Proven Strategies for Web Application Security

Justin C. Klein Keane
University of Pennsylvania
About Me

- Security Engineer for the University of Pennsylvania
- Chapter leader of Philadelphia OWASP
- Professional career as a web developer
Proven Strategies

- Sadly, I can't prove a negative
- Discussion followed by concrete examples from Penn
- Ideal solution may not meet all practical needs
Why are Web Apps Targets?

- Globally available
- Always on
- Client application available on most every platform
- Victims of the success of firewalls
Harden the Full Stack

- Every stack has multiple layers
- Utilize the security of each layer to the fullest
OS Hardening

- Limit file system privileges
- Isolate your web or application server
- Install a host based intrusion detection system
- Limit shell access and require secure connections
Isolate your Server

• Application compromise often leads to application server compromise

• Limit the privileges of the server process

• If possible isolate applications from one another
Our Solution

• When possible one codebase runs all applications
• Developers cannot change that code base
• Application server privileges limited (php.ini customization)
Database Security

- Limit application privileges to least necessary
- Isolate data in separate databases
- Make use of stored procedures if possible
- Use safe interaction libraries
Our DB Practice

• When developers start an app:
  – New app specific database provisioned
  – Two accounts
    • one for developers most privileges
    • one for the application with least privilege
  – App password obfuscated in code
  – Stored procedures for sensitive functions
Choose a Standard

- Best advice for a secure environment
- It doesn't matter what you choose
- Base your choice on an evaluation
  - Best choice is the best match to your environment
Standardize on a Platform

- Choose one platform and invest in
  - Security
  - Support
- Develop expertise
- Enable all security features
Our Choice

• School of Arts & Sciences chose Drupal and Symfony
• Our environment is largely PHP, MySQL and Linux based
• Lots of local Drupal developers
• Prior to choice we had no Drupal sites
Framework Security

- Each language has challenges
- No language is “insecure”
- Understand the business reasons for your platform
- Recognize, and use, safe libraries
- Keep your platform up to date
- Do security testing prior to deployment

© Justin C. Klein Keane @madirish2600
Our Solutions

• Customized PHP to limit unsafe functions

• Required Drupal modules to:
  - Audit logins for brute force
  - Audit configuration for security issues
  - Enforce password strength
  - Allow for the use of CAPTCHA
  - Logging to syslog

• Enforce safe database libraries (MySQLi, Doctrine, Drupal db)

• Security testing is a published component of the app deployment
People Standards

- Train your developers
- Establish secure coding baselines
- Publish approved libraries and tools
- Insist that applications meet the baseline
  - Then provide support!
Our Solution

- Published coding standards
- Trained developers perform code level reviews prior to deployment
- All changes are vetted before deployment
Automated Tools

- You should run them!
- You should realize they mostly suck!
- Best testing tool: a person
  - Penultimate – a person with tools and source code
Layer 8 Security

- Personnel charged with security should be developers
- One of these statements is dangerous:

```python
crsr.execute("SELECT * FROM table where id = %s" % id)
crsr.execute("SELECT * FROM table where id = %s", id)
```
Tools We Use

- W3af
- OWASP ZAP
- Firefox Tamper Data Plugin
- Eclipse
- Custom Python scripts
Detect Compromises

- Audit filesystem changes
  - Automated integrity checking
- Scan uploads for malware
- Build logging into your applications
- Perform automated log analysis
- Automated content checking
- Database layer detection
Detection We Use

- OSSEC
  - File integrity and log monitoring
  - Custom decoders and rules
  - Alerts sent via e-mail to security staff
- Clam AV
- SQL scripts look for dangerous tags and keywords
Respond Quickly

• Have backups

• Using version control is an excellent way to be able to recover from a compromise
Tools We Use

- Regular backups (nightly)
- CVS and Git for version control
- Separate development, staging and production environments
- Automated deployments
Conclusion

- Choose a tool and make it work
- Understand security options and enable them
- Secure each layer of the app stack
- Use isolation and monitoring to your advantage
- Enforce use of safe libraries
- Make security testing a requirement for deployment
- Have intrusion detection and incident response plans in place
Thank you

Questions?
About Me

• Security Engineer for the University of Pennsylvania
• Chapter leader of Philadelphia OWASP
• Professional career as a web developer

• Began my career as a web application developer
• Web security is my hobby and specialty
• I understand the challenges of web app security from both sides
• I've responded to a number of application compromises
• I review a lot of application code for security flaws (more on that later)
• Also known as Mad Irish
Proven Strategies

- Sadly, I can't prove a negative
- Discussion followed by concrete examples from Penn
- Ideal solution may not meet all practical needs

- Discuss best practices
- Highlight the things that work for us at Penn
- Since implementing the steps I'm going to outline today we haven't seen any application compromises in covered applications
- Doesn't really prove anything but we have seen attacks succeed at other orgs in Penn
- Our solutions may not be feasible in your situation
Why are Web Apps Targets?

