Apache Security Training

Ivan Ristic <ivanr@webkreator.com>

Talk Overview

1. Apache Security Concepts
2. Installation and configuration
3. Denial of Service attacks
4. Sharing Apache
5. Logging and monitoring
6. Infrastructure
7. Introduction to ModSecurity

Introduction

- What is this talk about?
- Defining the Apache Web platform
- About “Apache Security”
- About the speaker

What is this talk about?

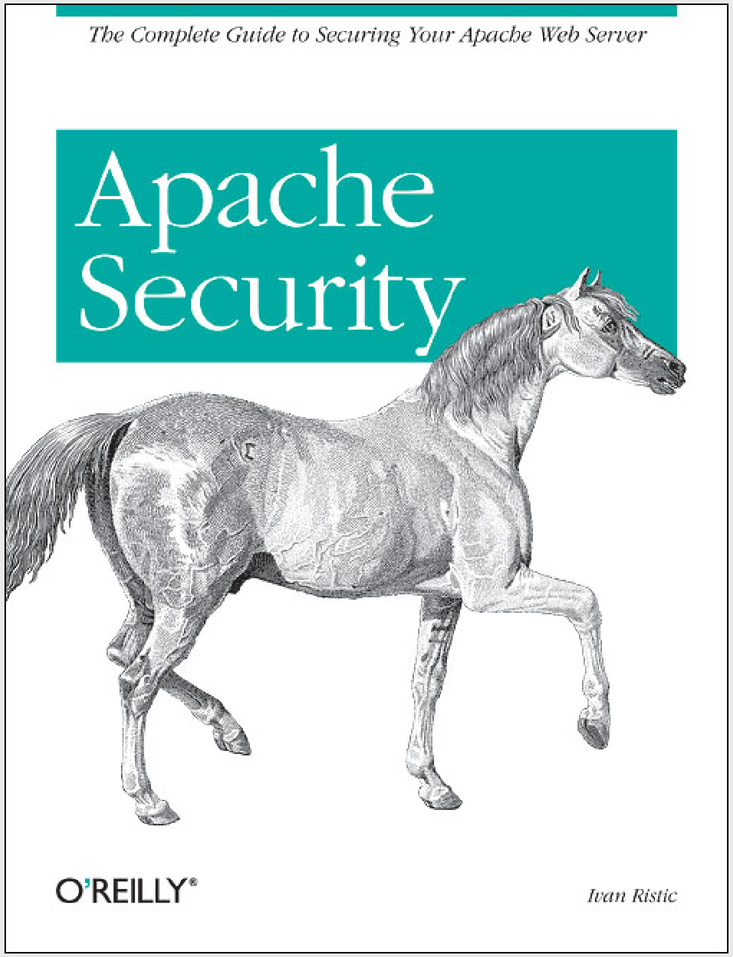
- A high-level overview of everything you need to know if you are deploying Apache.
- Loosely based on my book, Apache Security.
- A mixture of network security, host security, and web application security, in the combination relevant for the Apache web server.

Defining the Apache Web Platform

- Web server
- Application server or application server front-end
- mod_php, mod_perl, Tomcat, etc
- Reverse proxy
    - Performance
    - Load balancing and scalability
    - Architectural flexibility (centralisation, integration, decoupling, access control)
    - Security (web application firewall)
- Probably not the most performant of web servers, but certainly the best choice when all factors (price, performance, flexibility, extensibility, available expertise) are considered

About "Apache Security"

- Everything you need to know to deploy Apache securely
- Discussions on all levels: high-level content followed by technical details
- Published by O'Reilly in March 2005; 420 pages


The Complete Guide to Securing Your Apache Web Server

Apache Security

O'REILLY®

Ivan Ristic

About the Speaker

- Developer / architect / administrator, spent a great deal of time looking at web security issues from different points of view.
- Author of **ModSecurity**, an open source web application firewall (or IDS, if you prefer).
- Author of **Apache Security** (O'Reilly, 2005).
- Founder of **Thinking Stone**, a web security company.

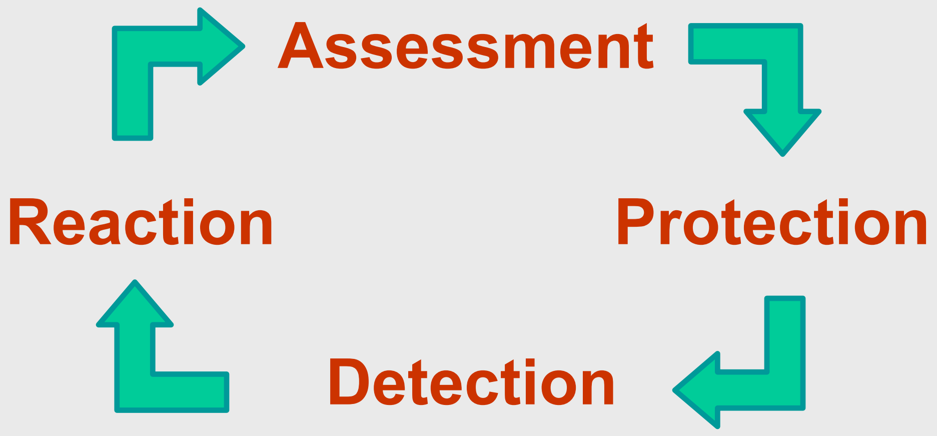
1. Apache Security Concepts

- What is security?
- Three web system views:
  - User view
  - Network view
  - Process view
- What are the threats?
- Choosing the strategy!

What Is Security?

