Understanding Plone Security
Plone Conference 2013
Brasília/Brasil

Fabiano Weimar dos Santos [Xiru]
xiru@xiru.org

Understanding Plone Security

Plone is a CMS known for being quite safe. However, there are few people that know the reasons that make it safer than other solutions.

A Common Argument

Some people say that Plone is less used and tested than other platforms, arguing something similar to the Linus's Law formulated by Eric Raymond in his essay and book "The Cathedral and the Bazaar" (1999).

Linus' Law

"Given enough eyeballs, all bugs are shallow."

or more formally...

"Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix will be obvious to someone."

In fact, software reviewing until reaching consensus about its acceptance is an efficient approach (even on security issues).

However...

However...

In "Facts and Fallacies about Software Engineering", Robert Glass refers to the law as a "mantra" of the open source movement, but calls it a fallacy.

Really?

Numbers

- On plone.org homepage we read: 340 core developers and more than 300 solution providers in 57 countries.
- The best security track record of any major CMS.
  - But for some people, Plone could be "full of unknown bugs".
- Let's try to show that Plone security is not based on obscurity.

Plone Source Code is Developed to be Secure

Plone security comes from Python and Zope communities wisdom.

and Python is AWESOME!

Secure Code Development

- Secure code development is an art, but there is no secret.
- Open Web Application Security Project (OWASP) publish valuable documentation about Secure Development.

OWASP Top 10 - 2013

- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Cross-Site Scripting (XSS)
- A4 Insecure Direct Object References
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Missing Function Level Access Control
- A8 Cross-Site Request Forgery (CSRF)
- A9 Using Components with Known Vulnerabilities
- A10 Unvalidated Redirects and Forwards

OWASP Top 10 x Plone: A1 Injection

- By default, Plone does not use relational databases. So, it is impossible to inject SQL on Plone.
- Even very old solutions that integrate Zope with relational databases (ZSQL) are planned to avoid SQL injection.
- The vast majority of security issues of PHP based CMS are usually SQL injection.

OWASP Top 10 x Plone: A2 Broken Authentication and Session Management

Problems usually happen when the integrator has the responsibility of session verification.

On Plone, integrators use PlonePAS.

PlonePAS

gistfile1.py — Python

```python
    security.declarePrivate('authenticateCredentials')
    def authenticateCredentials(self, credentials):
        """ Authenticate credentials example using cache"""
        login = credentials.get('login')
        password = credentials.get('password')
        if not login or not password:
            return None
        view_name = 'ExampleAuthPlugin_authenticateCredentials'
        criteria = {'login':login, 'password':password}
        cached_info = self.ZCacheable_get(
            view_name = view_name,
            keywords = criteria,
            default = -1)
        if cached_info != -1:
            return cached_info
        if self._DO_LOGIN_VERIFICATION(login, password) is not None:
            ret = (login, login)
            self.ZCacheable_set(ret, view_name=view_name, keywords=criteria)
            return ret
```

How Plone Defines the Authentication and Authorization?

- Plone uses Zope security structure.
- Users will access the application.
  - if you are not authenticated, your user is "Anonymous User" and your role will be "Anonymous".
- Since everything is coded in classes (or modules) Plone protects the access to Python methods.

How Plone protects the Python methods?

- General rule is, every "public" method must:
  - Have a docstring.
  - Not begin with underscore.
  - Have a permission.

If a method does not have a docstring OR
starts with an underscore OR
have a private permission OR

the user trying to access the method DO NOT have an allowed role defined in a permission on the context where the method is being executed

then the access is DENIED.

