# Halberd user's guide

Juan M. Bello Rivas

May 7, 2017

# Contents

# Chapter 1

# Introduction

Halberd discovers HTTP load balancers. It is useful for web application security auditing and for load balancer configuration testing.

## 1.1 Motivation

To cope with heavy traffic loads, web site administrators often install load balancer devices. These machines hide (possibly) many real web servers behind a virtual IP. They receive HTTP requests and redirect them to the real web servers in order to share the traffic between them.

There are a few ways to map the servers behind the VIP and to reach them individually.

Identifying and being able to reach all real servers individually (effectively bypassing the load balancer) is very important for an attacker trying to break into a site. It is often the case that there are configuration differences ranging from the slight:

- server software versions,
- server modules

to the extreme:

- different platforms
- server software.

For an attacker, this information is crucial because he might find vulnerable configurations that otherwise (without mapping the real servers) could have gone unnoticed.

But someone trying to break into a web site doesn't have server software as its only target. He will try to subvert dynamic server pages in several ways. By identifying all the real servers and scanning them individually for vulnerabilities, he might find bugs affecting only one or a few of the web servers. Even if all machines are running the same server software, halberd can enumerate them allowing more thorough vulnerability scans on the application level.

# Chapter 2

# Concepts

Halberd operates in stages:

1. Initially, it sends multiple requests to the target web server and records its responses. This is called the *sampling phase*. The time to spend in this phase and the amount of HTTP requests to be sent can be specified using the `--time` and `--parallelism` command line options. See 4.2.5 and 4.2.6.

2. After the analysis phase finishes, either normally or because the user interrupted it pressing Control-C, the program processes the replies looking for signs of load balancing. This is called the *analysis phase*.

3. Finally, halberd writes a report of its findings to the screen (or to a file if the `--out` option is used).

The user may skip the scanning phase and proceed to the analysis of samplings saved in a previous session. See 4.2.10 and 4.2.11.

The following is a list of detection techniques currently implemented:

## 2.1 Date comparison

HTTP responses reveal the internal clock of the web server that produces them. By tracking how many different clocks appear to be, halberd can have some insight into the number of real servers.

## 2.2 MIME header field names, values and their order

Differences in fields appearing in server responses can allow halberd to narrow down its search.

## 2.3 Generating high amounts of traffic

Under certain configurations, load balancers might start to distribute traffic only after a certain threshold has been reached. By default, halberd attempts to generate a sizable traffic volume to trigger this condition and reach as many real servers as possible. See 4.2.6.

## 2.4 Using different URLs

An HTTP load balancer can be configured to take URLs into account when deciding where to direct requests. Gathering URLs with a spider and feeding them to halberd will make it more likely that all servers will reply at some point. See 4.2.7.

## 2.5 Detecting server-side caches

Halberd might come across web sites having server-side caches (e.g.: Squid). This kind of configuration is appropriately identified by halberd and in case there is more than one cache, the program can enumerate them.

## 2.6 Obtaining public IP addresses

Sometimes cookies or special MIME fields in server responses can reveal public IP addresses or host names. In these cases the load balancer can be bypassed connecting directly to the real servers.

# Chapter 3

# Installation

## 3.1 Prerequisites

You need Python version 2.3 or above with the threading and MD5 (or SHA1) modules enabled. If you want to scan through HTTPS you will also need a Python interpreter configured with support for SSL sockets.

## 3.2 Supported platforms

Halberd should work in every machine satisfying the prerequisites mentioned above. The program has been successfully built and tested on GNU/Linux, Windows 2000 and Mac OS X.

## 3.3 Installation steps

Installing halberd is a very simple task. It suffices to write (perhaps as root):

```
python setup.py install
```

Halberd creates the file `$HOME/.halberd/halberd.cfg` when you execute it for the first time. This file lets the user configure proxy settings and specify an SSL certificate.

It is recommended that you read the output of `python setup.py install --help` in case you want to fine tune the installation process.

# Chapter 4

# Operation

This chapter explains how to use halberd in the real world.

## 4.1 Sample session

Peter is on the phone, talking to a client who asks him to perform a penetration test of a large web application that's just entered production.

He impatiently taps his fingers against the wooden table waiting for the email from the client to arrive. Our hero is finally given the green light and he begins his dutiful work.