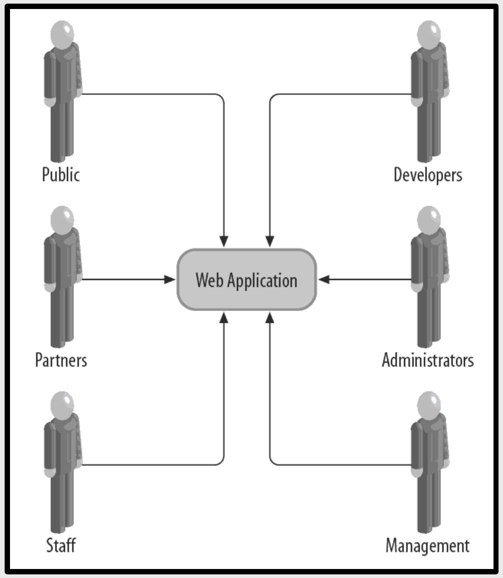
- Static definition
  - Confidentiality
  - Integrity
  - Availability
  - Accountability
- Dynamic definition
  - Assessment
  - Protection
  - Detection
  - Reaction

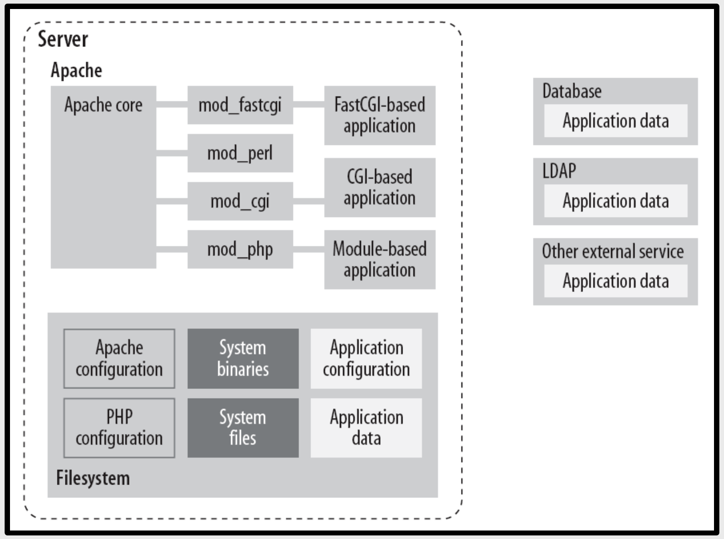
CIA^2

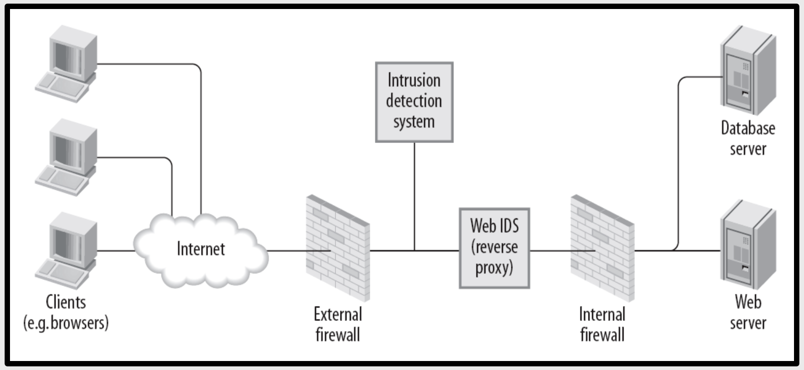


System Views

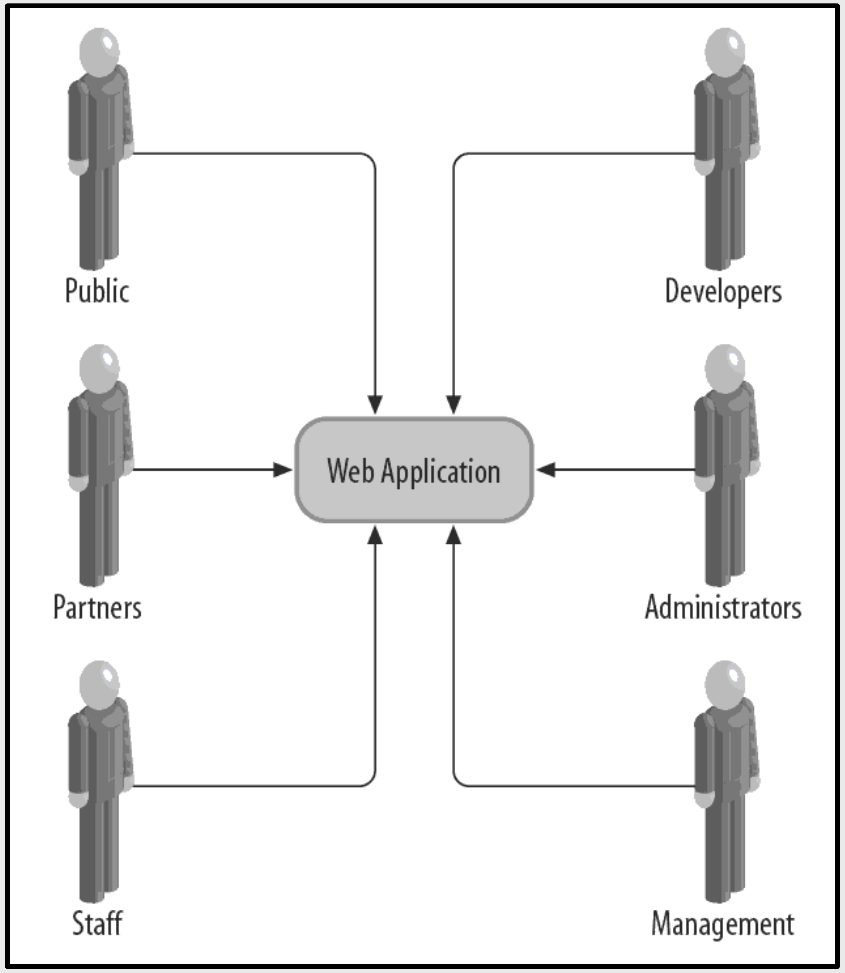
1. User view
2. Network view
3. Process view



