DrupalCon

NASHVILLE 2018

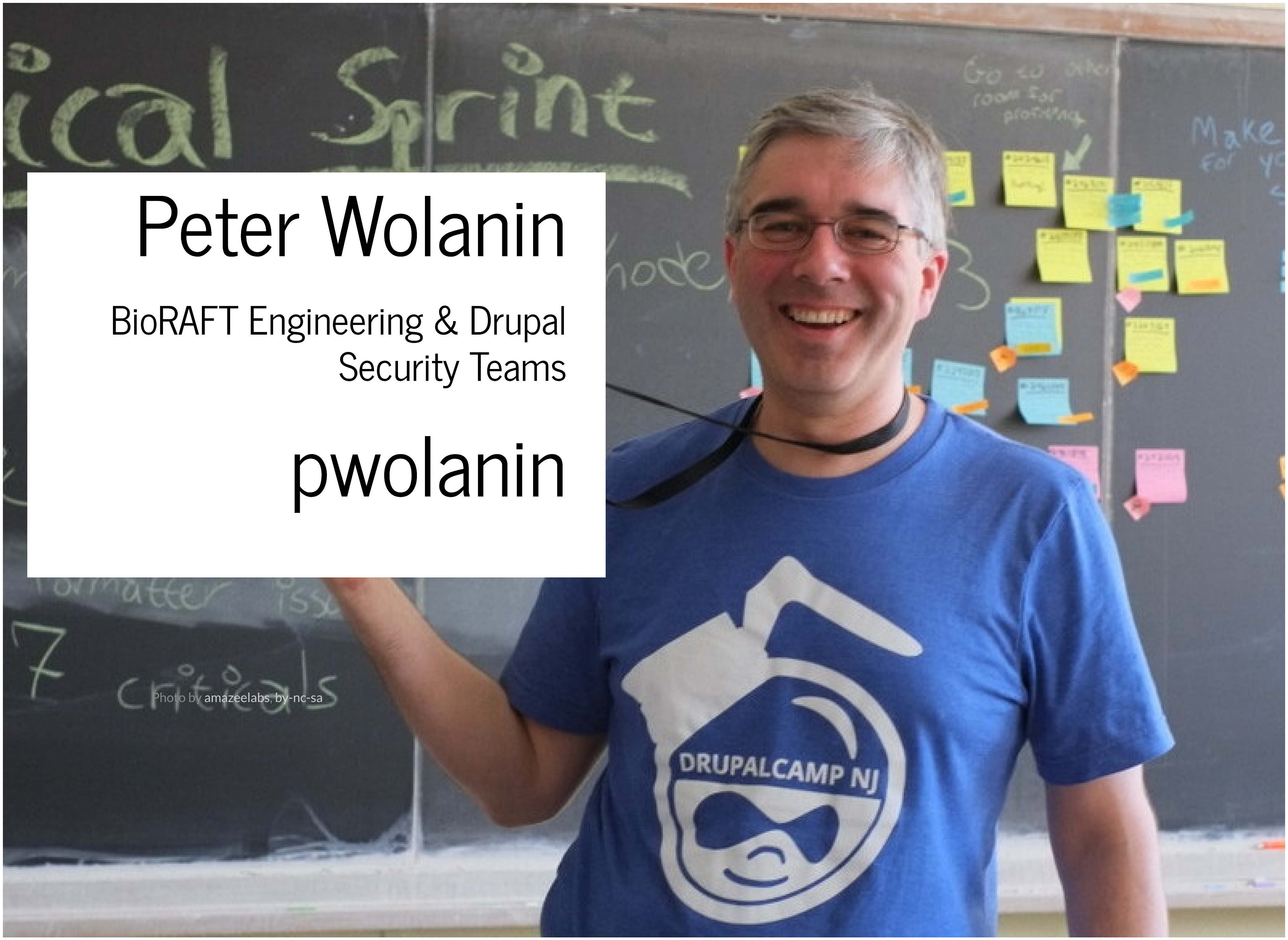
APRIL 9-13

DrupalCon
NASHVILLE 2018
APRIL 9-13

Drupal Core Auto-Update Architecture

Peter Wolanin

David Strauss



Peter Wolanin

BioRAFT Engineering & Drupal Security Teams

pwolanin

Photo by amazeelabs, by-nc-sa

David Strauss

Pantheon CTO / Co-Founder & Drupal Security Team

davidstrauss

Photo by Dominik Kiss, by-nc

Background

- The Update Framework (TUF)
- Paragon's Guide to Automatic Security Updates

Critical Unanswered Questions

- What problem are we actually trying to solve?
- Which users or percent of sites would be helped?
- What is the ROI for the Drupal project / community?

Is there a Use Case?

- Applicable only to sites with no QA process?
- Core only or contrib too?
- Taking all updates or only security?
- Handle multiple web heads or only single?
- Stricter BC requirements for contrib (and core)?

Personas

- **Deploy and Ignore:** Once the site has the functionality needed, there's little maintenance or updating. Doesn't subscribe to PSAs.
- **Diligent but with Simple Needs:** Typically applies updates within a week of release, possibly longer for non-security updates. Follows up on PSAs by directly updating the live site.
- **The Sophisticate:** Needs to apply at least one build step (for CSS, Composer, etc.). Runs QA in a pre-production environment. May deploy to a multi-head cluster.