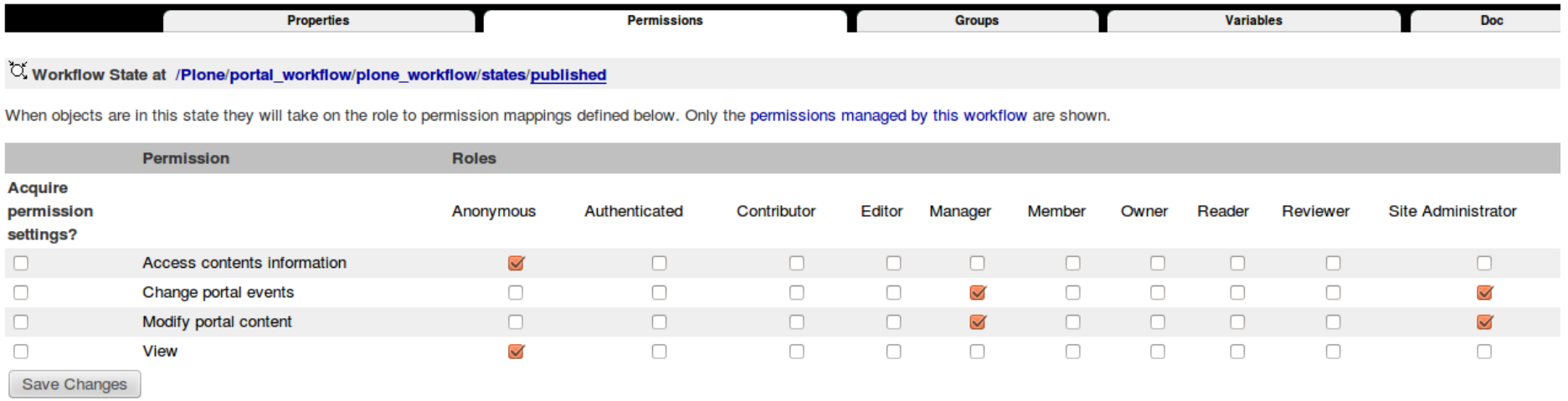
Complex?

- Rights are always granted by roles (not by users).
- Users will receive roles in contexts.
  - It could be done during authentication or manually.
- Plone defines the concept of groups: a set of users.
  - When a group receive roles in a context, all its group members receive these roles.
  - Groups is an abstraction to make privileges management easier.

But who manages the roles allowed on permissions?

- It needs to be done on every object.
- Solution: Workflows!
  - A workflow definition will change the allowed roles for a set of permissions when a transition happens.
  - Example: when a private document is published, the "View" permission is changed to allow the access of role “Anonymous”.

Workflow State Permissions



Properties | Permissions | Groups | Variables | Doc

Workflow State at /Plone/portal_workflow/plone_workflow/states/published

When objects are in this state they will take on the role to permission mappings defined below. Only the permissions managed by this workflow are shown.

Permission	Roles									
Acquire permission settings?	Anonymous	Authenticated	Contributor	Editor	Manager	Member	Owner	Reader	Reviewer	Site Administrator
☐	Access contents information	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐
☐	Change portal events	☐	☐	☐	☐	☑	☐	☐	☐	☐	☑
☐	Modify portal content	☐	☐	☐	☐	☑	☐	☐	☐	☐	☑
☐	View	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐

Save Changes

The Permissions Model

- It is complex to understand by new developers.
  - But it is easy to understand with proper training.
- It is very flexible.
- It is secure!
  - Why? Because all security verifications are automatically made by Zope. The application security management is not responsibility of the end user or the sysadmin. Even the developer has little code to write (and it helps to avoid security issues).

What else Plone does for you?

- XSS protection (A3)
  - All input data is evil
  - Content Filtering
- CSRF protection (A8)
  - CSRF token on all forms (since Plone 3.x)
  - Plone 2.5 does not have CSRF protection!
- All other security risks explained on OWASP Top 10 are covered on the Plone "way of doing things", **except** network traffic encryption.

Encryption

- NSA is watching us!
- Plone is being used on several portals with sensitive information. Be worried about the way how your servers are installed.
- Configure SSL/TLS is trick!
  - Read the SSL/TLS Deployment Best Practices
  - Online Validator: https://www.ssllabs.com/
  - Deploy Forward Secrecy

OWASP Top 10 2013
"The Plone Way" Summary

- A1 Injection
  - SQL injection is impossible when you don't use SQL.
- A2 Broken Authentication and Session Management
  - Plone has a secure and flexible authentication and session management system. Its extension is made using PlonePAS.
- A3 Cross-Site Scripting (XSS)
  - Plone do content transformation using filters to avoid evil input.