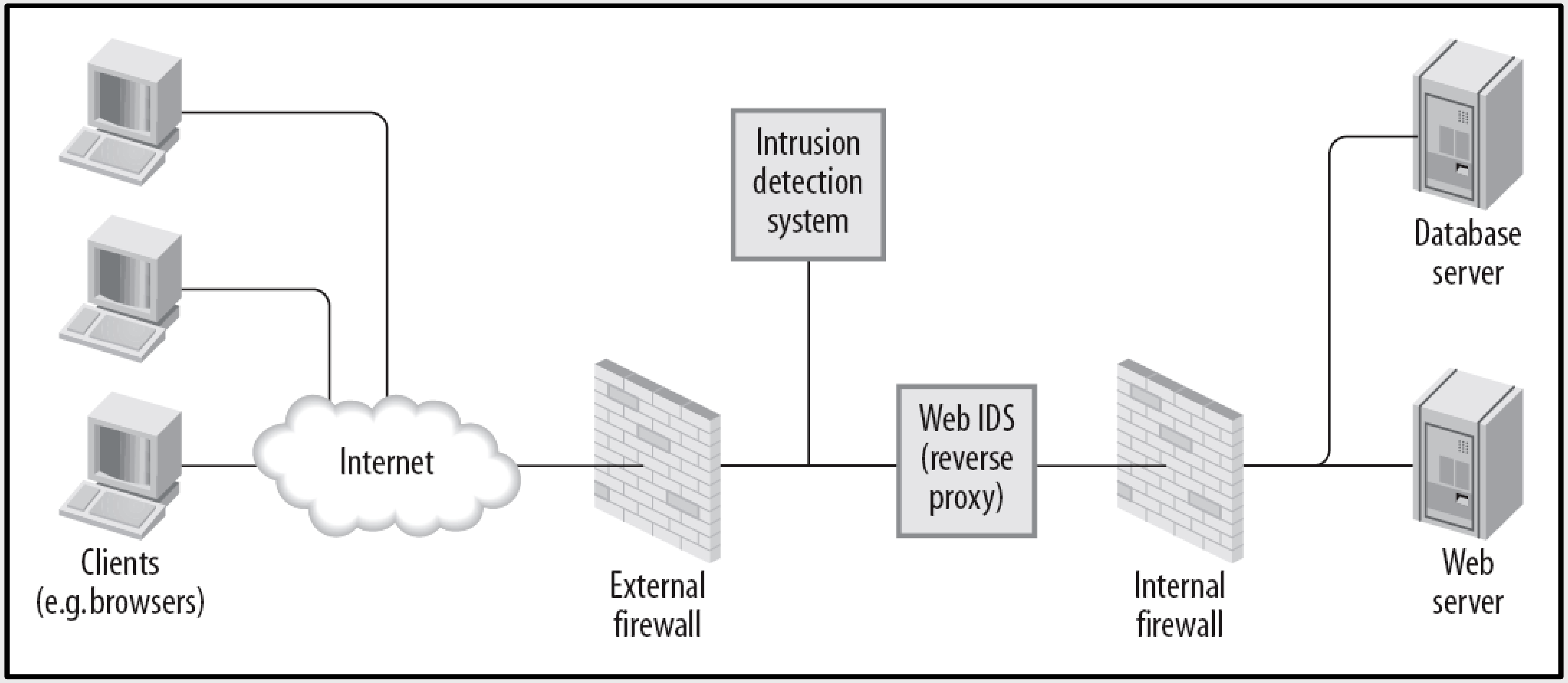




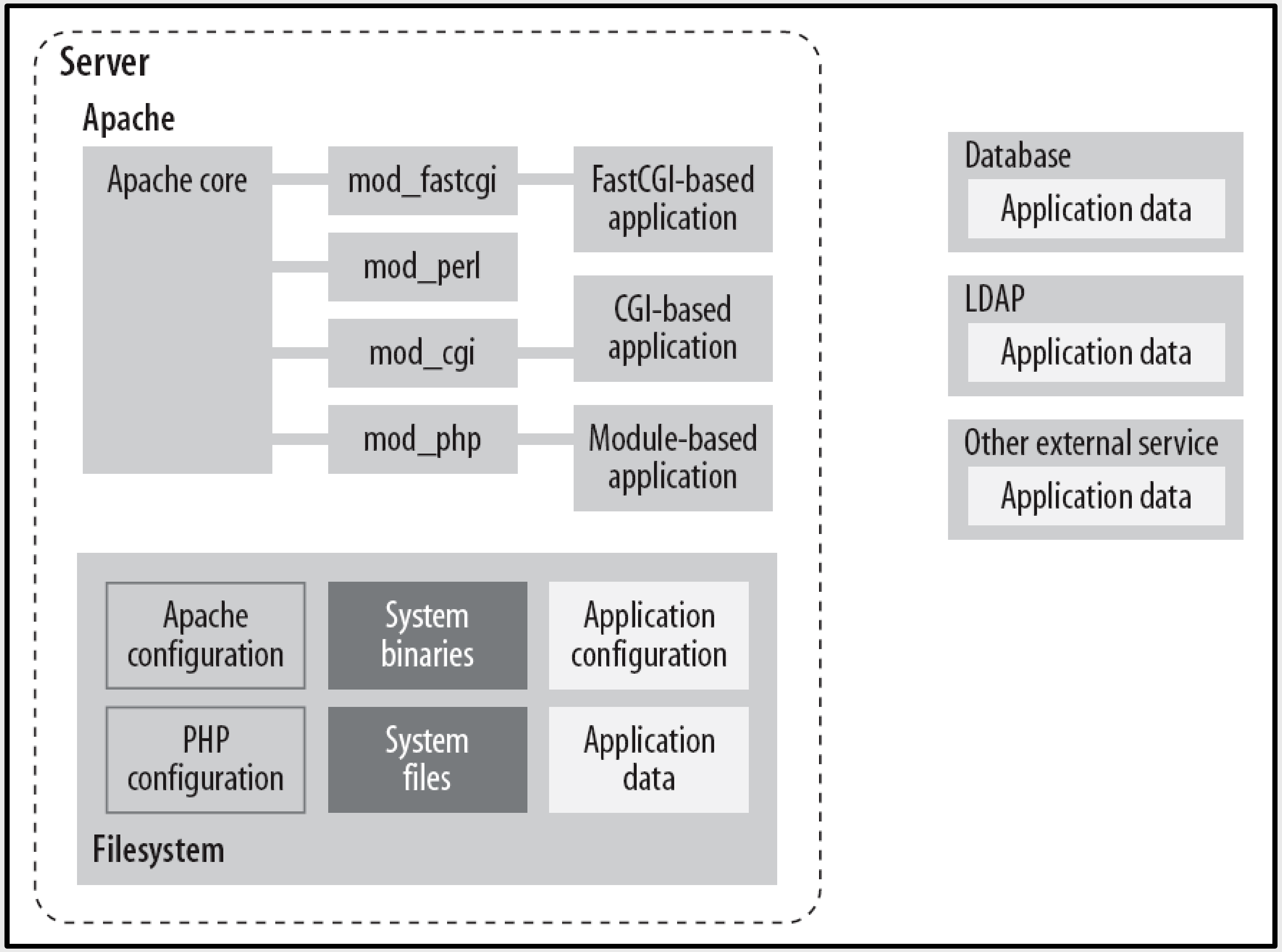
User View (1/3)


Public
Developers
Partners
Web Application
Administrators
Staff
Management

Network View (2/3)



Process View (3/3)



Possible Dangers

You can expect to experience five classes of problem:

- **Apache vulnerabilities**
- **Configuration problems**
- **Denial of Service attacks**
- **Web application security problems**
- **Attacks on users**

Choosing the Strategy

- Helper techniques
- Defensible systems
- Formulating the strategy

Your strategy sets up the stage for what happens later.