- Globally available
- Always on
- Client application available on most every platform
- Victims of the success of firewalls

Success of web based application delivery is source of security issues
The online application's convenience is its achilles heel
General purpose, always available clients mean anyone can attack app.
Important to realize that clients and servers are targets.
- Client app used for high and low security applications, making it an attractive prize.
Harden the Full Stack

• Every stack has multiple layers
• Utilize the security of each layer to the fullest

• Security in depth
• Be sure to enable all the security protection, detection, and mitigation features available.
• Know the strengths and security features of your platform
OS Hardening

- Limit file system privileges
- Isolate your web or application server
- Install a host based intrusion detection system
- Limit shell access and require secure connections
- Jail your app server if you can
- Be sure to monitor the server process
- Limit privileges of the server
- Customize the server to disable dangerous features until they are needed
- Limit human accounts on the machine
- Segregate your dev/staging/prod environments w. security on each
Isolate your Server

- Application compromise often leads to application server compromise
- Limit the privileges of the server process
- If possible isolate applications from one another

- As much as possible, isolate your apps
- Try to enforce barriers between code and data stores
- Maximize use of code outside of the web server root
Our Solution

• When possible one codebase runs all applications
• Developers cannot change that code base
• Application server privileges limited (php.ini customization)

• Single codebase controls kernel of applications, each app extends this core and can override but not alter it
• Developers focus on customizing apps to their needs rather than building from scratch
• Reduces errors and allows for a secure foundation
• Application code is restricted in terms of PHP functions and even Drupal API calls.
Database Security

- Limit application privileges to least necessary
- Isolate data in separate databases
- Make use of stored procedures if possible
- Use safe interaction libraries

There is a reason DBA is a career field
Your sysadmins or developers shouldn't be doing DBA work.
Be sure to enable all database security measures
Enforce safe interactions with the database wherever possible
Limit application account privileges
Separate application data to prevent change/loss/exposure
Our DB Practice

When developers start an app:
- New app specific database provisioned
- Two accounts
  - one for developers most privileges
  - one for the application with least privilege
- App password obfuscated in code
- Stored procedures for sensitive functions

Each app gets two new accounts that can be audited
- Developer account to set up db
- Application account gets minimum privileges to run the app

Credentials stored in app means you have to limit compromise damage
- Use stored procedures for sensitive tasks.

Each app has its own, distinct database
- Provides reliability, backup, and safety
Choose a Standard

- Best advice for a secure environment
- It doesn't matter what you choose
- Base your choice on an evaluation
  - Best choice is the best match to your environment

- The field of web app sec is huge – narrow your scope to have a chance at success
- Depth beats breadth in security
- Expertise is a service you can offer
- Perform a measured eval w stakeholders
  - Gather requirements
  - Evaluate choices
- If you can't make just one choice select a subset of choices
Standardize on a Platform

- Choose one platform and invest in
  - Security
  - Support
- Develop expertise
- Enable all security features

- Once you make your selection invest in it
- Explore and utilize all the security options available
- Expertise will lend itself to support
- Pursue training options
- Become owners of security related features – tie them into existing SEIM & monitoring
- Get involved in the community
- Collect names of companies that can help
Our Choice

• School of Arts & Sciences chose Drupal and Symfony
• Our environment is largely PHP, MySQL and Linux based
• Lots of local Drupal developers
• Prior to choice we had no Drupal sites

• Drupal is the main choice
  • Includes OpenScholar for self provisioned academic sites
• Symfony for custom web applications
• Work closely with the Drupal security team.
• Send developers, and security people, to both Drupal and Symfony training
• Both technologies are LAMP stack and a good fit for our infrastructure
• Many options for local support and consulting
Framework Security

- Each language has challenges
- No language is “insecure”
- Understand the business reasons for your platform
- Recognize, and use, safe libraries
- Keep your platform up to date
- Do security testing prior to deployment

- Identify the core challenges to the framework and focus on them
- Find the security extensions for the framework and enable/utilize them.
- Understand the advantages the platform provides
- Monitor the platform core and audit it if at all possible.
- Keep the platform up to date and stay involved in the platform community
- Know the security group for your platform
Our Solutions

- Customized PHP to limit unsafe functions
- Required Drupal modules to:
  - Audit logins for brute force
  - Audit configuration for security issues
  - Enforce password strength
  - Allow for the use of CAPTCHA
  - Logging to syslog
- Enforce safe database libraries (MySQLi, Doctrine, Drupal db)
- Security testing is a published component of the app deployment

- Customized PHP and Drupal core limit functions
- Enforced modules in each deployment that provide extra security
- Use of safe libraries
- Code audit is required before any app (in house, commercial, open source) is deployed
- Developed a security reporting process
People Standards

- Train your developers
- Establish secure coding baselines
- Publish approved libraries and tools
- Insist that applications meet the baseline
  - Then provide support!