OWASP Top 10 2013
"The Plone Way" Summary

- A4 Insecure Direct Object References
  - The security model avoids it by design.
- A5 Security Misconfiguration
  - Plone ships with a secure set of configurations.
  - It is a sysadmin responsibility to keep the system updated, applying all security hotfixes and updating the server operational system.
  - http://plone.org/products/plone-hotfix/

Plone Hotfixes?

```
from Products.kupu.plone.plonelibrarytool import PloneKupuLibraryTool

if hasattr(PloneKupuLibraryTool.getWysiwygmacros.im_func, '__doc__'):
    del PloneKupuLibraryTool.getWysiwygmacros.im_func.__doc__
```

```
>>> class a:
...   def b(self):
...     "foobar"
...     return 1
...
>>> a.b.im_func
<function b at 0x18a46e0>
>>> a.b.im_func.__doc__
'foobar'
>>> del a.b.im_func.__doc__
```

OWASP Top 10 2013 "The Plone Way" Summary

- A6 Sensitive Data Exposure
  - Default Plone sensitive data (like user passwords) are difficult to expose.
  - Cryptography must be used to protect the authenticated access on Plone. It is a sysadmin responsibility.
- A7 Missing Function Level Access Control
  - The security model avoids it by design.

OWASP Top 10 2013 "The Plone Way" Summary

- A8 Cross-Site Request Forgery (CSRF)
  - All forms are protected by a CSRF token
- A9 Using Components with Known Vulnerabilities
  - Plone has security fixes releases.
  - Don't be lazy: install the hotfixes!
  - Read the security announcements of: Linux kernel, apache, nginx, squid, etc. There is really nasty stuff in the wild.

OWASP Top 10 2013 "The Plone Way" Summary

- A10 Unvalidated Redirects and Forwards
  - Plone avoids to use redirects (the example scenario used on OWASP is uncommon).

First Conclusions

- Why Plone is Secure?
  - Mostly because of design decisions.
  - No SQL injection.
  - All evil input is filtered to avoid XSS.
  - All forms use CSRF tokens.
  - Zope security machinery avoids "gambiarras".

First Conclusions

- Sometimes we find security vulnerabilities (nobody is perfect). But when it happens, Plone security team releases the Plone Hotfixes.
- Plone is secure because Python has a high quality code (aka no JVM updates every week or 0-day vulnerabilities luckily fixed someday).