Helper Techniques

- Threat modelling
- System hardening matrix
  - Hardening techniques on one axis
  - System categories on the other (e.g. test, development, production, mission critical systems)
- Risk assessment
  - Exploitability, damage potential, asset value
- Patching plan
  - Patch immediately
  - Patch the next working day
  - Patch when the vendor patch comes our, or within five working days (when installed from source)

Defensible Systems

- **Defensible networks**, a term coined by Richard Bejtlich in “The TAO of Network Security Monitoring” (highly recommended).
- Four basic principles:
  - **Minimal**
  - **Compartmentalized**
  - **Maintainable**
  - **Observable**

Formulating the Strategy

- Our strategy formulated:
  - Accept you will fail
  - Be realistic about your resources
  - Compartmentalise
  - Start secure (know your stuff or find someone who does)
  - Remain secure (i.e. patch regularly)
  - Know what is happening
  - Be vigilant
  - React quickly

2. Installation and configuration

- **Use Apache 2**
- Keep up-to-date
- Use the latest version, apply the patches, verify the authenticity of the source code
- Construct configuration from scratch
- Use only the modules you need
- Configure limits
- Configure to fail securely
- **Use SSL**

Changing Web Server Identity

- Moderately useful. It used to help with some automated tools. It may help in the future.
  - Change it directly in the source code.
  - Use the SecServerSignature directive provided by ModSecurity.

Putting Apache in Jail

- Jails are an excellent tool to isolate Apache from the rest of the web server.
- However, they can be difficult to get right (because the mechanism is not natively supported by Apache).
- ModSecurity comes with an easy-to-use chroot mechanism that works in some cases.
- Things to consider:
  - Do not leave any setuid binaries inside.
  - Do not have processes of the Apache user running outside.
  - Do not allow the Apache user to write anywhere.

PHP (1)

- PHP is an excellent tool for building web application. Unfortunately, it was not designed with security in mind:
  - register_globals = off
  - allow_url_fopen = off
  - magic_quotes_gpc = off
  - enable_dl = off
  - expose_php = off
- Configure limits:
  - memory_limit = 8M
  - post_max_size = 8M
  - max_input_time = 60
  - max_execution_time = 30

PHP (2)

- Don't bother with Safe Mode unless you really have to; it's too easy to work around.
- PHP streams can be dangerous, especially the **php://input** construct.
- File access restrictions can be moderately successful:
  - open_basedir = /var/www
- Consider restricting file uploads (if not in use):
  - file_uploads = off
  - upload_max_filesize = 2M

SSL/TLS

- Good at protecting the communication channel, but it is not an end-to-end solution.
- Every web application should use SSL.
- Important web application should use client certificates too.
- Client certificates are the only mechanism that can reliably prevent session hijacking and related attacks.
- SSL is vulnerable to MITM attacks from the local network.
- User interfaces are inadequate.
- Do not mix SSL areas with non-SSL areas.

3. Denial of Service Attacks

- Network-based attacks
- HTTP-based attacks
- Real-life problems

Network-based DoS Attacks

- Very little you can do on the web server level
- Some can be defended from at the network firewall level
- Enable SYN cookies in the operating system
- Be prepared:
  - Know when you are being attacked
  - Have the details of your upstream provider ready

HTTP-based Attacks

- Possible types of attack:
  - Apache vulnerabilities
  - Attacks against the programming model (problem with the limited number of Apache processes)
  - Brute-force attacks.
- Solutions:
  - Patch Apache regularly
  - Configure Apache limits
  - Figure out who is attacking you. Reject such traffic in the firewall. **Often difficult to do.**

Local Attacks

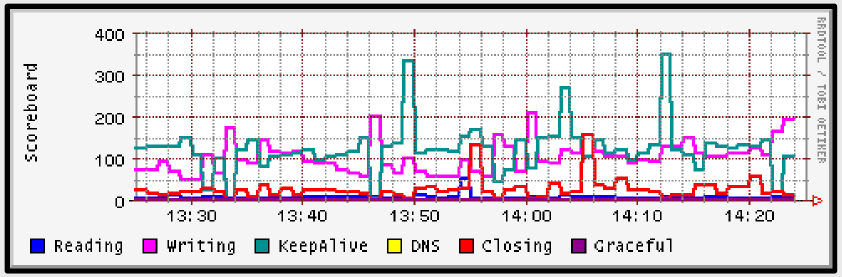
- Local users can do a lot to harm the server:
  - Process creation attack (fight with user limits, process accounting).
  - Memory allocation attacks
  - Disk overflow attacks
  - Kernel vulnerabilities

httpd-guardian

- This is a simple tool I released as part of my Apache httpd tools project (http://www.apachesecurity.net/tools/):
  - It receives information on all requests processed by the web server.
  - Keeps track of the number of hits per IP address.
  - Can talk to the host firewall to blacklist an offending IP address.
  - Will probably become smarter in the future.

Real-life Problems

- What you will encounter:
  - Slow clients and large files (and download accelerators) problems
  - Traffic spikes (e.g. Slashdot, cyber-activism, attacks from competitors)
  - Badly written web applications
- Mitigation:
  - Fix web applications
  - Buy more RAM
  - Tweak the Keep-Alive settings
  - Add response compression (mod_deflate)
  - Add caching (mod_cache)
  - Traffic-shaping modules



4. Sharing Apache

- Sharing with developers
- Sharing with others (virtual hosting)
- Problems:
  - Shared server resources (CPU, RAM)
  - Ability to execute binaries on the server
  - File permissions
  - Shared web server process
  - Shared domain names
- **Who controls the web server?**

Sharing Goals

Our main goal is to make the server and the shared Apache useful *and* safe:

- Protect server (system resources) from users.
- Protect Apache from users.
- Protect users from each other.
- Make privilege escalation difficult, if not impossible.

