GuardRails: A (Nearly) Painless Solution to Insecure Web Applications

Jonathan Burket, Patrick Mutchler, Michael Weaver, Muzzammil Zaveri
University of Virginia

GuardRails is a source-to-source tool that uses annotations to produce secure Ruby on Rails applications with minimal input from the developer.

Purpose:

As web application security becomes more important and the number of security threats grows, developers must write large amounts of situation specific security code. This makes code difficult to read and bugs easy to insert. We propose GuardRails, a lightweight extension to the Ruby on Rails web application framework that makes it easy to develop secure web applications without security expertise.

```
xss_terminate :except => [:author_name], :sanitize => [:title],
:html5lib_sanitize => [:body]
      verify :method => :post, :only => [:transfer],
      :redirect_to => {:action => :list}
```

Using GuardRails:

To use GuardRails, a developer writes typical Ruby on Rails code and adds security annotations. These annotations describe data invariants that can be checked dynamically by the application. Data invariants range from permissions to forced sanitization and data origin. Annotations use a syntax unique to GuardRails that, when run through the Ruby interpreter, gets ignored as comments so adding GuardRails annotations cannot break existing code.

```
# @(sanitize:title, html5lib_sanitize:body)
Class Foo
      # @(send_only_post)
      Class Bar
```

Data Policies

Rules for viewing or modifying data are linked directly to the data itself

Data policies allow for rules and regulations to be attached to specific types of data. Data types that posses such policies will ensure that the rules and regulations have been met when accessed(edit, modification etc.…). In addition, the use of data policies will ensure that developers no longer have to write convoluted access logic statements. Take for instance the example shown below

```
# @(edit:price:admin)
Boombox = Product.new

#success
Boombox.address = "123 CandyLand Lane"
Boombox.save

"Success: Address Changed:

#fail (non-administrator)
Boombox.price = "$1"
Boombox.save
```

Cannot change price!

```
# @(edit:modify_once)
 p1 = Project.new

#success
p1.title = "Hello World"
p1.save

#fails
p1.title = "Real Title"
p1.save
```

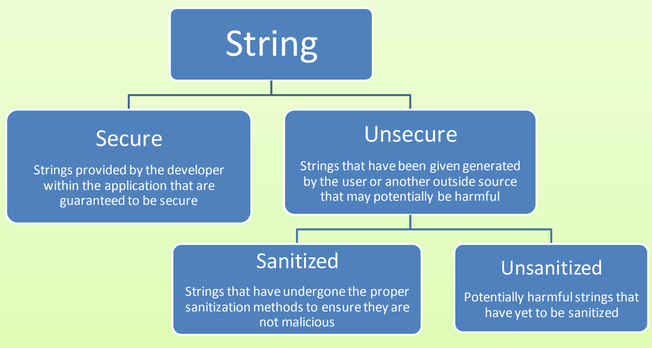
Total number of modifications surpassed!

Security Typing

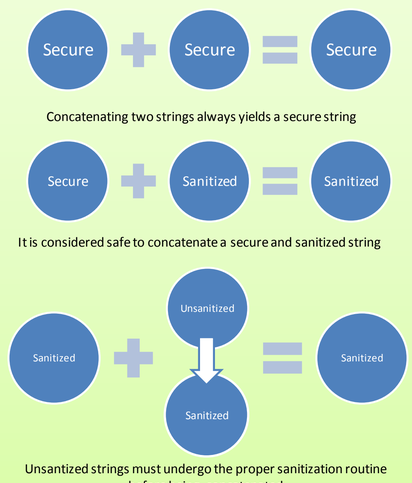
Data is wrapped with security information that protects its integrity throughout the application

Pieces of data are assigned *security types:*

Example: Safe handling of strings is central to the security of an application to prevent against injection and cross-site scripting (XSS) attacks



To preserve security types in the application, type system rules control how data interacts :



GuardRails

Access Control

Automatically produce permissions checks where data is accessed/changed

```
conditions = ["#{Project.table_name}.id
                              IN (#{ids.join(',')})
                              AND #{Project.visible_by}"]
Issue.send( ... :find => { :conditions => conditions})
```

Without the Project.visible_by condition, unauthorized users were able to see the issues of private projects.

With GuardRails, a developer can specify both the conditions for data access and editing directly from the data model. The system then inserts proper permissions checks rather than requiring the developer to do so.

Protected data access can be implemented by overriding the **find** method to check for certain authorization conditions specified in the annotations. Protected data updates can be implemented by statically searching for where protected data can be modified and inserting explicit permissions checks.

Sanitization

Automatically use sanitization routines on potentially harmful input

```
tags = %w(a acronym b strong i em li ul ol h1 h2 h3 h4 h5 h6
                              blockquote br cite sub sup ins p)
user_input = sanitize(user_input, :tags => tags,
                              :attributes => %w(href title))
send(user_input)
```

Without the sanitization call, the application is vulnerable to Cross-Site Scripting attacks.

Because HTML sanitization is so ubiquitous, GuardRails enforces it automatically rather than requiring annotations. This further reduces the opportunity to introduce security flaws.

GuardRails takes advantage of security typing to automatically apply appropriate sanitization routines. Strings that are rendered by a browser must be either secure or unsecure sanitized. Strings that do not satisfy these conditions are sanitized before being sent to the client.