Premises

- Few sites run just Drupal core.
- Contributed modules have changing Composer dependencies.
- Composer dependencies can, themselves, require security updates.
- Fundamentally changing the deployment requirements would be bad (at a minimum, push the work to Drupal 9).
- The most straightforward installation of Drupal should support automatic updating.

Pick 2 or 3

- Security (integrity and availability)
- Ease of use (reliability)
- Compatibility (dependencies on libraries or PHP version)
- Proudly Invented Elsewhere (PIE)

Insecure approaches

- WordPress-style
  - Webserver overwrites PHP files
  - Single endpoint for updates: api.wordpress.org
  - Does not perform any digital signature verification
  - Any compromise of the endpoint could compromise all sites auto-updating
  - https://www.wordfence.com/blog/2016/11/hacking-27-web-via-wordpress-auto-update/

Option: Airship's Auto-Update Framework

- Pros
  - Someone else built it.
- Cons
  - In-place update prevents multi headed configs
  - Requires PHP 7.2+
  - Their update system (while maybe secure) has a lot of marketing cruft and seems very built around their product and is not componentized or broken apart into a separate project.
  - Seems to go for perfect and not realistic.

Option: Other user account updates core

This option has been available for a long time for people with scripting skips but seems little used.

- Needs a CLI tool that can update core or modules
- Requires some kind of process to build the new codebase an move it into place
- Hard to manage multiple web heads (or even one) without downtime
- Requires composer at least to download dependencies
- Otherwise, suitable for sites with only core and custom modules?

Code as cache?

Are we thinking about the problem wrong? We have a protected mechanism for writing PHP cache files for Twig - if that's secure enough can we use it for everything?

- Have an entry point for web requests
- Have an auto-update subsystem
- Most site code is a validated/signed/protected cache (e.g. PHAR or mtime-protected)
- Take inspiration from ostree, ChromeOS, and Cisco firmware updates to support atomic, distributed switching and rollback.
- An immutable "cache" for more sophisticated or secure deployments. Allow pinning the vendor directory.

Option: Monolithic PHAR Distribution

- One PHAR for the entire site deployment
- Built on Drupal.org servers (or affiliated infrastructure)
- Minimizes complexity of the web server infrastructure
- Pros
  - Simplicity
  - Could just be packages of full Composer runs
- Cons
  - Heavy d.o infrastructure requirements
  - Might/would have to package all modules/themes/etc
  - Might not allow for any patches to code (can break the build)
  - Hard to download in one request

Option: Modular PHAR Distribution

- One or more PHAR files for core
- One PHAR for each module
- Decomposed dependencies (one PHAR per unique dependency)?
- Pros
  - CDN catchability and mirror support
  - Lower bandwidth requirements than a massive file
- Cons
  - No direct support for decomposing Composer dependencies into PHAR files
  - Harder than monolithic phar to support patches (makes them uncachable)
  - Need to create our own way to manage phar files
  - Lose developer transparency
  - Lose compatibility with composer

Recommendations

- Because contrib modules may have changing Composer dependencies, full support for composer is key to any comprehensive solution
- The Composer dependencies themselves may need security updates
- Updates need to happen without human interaction
- Updates need to be highly reliable and not resource-intensive
- Overwriting files in-place is undesirable: needs an atomic method of switching to an updated code base
- Support support multi-head setups (existing methods have no path to doing so)

Possible Implications...

- Requires a writeable code directory - preferably outside the docroot
- Have a way to expose corresponding assets (css/js/images)
- Drupal core and all code needs to be together in vendor/
- Have a "bootloader" to pick the correct codebase (like ostree, ChromeOS, and Cisco firmware)
- Needs to have a way to get a secure manifest
- Core needs to have one or more public keys to base trust on

Possible Implications con't

- An incremental download and assembly of new codebase on cron
- Amortized updates would improve compatibility
- Add to core some support for downloading based on composer.lock
- Drupal install via autoupdate to insure future updates work
- Disallow git repos and patches in composer.json
- Add to core a way for web heads to register
- Support a build step to generate an immutable build artifact to deploy

One Way to Do It

- **index.php:** Bootloader. Initializes autoload path.
- **install.php:** NetInstall style. Performs one "update" as part of installation. Never used afterward.
    - files/
        - vendor-kjfksld/
            - *Composer-style layout of core and contrib.*
        - vendor-sdfiuj84/
            - *Composer-style layout of core and contrib.*
- *Vendor path would also be configurable.*

Infrastructure for Dependency Resolution

Some of these services are already under consideration and would help facilitate automatic updates by core or via scripting

- A secure key infrastructure and release signing or composer.lock signing
- A Composer UI?
- A service to generate a composer.lock for site or a given composer.json
- Re-generating lock files when then are updates (or only security updates?)
- We only need dynamic dependency resolution for contrib updates. Core could be prebaked.

Discuss!

- Is there a clear problem and compelling use case?
- Do you agree with our recomendations?
- Do we have the resources to develop and maintain new infra?
- Do we have resources and expertise to substantially rework the code layout?
- Drupal 8, 9, or 10?

What did you think?

Locate this session at the DrupalCon Nashville website:

https://events.drupal.org/node/21010

Take the Survey!

https://www.surveymonkey.com/r/DrupalConNashville