Ruby Security

And several advanced features

Running other Programs

Using the `system` method to execute the Windows `date` command

```
system "date /t"
```

or using *backticks*

```
`date /t`
```

A Simple Interpreter

The `eval` method evaluates (executes) the code passed to it and returns the result

```
while  "q" != x = gets.chomp
   puts "=> #{eval(x)}"
end
```

Read – Eval – Print Loop

Safely Handling Data

All external data is dangerous!

- Command line
- External files
- Form fields on a web page

Example:

Suppose the user input to the simple interpreter was

```
`del C:\*.* /s`
```

Safely Handling Data

Ruby scripts can be made safer by marking all external data as *tainted*

Examples

```
s = "Hi Ruby"
s.tainted?   # false
x = 250
x.tainted?   # false
a1 = [s, x]
a1.tainted?  # false
f = File.open("somefile").readlines.first
f.tainted?   # true
a2 = [f, s, x]
a2.tainted?  # false
```

Regular Expressions and Security

Regular expressions often used to validate user input

Example: Forms on web pages can be used for several common attacks

- SQL injection
- Cross site scripting

Safely Handling Data

Blocking execution of operating system commands

```
while cmd = gets
  cmd.untaint if not cmd =~ /[`]/
  next if cmd.tainted?
  puts "=> #{eval(cmd)}"
end
```

Safe Levels

Ruby supports specifying what features to make available and how it should deal with tainted data

Safe levels are set by assigning a value to $SAFE

Safe Levels

0	No checking of the use of externally supplied (tainted) data is performed. This is Ruby's default mode.
>= 1	Disallow the use of tainted data by potentially dangerous operations.
>= 2	Prohibit the loading of program files from globally writable locations.
>= 3	All newly created objects are considered tainted
>= 4	Partitions the running program in two. Nontainted objects may not be modified. Typically, this will be used to create a sandbox: the program sets up an environment using a lower $SAFE level, then resets $SAFE to 4 to prevent subsequent changes to that environment.

Safe Levels

```
$SAFE = 1
cmd = gets
puts "=> #{eval(cmd)}"
```

Script will terminate with a SecurityError