Shareable Resources

- CPU
- RAM
- Process list (users can create processes)
- Filesystems
- Apache processes
- Domain names
- Databases

Common Usage Scenarios

1. Server and applications managed by the same team. Very rare.
2. Server managed by one team (administrators), applications by another (developers).
3. Server managed by one team. There are many application teams.
4. Server managed by one team. There are many users on the server and it is difficult to hold them accountable (e.g. students, web hosting).

User Access Levels

There are **only two user access levels**:

1. Filesystem access (e.g. via FTP).
2. Process creation (e.g. CGI).

Some consider shell (interactive) access to be a third level. But, in reality, everything that can be done with a shell can be done with the process creation privilege.

Filesystem Access

Protecting the server from its users:

- Use different partitions for system and user files.
- Enforce user quotas.
- Be aware users will be able to access the world-readable files.
- Don't have any world-writeable files.
- Configure Apache not to follow symbolic links.

Protecting users from each other:

1. Give **r** and **x** to the world.
2. Put all users (but not Apache) into a group, then forbid group access to the web server files.
3. Force group ownership to the Apache group, then allow **r** and **x** to this group only.

Still, if Apache can access it, users can access it!

Program Execution (1)

- Program execution is dangerous. Users can exploit vulnerable setuid binaries or kernel vulnerabilities.
- If users are allowed to execute CGI scripts as Apache, or use a module, then they can do anything Apache can:
  - Access all files, possibly write to some.
  - Get full access to sensitive stuff, such as SSL session cache.
  - Write to Apache logs.
  - Get the same access to system resources as Apache.
- It is possible to:
  - Hijack web serving on all domain names
  - Kill Apache processes
  - Access the Apache process memory (e.g. to retrieve SSL certificate information).

Program Execution (2)

- Remedies:
  - Do not use scripting modules.
  - Always use execution wrappers.
  - Use FastCGI.
  - Give each user its own Apache (separate everything: IP, user identity).
  - If there is not enough IP addresses put a reverse proxy in front.
- Not recommended:
  - Perchild MPM
  - Metux MPM
  - Running Apache as root to change identity per-request.

Distributed Apache Configuration

It is quite convenient to allow the users to control parts of the configuration via .htaccess files. But you need to be aware what you are giving away:

- AuthConfig - OK.
- FileInfo - **Not OK**.
- Indexes - OK.
- Limit - OK.
- Options (ExecCGI, FollowSymlinks, Includes, IncludesNOEXEC, Indexes, MultiViews, SymlinksIfOwnerMatch) - **Not OK**.

5. Logging and Monitoring

- Increase logging detail
- Think about log retention
- Include application logs in your plans
- Apache health monitoring
- Event monitoring

Logging Basics

```
# Access log
LogFormat "%h %l %u %t \"%r\" %>s %b
   \"%{Referer}i\" \"%{User-Agent}i\"" combined
CustomLog logs/access_log combined

# Error Log
LogLevel info
ErrorLog logs/error_log
```

Increase Logging Detail

- Add information to the access_log:
  - Referrer
  - User agent
  - Username
  - Session token
  - UNIQUE_ID
  - Transaction duration
- Set error_log level to "**info**"
- Use mod_security:
  - Log POST data
  - Performance measurement

Log Retention

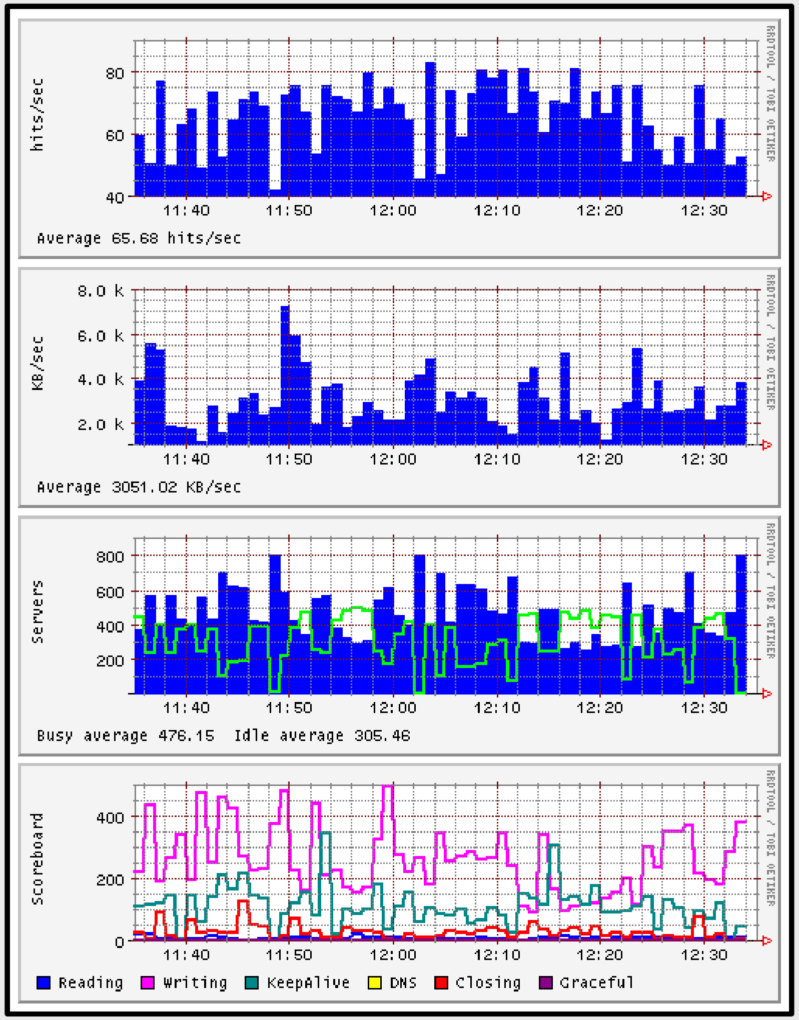
- What do you want to keep and for how long?
- Put logs on a separate partition
- Make sure the filesystem does not overflow (log rotation)
- Keep recent logs on the server for easy access and troubleshooting
- Centralise logs for additional security
  - Syslog
  - Syslog-NG is quite popular
  - Spread toolkit (**mod_log_spread**)

Application Logs

- Treat them equally (rotation, centralisation)
- If you can, get the application to utilise the HTTP codes:
  - Log analysis will be much easier
  - You can configure **mod_security** to selectively log POST data based on the response code

Apache Health Monitoring

- Performance
- Availability
- mod_status
- mod_watch
- apache-monitor



An hour of activity of the Apache running on www.apache.org. Produced with apache-monitor.