- Developers have access to coding guidelines
- Developers understand the security review
- Security feedback is designed to educate as well as get fixes
- Publish libraries and best practices
- Provide support to applications using our process
Our Solution

- Published coding standards
- Trained developers perform code level reviews prior to deployment
- All changes are vetted before deployment

- We review all new code based on our published standards.
- We produce training material and have new coops review code and work with developers to fix problems.
- We find vulnerabilities, produce proof of concepts, propose a solution/patch, and then work to implement it.
- Any updates or future changes are also reviewed.
- Code review is aided by tools.
- Code review can't be exhaustive, must focus on key risk areas.
- Often more time consuming to
Automated Tools

- You should run them!
- You should realize they mostly suck!
- Best testing tool: a person
  - Penultimate – a person with tools and source code

- Automated tools won't buy you much
- You should run scanners, but you should really simulate attacker behavior:
  - Run attack tools like SQLMap, Havij, ZAP, w3af
  - Run discovery using search engines
  - Fingerprint bad guy tools
- Automated tools will find low hanging fruit
- Tools are best for assisting a manual code audit
- Auditors should focus on high risk areas of the application
• Personnel charged with security should be developers
• One of these statements is dangerous:
  
crsr.execute("SELECT * FROM table where id = %s" % id)
crsr.execute("SELECT * FROM table where id = %s", id)

• Some security issues are easy to spot
• Others require an expert in the programming language or framework
• Develop expertise so you can spot the hard problems and solve them properly
• Invest in training, knowing more about a framework or language will only help.
• Some of these are attack tools
• Use the tools that work best for you
• An IDE is an often overlooked tool
  • Sorting, searching, and parsing code is invaluable when back tracing a bug
• Tools will vary for white/gray/black box testing
• Tools evolve so keep on top of what is available
• Best tools are the ones you build yourself – Perl or Python are excellent
• LAMPSecurity CTF
Detect Compromises

- Audit filesystem changes
  - Automated integrity checking
- Scan uploads for malware
- Build logging into your applications
- Perform automated log analysis
- Automated content checking
- Database layer detection

- Know which directories should change
- Check your site on the filesystem and via the web
- Scan new material with an AV engine
- Add all logging you can to your application and then watch your logs!
- Scan your data stores (database) for malicious content or entries.
Detection We Use

- **OSSEC**
  - File integrity and log monitoring
  - Custom decoders and rules
  - Alerts sent via e-mail to security staff
- **Clam AV**
- **SQL scripts look for dangerous tags and keywords**

**OSSEC automatically monitors our application logs**
- Custom decoders for application logs that go to syslog
- Custom rules specific to the application
- Custom rules look for attack tool user agents
- Out of the box rules for many attacks
- Clam AV will spot web backdoor scripts
- Cron job runs SQL query that searches for prohibited tags and phrases in text content
• Have backups
• Using version control is an excellent way to be able to recover from a compromise

• Deployments should be scripted.
• Source for deployment should be secure
• Diff the source and the deployed app
  • Use this to monitor changes
• Quickly recover by re-deployment
• Have archives of your filesystem, your configuration, and your database
Tools We Use

• Regular backups (nightly)
• CVS and Git for version control
• Separate development, staging and production environments
• Automated deployments

• Nightly database and filesystem backup
• CVS and/or Git is used for deployment and can pave over a compromised site
• Separate development, staging, and deployment servers
  • Allows us to test offline in staging or dev
• Automated deployments can be complete with a single command.
• Virtual machine snapshots could also prove useful
Conclusion

- Choose a tool and make it work
- Understand security options and enable them
- Secure each layer of the app stack
- Use isolation and monitoring to your advantage
- Enforce use of safe libraries
- Make security testing a requirement for deployment
- Have intrusion detection and incident response plans in place

- Put your eggs in one basket and become an expert on that environment
- Extend support/service to clients using that environment
- Be able to articulate the security wins of the environment
- Invest in a support architecture
- Monitor and maintain your supported environment, at the least in incident responses
- Don't forget to protect as well as detect and react.
- Make security testing a part of deployment
Thank you

Questions?