Linux Security for Developers

Insights for building a (more) secure world

Michael Boelen
michael.boelen@cisofy.com
14 January 2016
We Love Construction
And Magic!

Turning data into:
- Useful output
- Stable software
- Nice services
Why Invest in Security Now?

- Spying
- Internet of Things
- Law
  - 2016  Dutch Data Protection Act
  - 2017-2018  European data protection law
Agenda

● What can go wrong?
● What can we do?
● Strategies and Tools
Michael Boelen

- Open Source Security
  - Rootkit Hunter (malware scan)
  - Lynis (security scan)
- 150+ blog posts at Linux-Audit.com
- Founder of CISOfy
What can go wrong?
Passwords
Case: Phone House

http://sijmen.ruwhof.net/weblog/608-personal-data-of-dutch-telecom-providers-extremely-poorly-protected-how-i-could-access-12-million-records
Creative Users
HOW MANY TIMES A DAY ARE YOU HANDING OVER YOUR INFORMATION?

From the moment we wake — whether on that WiFi-enabled “smart” coffee maker — to the time we make our first Facebook post of the day — until we go to sleep — our digital footprint is being collected and how it’s being shared.

The infographics below outline just a few of the hundreds of ways we voluntarily open our everyday lives to intelligence-gathering companies, government agencies, data brokers and marketers, simply by using our trust in and growing array of technologies.

DOCTOR

Visitor Registration

Surveillance camera

Visitor Registration

RFID access card logs the date, time, and place when you arrive at work.

Malware secretly sends images through webcam

Camera identifies your face to offer you a Facedeal

Check into pub with FourSquare

Webcam recording passengers

Mobile payment

WiFi enabled coffee maker

Smart appliances track energy, water, and gas usage

Personal electric vehicle

Driving to party with EFS

Google Glass sending video to YouTube

Home energy network controller powers on utilities and locks the doors

DATA HUB OF THE UNIVERSE

MORNING

LUNCH

PUBLIC

EVENING

LEISURE

HOME

EVENING

HOME

THOUGHT

THE TAKE-AWAY?

Understand how much data you are sharing simply through everyday use of gadgets and apps. Be aware of how that data may be revealing more intimate details about you. Without consent, it may result in damaging assumptions. What can you do to recover the data that you have been giving away everyday?
What can we do?
“Developers should become auditors of their creative work, and that of others.”

Michael Boelen, 14 January 2016
What can we do?

Improve in steps

● Level 1: Basics
● Level 2: Take ownership
● Level 3: Perform auditing
Level 1: The Basics
Input Validation

Validate!

- Trust nothing
- Double check
  - **Client** = for active user
  - **Server** = for all users
Input Validation

Why Validate?
Prevent data injection *(SQL, RDF, OWL, SPARQL, SeRQL, RDQL, XML, JSON, etc.)*

Where?
*Input forms, data imports*
Data Protection

Encryption:

- **Good** Encryption solves a lot
- **Bad** Knowledge required
- **Ugly** Easy to implement incorrectly
Secure Programming

Using universally unique identifier?

UUID1 = Host (MAC) + sequence + time
UUID4 = Random
Two-factor Authentication

Use
● GitHub

Implement
● Your apps?
Level 2: Take Ownership
Ownership

What?

● The code
● Development systems
● Deployment
● Production
Hardening

● Add new defenses
● Improve existing defenses
● Reduce weaknesses
Hardening

What to harden?

- Operating System
- Software + Configuration
- Access controls
OS Hardening

Operating System:

- Services
- Users
- Permissions
Software Hardening

Software:

● Minimal installation
● Configuration
● Tuning
Access Hardening

Users and Access Controls:

● Who can access what
● Password policies
● Accountability
Data Hardening

Focus on data streams

- Network (data in transit)
- Storage (data at rest)
- Access
Network Hardening

Traffic flows

● Is all incoming traffic needed?
● What about outgoing?
● IPv6?
HTTP Hardening

Header

**X-Frame-Options SAMEORIGIN**
Allow only iframe targets from our own domain

**X-Frame-Options DENY**
Do not allow rendering in iframe
HTTP Hardening

Header

X-XSS-Protection 1; mode=block

Block reflective XSS, avoid returning previous input (e.g. form)
HTTP Hardening

Header

X-Content-Type-Options nosniff

Don't peek into server responses, consider text/html by default
HTTP Hardening

securityheaders.io

Scan your site now

http://linux-audit.com

Scan

Security Report Summary

Site: http://linux-audit.com
Score: A+
Report Time: 12 Jan 2016, 20:10:23 UTC

Headers:
- Content-Security-Policy
- X-Frame-Options
- X-Content-Type-Options
- X-XSS-Protection