Event Monitoring

- Funnel all events into log files
- Do not rely on ad-hoc notification
- Have automated scripts inspect the logs on regular basis
  - Artificial Ignorance
- Real-time monitoring is very cool, but difficult to get right.
  - **Swatch**
  - **SEC** (Simple Event Correlator)

6. Infrastructure

- Network security
- Host security
- Isolation strategies
- Use of reverse proxies

Isolation strategies

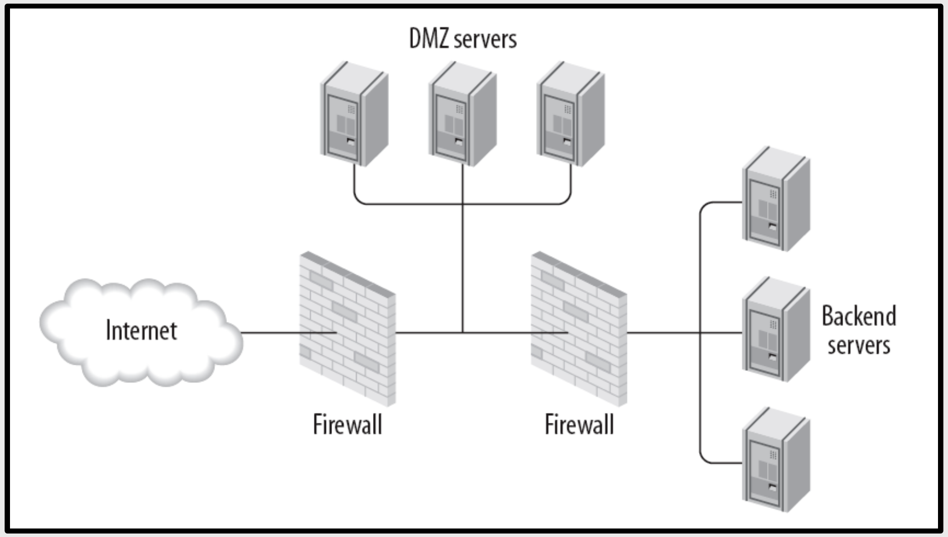
- Techniques:
  - Run as separate user (**suEXEC**, **FastCGI**)
  - Filesystem isolation (**permissions**, **chroot**)
  - Virtual servers
  - Physical servers
- Apache from operating system
- Applications from Apache
- Application modules from each other
- Use separate (restricted) database accounts, or separate database engines

Host Security

- **Timely patching**
- Restricted user access
- Minimal services
- Host-based firewall
- Kernel hardening (grsecurity, SELinux)
- Event monitoring
- Process monitoring
- Integrity validation

Network Security

- Network firewall
- Demilitarised zones
- Centralized logging
- Network monitoring
- Intrusion detection
- Web intrusion detection
- Independent security assessment



Use of Reverse Proxies

- Reverse proxy patterns
  1. Front door
  2. Integration reverse proxy
  3. Protection reverse proxy
  4. Performance reverse proxy
  5. Choke reverse proxy
  6. High availability reverse proxy
- Logical patterns, orthogonal to each other
- Many patterns often used in a single physical reverse proxy

Apache Reverse Proxy

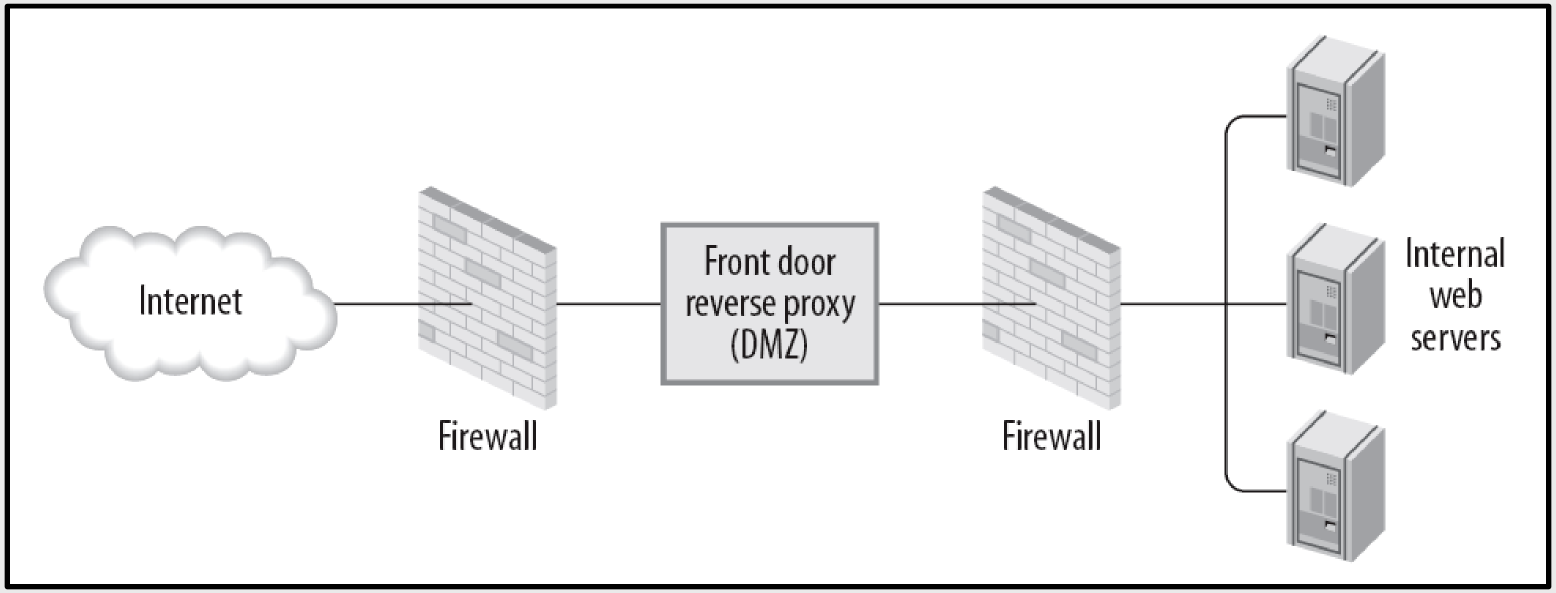
- Construct a reverse proxy using the following modules:
  - mod_proxy, mod_proxy_http
  - mod_headers
  - mod_rewrite
  - mod_proxy_html (third-party module)
  - mod_deflate
  - mod_cache, mod_disk_cache, mod_mem_cache
  - mod_ssl
  - mod_security

