

Finding Weaknesses in Web Applications Through the Means of Fuzzing

Rune Hammersland

Outline

- Introduction
- What is fuzzing?
- What we have done
- Experiment
- Findings
- Conclusions

Introduction

- Bad handling of input in web applications leads to vulnerabilities.
- Testing input handling for web applications is hard.
- Letting the programmer choose test values might not be wise.

Fuzzing?

- Random testing technique discovered by Miller et al. “a dark and stormy night.”
- Study from 1990 done on UNIX command line applications shows about a third crashed or hung.
- Later studies includes:
 - Command line: GNU/Linux, Windows NT, Mac OS X
 - GUI programs: X11, Windows NT/2000, Mac OS X

Fuzzing?

- Has also been used:
 - Month of Kernel Bugs
 - Month of Browser Bugs
 - Programming libraries
 - Network protocols

We Have:

- Evaluated fuzzing for web applications (is the method feasible in this area?)
- Created a tool chain for fuzzing web applications.
- Used the tool chain to find weaknesses in web applications.

Experiment

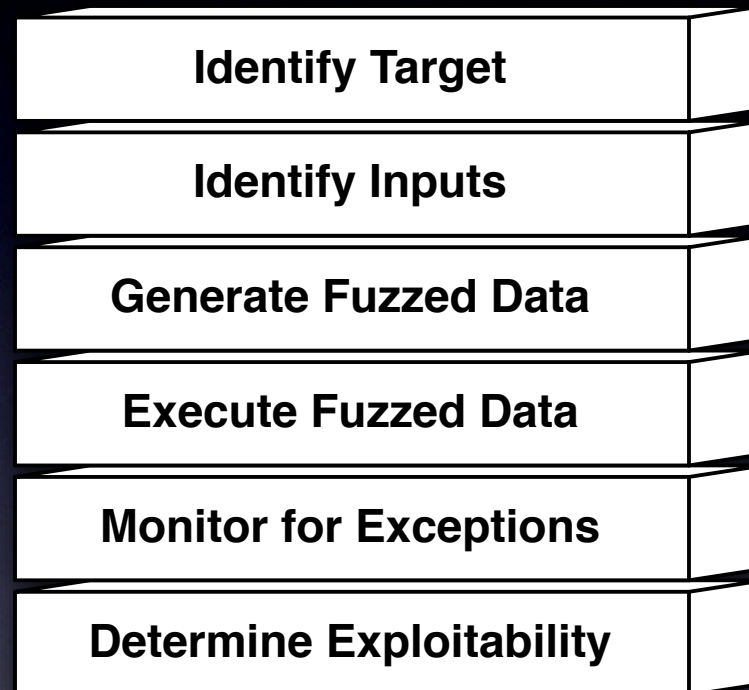


Figure taken from
“Fuzzing: Brute Force Vulnerability Discovery”
by Sutton and Amini

Experiment (targets)

- Chyrp (weblog, PHP)
- eZ Publish (CMS, PHP)
- Junebug (wiki, Camping)
- Mephisto (CMS, Rails)
- ozimodo (weblog, Rails)
- Request Tracker (ticketing system, Perl)
- Sciret (knowledge base, PHP)
- Wordpress (weblog, PHP)

Experiment (identify input)

- We use a crawler to traverse the application's pages.
- Each page sent to a callback function.
- Weed out forms and create attack script.

Experiment (identify input)

My Awesome Site

http://chyrp.net/demos/7/ RSS Google

my awesome site

Hi there!

This is your special sandbox for Chyrp. As long as you don't log out, you can toy around with this install as much as you like. Hack it if you want, it'll only affect you anyway.

Some things to try:

- Enabling some [Modules](#)
- Trying out the various [Feathers](#)
- Editing this post (click "Edit" down there)
- Deleting this post
- [Writing](#) a new post
- [Adding a page](#)

June 4th, 2008 / [Trackback](#) / [Edit](#) / [Delete](#)

Pages

- [Home](#)

Archives

- [June 2008](#)

Welcome, Admin!

- [User Controls](#)
- [Toggle Admin Bar](#)
- [Log Out](#)

Stats

- Queries: 11
- Load Time: 0.009

Search...

rss / theme © aaron macdonald 2007
powered by [chyrp](#)

Experiment (identify input)

```
<form action="/chyrp/" method="get">  
  <input type="hidden" name="action" value="search" />  
  <input type="search" name="query" />  
</form>
```

Experiment (generate fuzz)

```
setup("Chyrp") do
  @host = "10.0.0.2"
  @port = 80

  before do
    # ...
  end

  attack("/chyrp/") do
    many :get, "/chyrp/", {:action => "search", :query => ""}
  end
end
```

Experiment (generate fuzz)

```
attack("Search box") do
  url = "/chyrp/"
  many :get, url, {:action => "search", :query => str(100)}
  many :get, url, {:action => "search", :query => byte(100)}
  many :get, url, {:action => "search", :query => big}
end
```

A random request could look like:

```
GET /chyrp/?action=search&query=2016540511
```

Experiment (fuzzing)

- Run fuzzer: `ruby fuzz.rb targets/chyrrp.rb`
- Useful to tail logs from web server.
- Fuzzer dumps log files in a directory for later analysis.

Experiment (analysis)

- CSV files containing number of status codes returned and request timings.
- YAML file contains serialized requests and responses.
- We started by filtering HTTP response codes.

Findings

- Non-semantic use of HTTP status codes (Wordpress)
- Failure to validate input on server (Mephisto)
- Failure to handle exceptions (Mephisto)
- Resource exhaustion (Request Tracker)

Conclusions

- Web applications are vulnerable to fuzzing.
- Small tests found some results.
 - More comprehensive tests should be done for “real” testing.
- Tweaking attack scripts by hand is boring.

Questions?