Raw Headers

HTTP/1.1 200 OK
Date: Tue, 12 Jan 2016, 20:10:21 GMT
Content-Type: text/html; charset=UTF-8
Vary: Accept-Encoding
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Age: 0
x-Cache: uncached
Transfer-Encoding: chunked
Hardening

Myth: After hardening I’m done

Server Shield v1.1.5

Server Shield is a lightweight method of protecting and hardening your Linux server. It is easy to install, hard to mess up, and makes your server instantly and effortlessly resistant to many basic and advanced attacks.

All IP addresses will be automatically detected and used for the firewall configuration. Automatic security updates are enabled by default.

No maintenance required—just set it and forget it!
Hardening

- Security should be an ongoing process
- Which means it is never finished
- New attacks = more hardening
  - POODLE
  - Hearthbleed
Level 3: Perform Auditing
Myth

Auditing =

● A lot of work!
● Booooooring!
● And.. prone to errors...
Fact

Well, it can be.
Common Strategy

1. Audit
2. Get a lot of findings
3. Start hardening
4. ........
5. Quit
Strategy (New)

1. Focus
2. Audit
3. Focus
4. Harden
5. Repeat!
1. Focus

● Determine what to scan
● Limit scope of systems / applications
2. Audit

● Start small
● Collect data
3. Focus

Determine hardening focus

- Impact
- Number
- Area (e.g. crypto)
4. Harden

- Create implementation plan
- Perform lock down
- Document
  - What, Why, How
  - Exceptions
5. Repeat

● Keep measuring your actions

● Again:
  ○ Ongoing process
  ○ Never finishes
  ○ New attacks
Questions?
Tools

Options:
1. Guides
2. Utilities
Benchmarks / Guides

● Center for Internet Security (CIS)
● NIST / NSA
● OWASP
● Vendors
Benchmarks / Guides

Pros
Free to use
Detailed
You are in control

Cons
Time intensive
Usually no tooling
Limited distributions
Delayed releases
OWASP

Open Web Application Security Project
OWASP

Security Knowledge Framework

OWASP Security Knowledge Framework

The OWASP Security Knowledge Framework is intended to be a tool that is used as a guide for building and verifying secure software. It can also be used to train developers about application security. Education is the first step in the Secure Software Development Lifecycle.

The 4 Core usage of SKF:

- Security Requirements OWASP ASVS for development and for third party vendor applications
- Security knowledge reference (Code examples/ Knowledge Base items)
- Security is part of design with the pre-development functionality in SKF
- Security post-development functionality in SKF for verification with the OWASP ASVS
## OWASP

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Since</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation).</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.0</td>
</tr>
<tr>
<td>2.2</td>
<td>Verify that all password fields do not echo the user's password when it is entered.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.0</td>
</tr>
<tr>
<td>2.4</td>
<td>Verify all authentication controls are enforced on the server side.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.0</td>
</tr>
<tr>
<td>2.6</td>
<td>Verify all authentication controls fail securely to ensure attackers cannot log in.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.0</td>
</tr>
<tr>
<td>2.7</td>
<td>Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases/highly complex passwords being entered.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>3.0</td>
</tr>
<tr>
<td>2.8</td>
<td>Verify all account identity authentication functions (such as update profile, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2.0</td>
</tr>
<tr>
<td>2.9</td>
<td>Verify that the changing password functionality includes the old password, the new password, and a password confirmation.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1.0</td>
</tr>
<tr>
<td>2.12</td>
<td>Verify that all suspicious authentication decisions are logged. This should include requests with relevant metadata needed for security investigations.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2.0</td>
</tr>
<tr>
<td>2.13</td>
<td>Verify that account passwords make use of a sufficient strength encryption routine and that it withstands brute force attack against the encryption routine.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>3.0</td>
</tr>
<tr>
<td>2.16</td>
<td>Verify that credentials are transported using a suitable encrypted link and that all pages/functions that require a user to enter credentials are done so using an encrypted link.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>3.0</td>
</tr>
</tbody>
</table>
OWASP Wordpress Security Implementation Guideline