Basic configuration is very simple:

```
ProxyRequests Off
ProxyPass / http://internal.example.com
ProxyPassReverse / http://internal.example.com
```

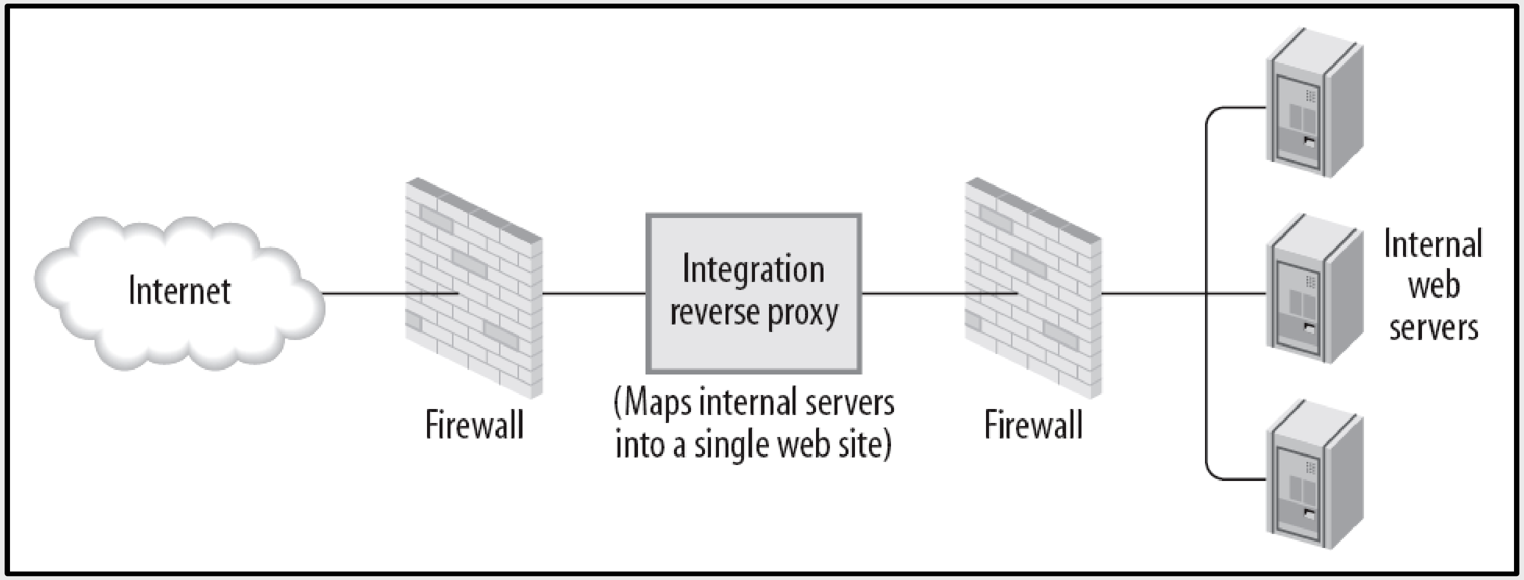
Front Door (1/6)

- Make all HTTP traffic go through the proxy
- Centralisation makes access control, logging, and monitoring easier



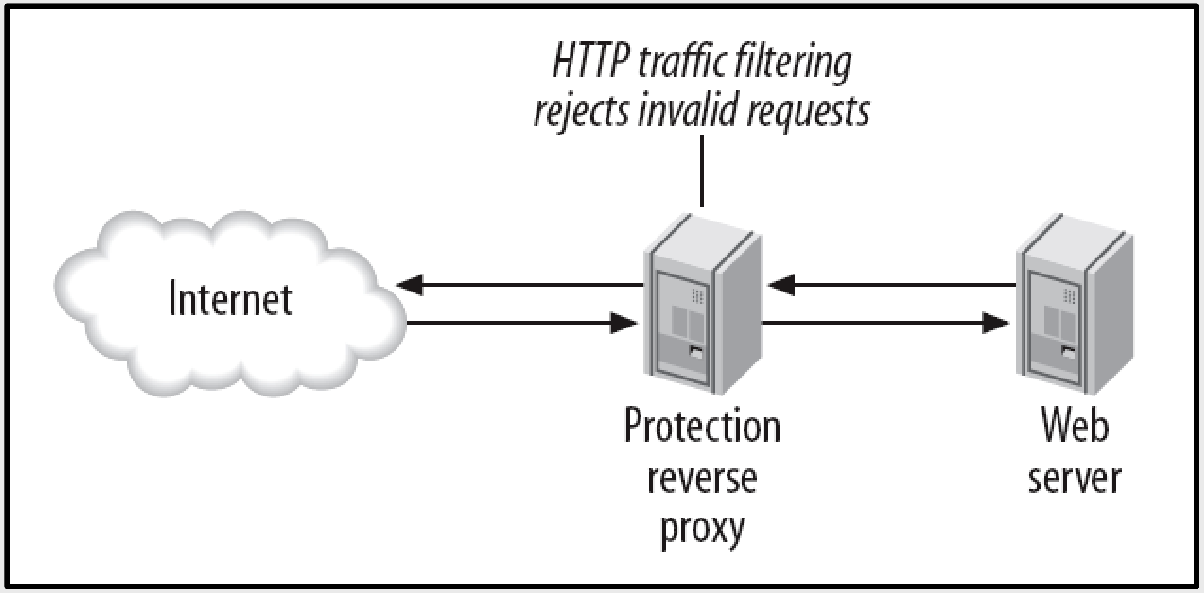
Integration Reverse Proxy (2/6)

- Combine multiple web servers into one
- Hide the internals
- Decouple interface from implementation


Internet
Firewall
Integration reverse proxy
(Maps internal servers into a single web site)
Firewall
Internal web servers

