0x601010) at index.cc:45
45 arr[i][j] = 0;
(gdb) list
40 //Clear address array
41 for(int i = 0; i <= ADDR; ++i)
42 {
43 for(int j = 0; j < SH; ++i)
44 {
45 arr[i][j] = 0;
46 }
47 }
48 return;
49 }
```

List starting at a particular line

list 107

List starting at a particular line by file

list main.c:283

Stack Frames, Where am I?

```
(gdb) where
#0  0x00000000004006b3 in badStuff (arr=0x601010) at index.cc:45
#1  0x000000000040064a in func1 () at index.cc:29
#2  0x00000000004005e8 in main (argc=1, argv=0x7fffffffe978) at index.cc:12
```

Useful stack commands:

- up
- down
- frame N
- info locals

Breakpoints

Setting breakpoints by function:

```
break main
```

Setting breakpoints by line:

```
break 10
```

With multiple files:

```
break main.c:107
break filter.c:26
...
break parse_arg.c:381
```

Listing or deleting breakpoints

```
info breakpoints
delete
delete 3
```

Outline

Other useful commands

control-c	stop the program
continue	start the program running from where you stopped it
finish	run until selected stack frame returns
step	execute until another line is reached
step N	do it N times
next	execute line and including function calls
set print pretty	“prettier” printing of data structures
wh	I’ll show you. . .

Shorthand command work as long as they are unambiguous

- “c” is continue
- “n” is next

Other interesting things. . .

You can attach GDB to a program that is already running.

You can also run shell commands from inside GDB.

You can remake your program from inside GDB.

GDB can be run inside of emacs/xemacs.

```
jswaro@ds9:~$ gdb someprogram
(gdb) shell ps -e | grep someprogram
 9570 pts/0    00:05:27 someprogram
(gdb) attach 9570
Attaching to process 9570
...
0x00000000004006a4 in addEntry (entry=0x7fffe7265530, value=0x10c3050) at loop.cc:51
51 while( it->next )
(gdb)
```

Summary

GDB is a great tool that will save you hours of time when debugging almost any problem.

Good Tools

- Version Control Systems(git, Mercurial, svn, etc. . .)
- valgrind/discover
- Meld (GUI diff utility)

Acknowledgements

Thanks go out to Dr. Ostermann for his help with the GDB section of this presentation. Most of the content for these slides came directly from his own GDB presentation.