1. Considerations
2. General security
   2.1 Device security
3. Infrastructure security
   3.1 Apache hardening
   3.2 PHP hardening
   3.3 MySQL hardening
   3.4 Remote access
4. WordPress security
   4.1 Updates
      4.1.1 WordPress Core
   4.2 Removal of unused plugins and themes
   4.3 Plugins & Themes Security
   4.4 Backup
      4.4.1 Automation
   4.5 User roles and proper usage
   4.6 Restrict the access to the admin interface
   4.7 Prevent brute-forcing
   4.8 Implement two factor authentication
   4.9 Remove or change the default administrator account
   4.10 Disable user registration if not needed
   4.11 Change the database prefix
Tools
Tools

Tools make life easier, right?

Not always...
Tools

Problem 1: There aren’t many
Tools

Problem 2: Usually outdated

eglimi/linux_hardening

A report describing how to harden a Linux System. This work has been done as a semester project at university. It is no longer maintained and kept for reference only.

Updated on 27 Dec 2009
Tools

Problem 3: Limited support

AdaLovelance/hardeningserverfromscratch

Este repositorio es un conjunto de scripts para proveer seguridad en un servidor GNU/Linux

Updated 22 days ago
Tools

Problem 4: Hard to use
Introducing Lynis
Lynis

Free
Open source
Shell
Simple
Flexible
Portable
Lynis

Background

● Since 2007
● GPLv3
● Requirements
  ○ Flexible
  ○ Portable
Lynis

Goals

● Perform a quick security scan
● Collect data
● Define next hardening steps
Lynis

Simple

- No installation needed
- Run with just one parameter
- No configuration needed
Lynis

Flexibility

● No dependencies*
● Option to extend easily
● Custom tests

* Besides common tools like awk, grep, ps
How it works

1. Initialise
2. OS detection
3. Detect binaries
4. Run helpers/plugins/tests
5. Show report
Bonus: Integration

- Deployment cycle
- Create your own tests: include/tests_custom
Running

1. lynis
2. lynis audit system
3. lynis audit system --quick
4. lynis audit system --quick --quiet
Auditing Code
Code Validation

Quick wins

● Python: Pylint
● Ruby: ruby-lint
● Shell: shlint
Code Validation

Professional services

● Pentesting
● Code reviews
Auditing Repositories
Sensitive Data

- Secret keys
- Passwords
- Unique IDs
- Customers
Sensitive Data

Search your GitHub repos:
extension:conf password
extension:pem private
filename:.bashrc
filename:.ssh
language:ruby secret
language:python password
Hardening

Harden:

● Your systems
● Your code
● Your sensitive data
Latest Developments
Developments

- Data protection laws
- OWASP
- New Rails security HTTP headers
- Internet of Things
- DevOps→SecDevOps / DevOpsSec
Conclusions
Lesson 1: Continuous Auditing

Many small efforts = Big impact!
Lesson 2: Implement Lynis

```bash
#include lynis.sh
```

---

**[+] Users, Groups and Authentication**

- Searching administrator accounts... [OK]
- Checking UIDs... [OK]
- Checking chgrp tool... [FORBID]
- Consistency check in /etc/group file... [OK]
- Test group files (grep)...
- Checking login shells... [WARNING]
- Checking non-unique group ID's... [OK]
- Checking non-unique group names... [OK]
- Checking LDAP authentication support
- Check /etc/hosts file

[Press [ENTER] to continue, or [CTRL]+C to stop]

**[+] Shells**

- Checking console TTYs...
  - Checking shells from /etc/shells...
  - Result: found x shells (valid shells: 6).

[Press [ENTER] to continue, or [CTRL]+C to stop]

**[+] File systems**

- [FreeBSD] Querying GFS mount points (fsets)...
- Query swap partitions (GFS)
- Testing swap partitions...
- Checking for old files in /tmp...
- Checking /tmp sticky bit...

[OK]
[OK]
[OK]
[OK]
[WARNING]
[OK]
[OK]
Lesson 3: Leverage Security

Security

● Less: Crisis and Leaks
● More: Development Time
Success!

You Finished This Presentation
Want More?

Follow Me

● Twitter: @mboelen
● Personal website: michaelboelen.com
● Blog: linux-audit.com