First of all, a load balancer scanning is due. He fires up a virtual terminal and writes:

```
$ halberd http://www.target-company.com
```

Peter reads the output of his previous incantation and nods...

```
halberd 0.2.0

INFO looking up host www.target-company.com...
INFO host lookup done.
INFO www.target-company.com resolves to x.x.x.1
INFO www.target-company.com resolves to x.x.x.2
x.x.x.1   [######    ]  clues:   3 | replies:  17 | missed:   0
```

After 17 replies he thinks halberd has enough samples to analyze, so he hits Control-C and stops the scan for this host. Peter notes there is DNS load balancing in place so he'll have to check both IP addresses.

```

*** finished (received SIGINT) ***

==============================================================
http://www.target-company.com (x.x.x.1): 1 real server(s)
==============================================================
server 1: foo/1.2.3  mod_bar/4.2 (Unix)
-------------------------------------------------------------

difference: 3600 seconds
successful requests: 17 hits (100.00%)
header fingerprint: c0ba8262100168851872c8feea3196f21ba2d732
different headers:
  1. Server: foo/1.2.3  mod_bar/4.2 (Unix)
```

”Nothing to see here, let’s move along” muttered Peter while halberd progressed to the second IP address.

```
============================================================
http://www.target-company.com (x.x.x.2): 2 real server(s)
============================================================

server 1: foo/1.2.2  mod_bar/4.2 (Unix)
------------------------------------------------------------

difference: 3600 seconds
successful requests: 11 hits (33.33%)
header fingerprint: 732deadbeef100168851872c8feea196f21ba2d2
different headers:
  1. Date: Wed, 16 Aug 2006 22:47:04 GMT
  2. Server: foo/1.2.2  mod_bar/4.2 (Unix)

server 2: foo/1.2.3  mod_bar/4.2 (Unix)
------------------------------------------------------------

difference: 3662 seconds
successful requests: 23 hits (66.66%)
header fingerprint: ad2d33a88f259b434c094a7b1172f5697a35cff4
different headers:
  1. Date: Wed, 16 Aug 2006 22:46:02 GMT
  2. Server: foo/1.2.3  mod_bar/4.2 (Unix)
```

”Aha!” shouts our fellow, almost falling off his chair. ”Not only did they forget to set up NTP on those servers, letting me distinguish them easily, they also have different *server versions*!”

His eyes go blank while he runs a search in his brain cell based, caffeine fueled vulnerability database for the terms foo/1.2.2 mod_bar/4.2. He remembers there was a recent exploit for a buffer overflow in that version of foo.

Peter has to take into account that the vulnerable web server gets one third of the traffic (33.33%). Thus, he executes the exploit enough times to make sure it hits the exposed target and he finally breaks into the machine.

”I’m so lucky to have this wonderful load balancer detector in my toolkit! This vulnerability could have easily been skipped.”

Our hero begins to walk in circles around the room, planning the next stages of his assault and anticipating with apprehension the time when he’ll have to write the report for his client.

## 4.2 Command line options

### 4.2.1 `--version`

Shows the program's version number.

### 4.2.2 `-h, --help`

Shows a help message describing every option.

### 4.2.3 `-q, --quiet`

Runs quietly, limiting the amount of information being displayed while the program runs.

### 4.2.4 `-d, --debug`

Enables debugging information. This can be useful if you want halberd to dump all the HTTP headers or if you're debugging the tool itself.

### 4.2.5 `-t NUM, --time=NUM`

Stops halberd after NUM seconds have passed since the beginning of the sampling phase.

### 4.2.6 `-p NUM, --parallelism=NUM`

Specifies the number of parallel threads to use for network operations. This can increase the amount of requests per second during the sampling phase.

### 4.2.7 `-u FILE, --urlfile=FILE`

Read URLs from FILE. FILE is a text archive containing an URL in each line. halberd will scan these URLs one by one.

### 4.2.8 `-o FILE, --out=FILE`

Writes the human-readable results from the analysis phase to FILE.

### 4.2.9 `-a ADDR, --address=ADDR`

Specifies the target by its IP address.

### 4.2.10 `-r FILE, --read=FILE`

Loads and analyzes *clues* from FILE. *Clues* are what halberd uses to figure out (during the analysis phase) whether there is a load balancer or not. These *clues* can be written (un-analyzed) to a file with the `--write` option for comparison with future scans or other purposes.

### 4.2.11 `-w DIR, --write=DIR`

Saves *clues* to the specified directory. If it doesn't exist, it will be created.

For being portable, *clues* are stored in text files contained in a special directory layout. The following is an example of the generated file hierarchy:

```
http___www_target_company_com/
http___www_target_company_com/x_x_x_1.clu
http___www_target_company_com/x_x_x_2.clu
```

### 4.2.12 `--config=FILE`

Tells halberd to use the configuration stored in FILE instead of the default `halberd.cfg`.

# Chapter 5

# Support

Suggestions, bug reports and patches can be emailed to the author at `jmbr@superadditive.com`