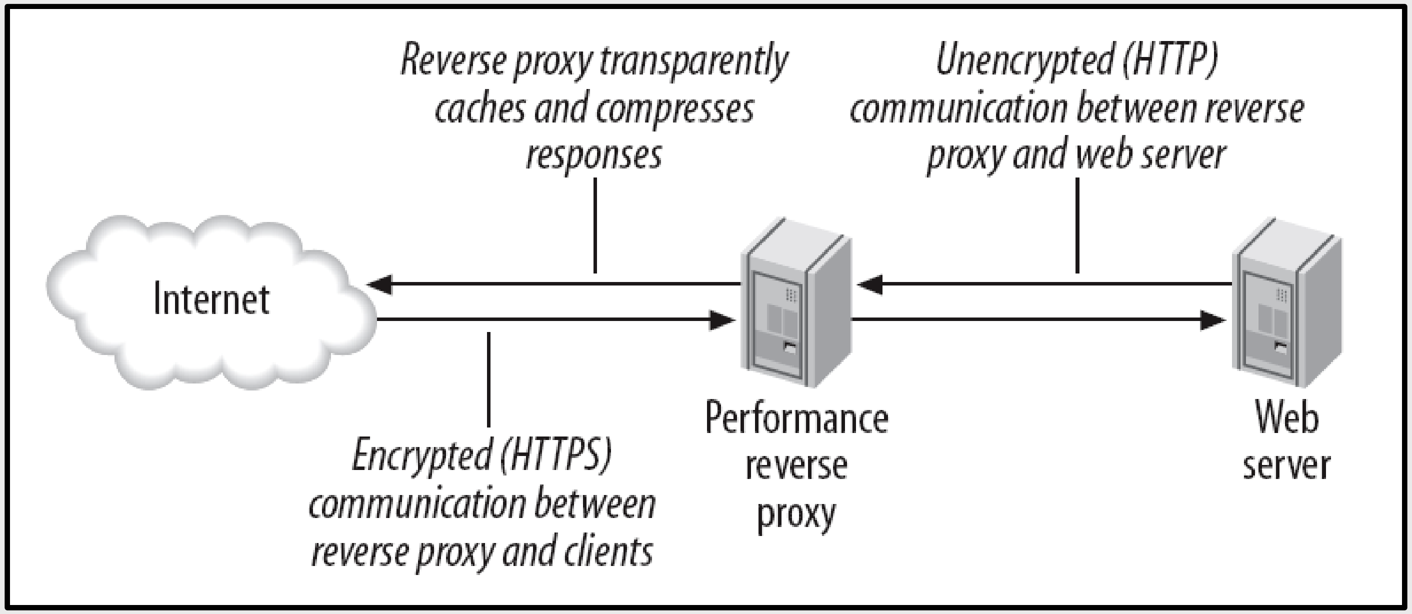
Protection Reverse Proxy (3/6)

- Observes traffic in and out
- Blocks invalid requests and attacks
- Prevents information disclosure



Performance Reverse Proxy (4/6)

- Transparent caching
- Transparent response compression
- SSL termination

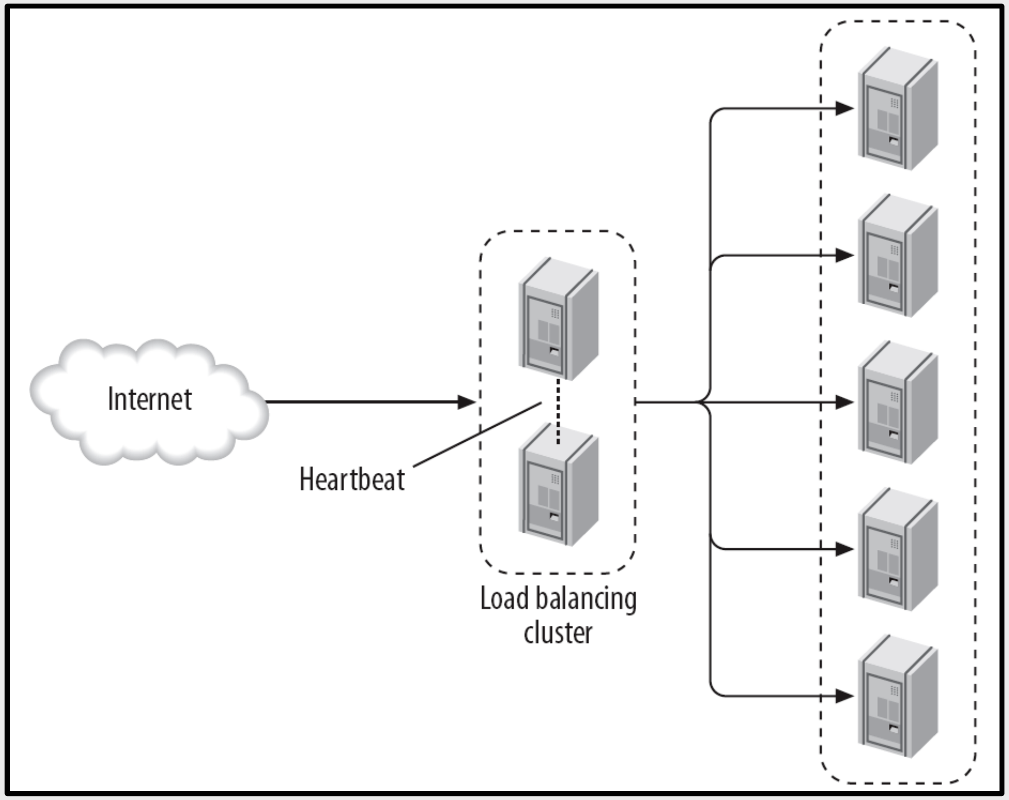

Reverse proxy transparently caches and compresses responses
Unencrypted (HTTP) communication between reverse proxy and web server
Internet
Encrypted (HTTPS) communication between reverse proxy and clients
Performance reverse proxy
Web server

Choke Reverse Proxy (5/6)

- HTTP is increasingly used for Remote Procedure Calling (RPC)
- Make all **internal** HTTP traffic go through the proxy
- Centralisation makes access control, logging, and monitoring easier

High Availability Reverse Proxy (6/6)

- Load balancing
- Fault tolerance
- Scalability



THE END
Questions?

Thank you!

Download this presentation from
http://www.thinkingstone.com/talks/