Tips and Tricks

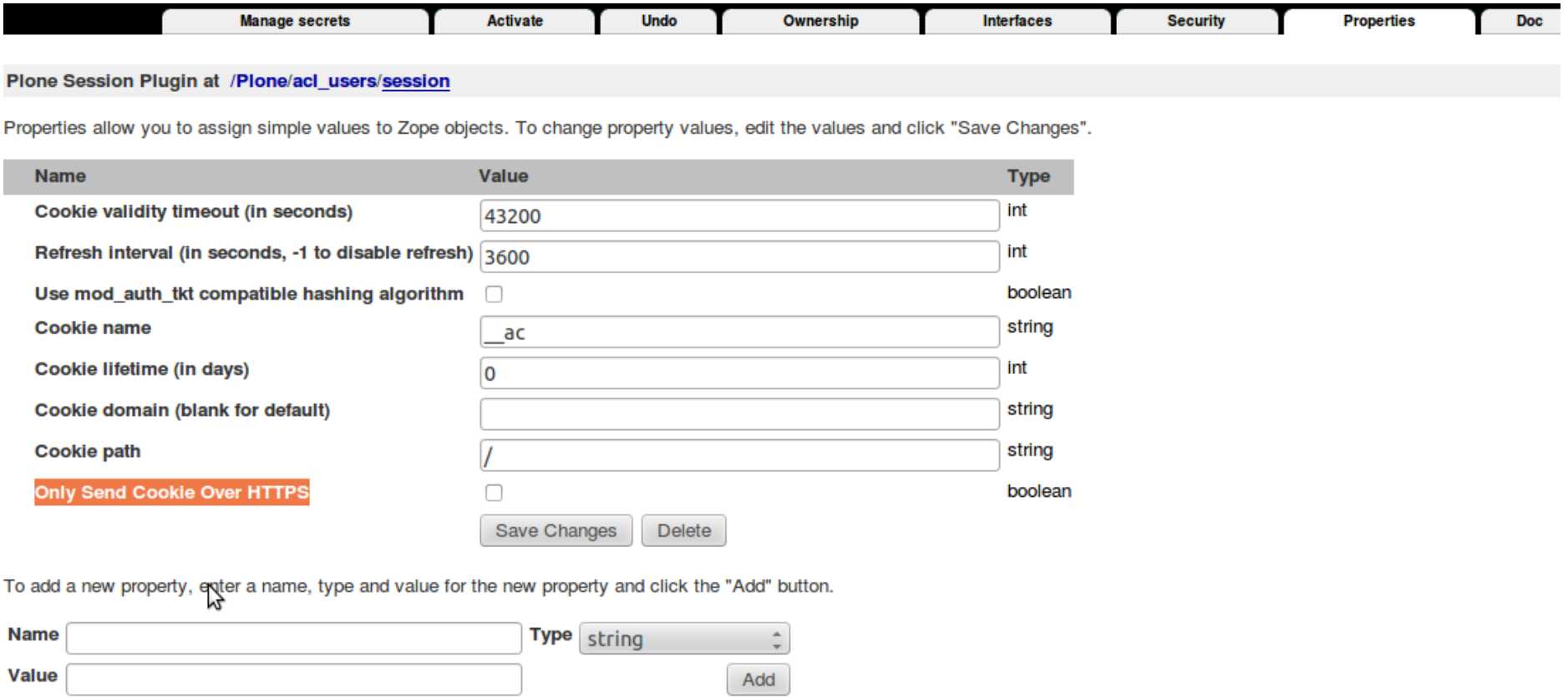
HTTP Cookie Options

- HttpOnly Cookie
  - An HttpOnly session cookie will be used only when transmitting HTTP or HTTPS requests
  - Restrict access from non-HTTP APIs, such as JavaScript
  - Zope support added on 2.12.0b1 https://bugs.launchpad.net/zope2/+bug/367393
  - Enabled by default on Plone (plone.session 3.0b4)

HTTP Cookie Options

- Secure Cookie
  - Only used via HTTPS
  - Makes the cookie less likely to be exposed to cookie theft
  - http://plone.org/documentation/kb/securing-plone

HTTP Cookie Options



What Plone could do better?

Useful HTTP headers

- X-Frame-Options, Frame-Options
  - Example: X-Frame-Options: deny
  - Provides Clickjacking protection.
    - deny - no rendering within a frame
    - sameorigin - no rendering if origin mismatch
    - allow-from: DOMAIN - allow rendering if framed by frame loaded from DOMAIN
  - On nginx, use:
    - add_header X-Frame-Options SAMEORIGIN;

Clickjacking

- Term to describe the attack that is "hijacking" clicks.
- Uses multiple transparent layers to trick the user to click a button or link on another page.
- Can be used to steal likes on Facebook, followers on Twitter or much more dangerous things.

Useful HTTP headers

- X-XSS-Protection
  - Example: X-XSS-Protection: 1; mode=block
  - Enables the Cross-site scripting (XSS) filter built into most recent web browsers.
  - It's usually enabled by default.
    - Re-enable the filter if it was disabled by the user.

Useful HTTP headers

- Strict-Transport-Security
  - Example: Strict-Transport-Security: max-age=16070400; includeSubDomains
  - Enforces secure (HTTP over SSL/TLS) connections to the server.
  - Reduces impact of bugs leaking session data and defends against Man-in-the-middle attacks.
  - Disables the ability for user's to ignore SSL negotiation warnings.

Useful HTTP headers

- X-Content-Type-Options
  - Example: X-Content-Type-Options: nosniff
  - Prevents MIME-sniffing a response away from the declared content-type.
  - Reduces exposure to drive-by download attacks and sites serving user uploaded content.

Useful HTTP headers

- X-Content-Security-Policy, X-WebKit-CSP
  - Example: X-WebKit-CSP: default-src 'self'
  - CSP prevents a wide range of attacks
  - Defined on Content Security Policy 1.0
    - W3C Candidate Recommendation 15 November 2012
    - Declarative policy that lets the authors (or server administrators) of a web application inform the client from where the application expects to load resources.
    - Enable blocking of inline JavaScript

Useful HTTP headers

- Who uses these headers?
  - Google+
  - Facebook
  - Twitter
- Plone could set the headers using a proxy
  - Apache mod_headers
  - nginx proxy_set_header
  - or we can add support in a sprint ;)

Thank You

Fabiano Weimar dos Santos [Xiru]
xiru@xiru.org
Twitter: @xiru

Got interested? Talk with me :)

yes, I'm looking for a new job