acquia

# Security Testing with Static and Dynamic Analysis

Barry Jaspan

Drupalcon Szeged 2008

# State of web security

- During this talk, 7 Drupal sites will launch
- All of them are insecure
- 100% of the other dynamic sites you use are insecure, too
  - Including your bank
- Programmers will *never* stop making security mistakes
- We're doomed

# Questions?

Barry Jaspan
Acquia, Inc.
barry.jaspan@acquia.com

# Identifying problems

- Reacting to attacks
  - Ineffective: the attacker may not reveal himself
  - Stressful: attacks tend to cause panic
  - Unreliable: you may not find the hole
  - Extremely undesirable: this is NOT how you want to discover a problem
  - Unavoidable: realistically, this will happen

# Identifying problems

acquia

- Human security audits
  - Effective: security specialists are good at their job
  - Expensive: they know it
  - Unreliable: no one ever finds all the problems
  - Impractical: there is too much code, changing too quickly
  - Irreplaceable: human analysis will ALWAYS be required

# Identifying problems

acquia

- Automated security testing
  - Effective: proven to catch many problems
  - Inexpensive: software is cheaper than people
  - Unreliable: no tool ever finds all the problems
- All approaches are unreliable!
  - Best bet is to combine as many as possible
- Types of automatic testing
  - White box: uses insider access to the source
  - Black box: remote penetration testing
  - This talk is about white-box testing

# Best ways to hurt yourself

- Winner! Insufficiently validated input
  - XSS
  - SQLi
  - CSRF
- Runners up
  - Logic errors
  - Security model flaws
  - Countless others
- Automated testing is quite effective for unvalidated input

# Data tainting

- "Taint" data based on source (*not content*)
  - $_GET, $_POST, $_SERVER, command line...
- Propagate taint

```
$foo = $_GET['foo'];      // $foo is tainted
$bar = 'bar';             // $bar is not tainted
$mix = $foo . $bar;       // $mix is tainted

myfunc($mix);
function myfunc($arg) { /* $arg is tainted */ }
```

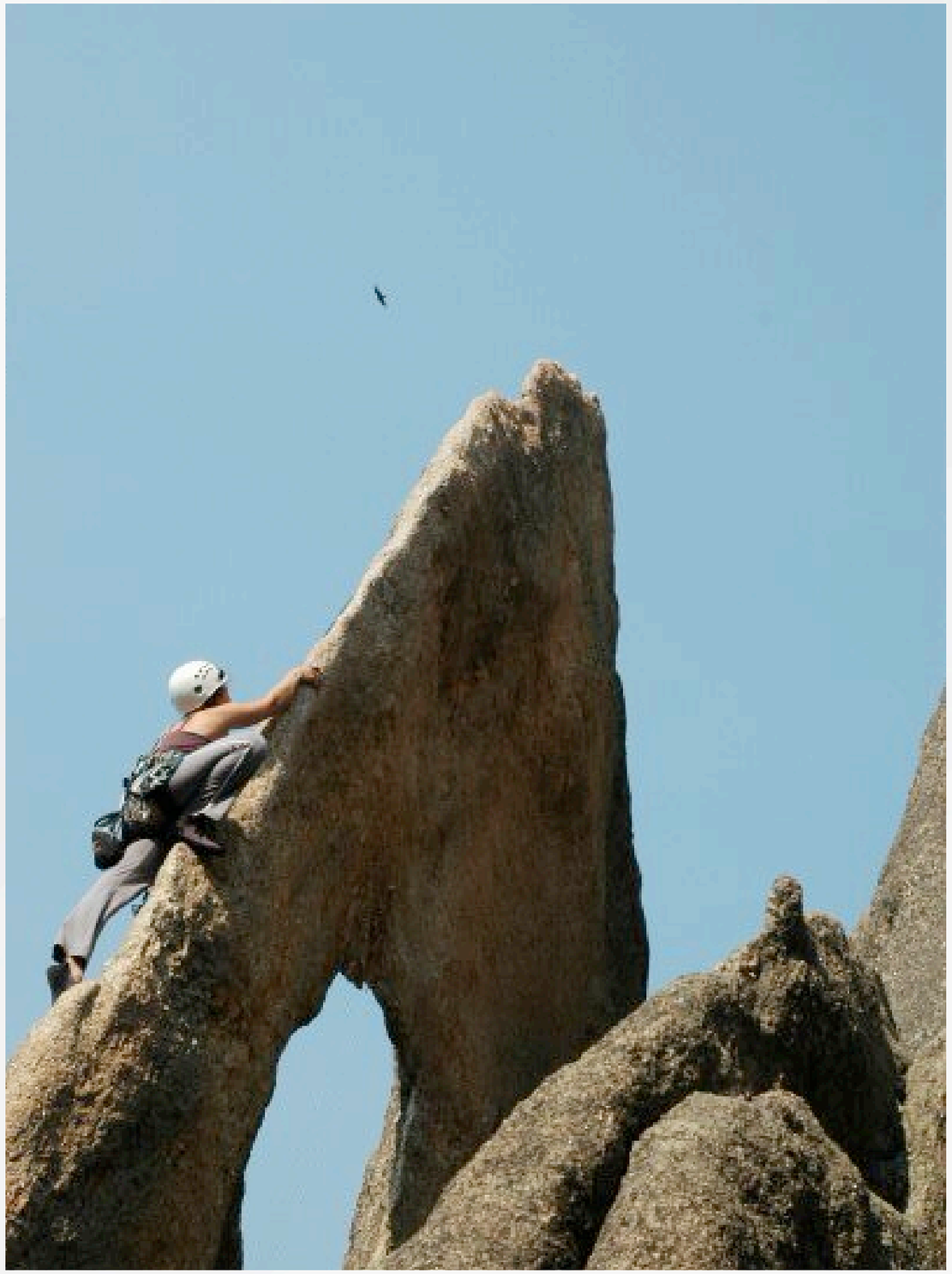
- Define de-tainters and sinks which reject taint

```
echo $bar;                // okay
echo $mix;                // error; echo rejects taint
echo check_plain($mix);   // okay; check_plain de-taints
```

- Rules define taint types, sources, and sinks

# Static Analysis

- Simple example: Coder module
- Study source without running it
- Consider all possible control paths
- Use data flow analysis to track tainted data
- Use rules to find bugs
- Completeness is impossible; so what?
- False positives occur



# Static Analysis: Fortify SCA

**Source:** bootstrap.inc:290 Read $_SERVER['HTTP_HOST']()

```
288   $confdir = 'sites';
289   $uri = explode('/', $_SERVER['SCRIPT_NAME'] ? $_SERVER['SCRIPT_NAME'] :
      $_SERVER['SCRIPT_FILENAME']);
290   $server = explode('.', implode('.', array_reverse(explode(':',
      rtrim($_SERVER['HTTP_HOST'], '')))));
291   for ($i = count($uri) - 1; $i > 0; $i--) {
292     for ($j = count($server); $j > 0; $j--) {
```

**Sink:** file.inc:141 mkdir()

```
139   // Check if directory exists.
140   if (!is_dir($directory)) {
141     if (($mode & FILE_CREATE_DIRECTORY) && @mkdir($directory)) {
142       @chmod($directory, 0775); // Necessary for non-webserver users.
143     }
```

- `$base_path` based on HTTP_HOST
- Defined in bootstrap, passed to file.inc
- Static analysis identifies the flow
- False positive, but *just barely*

# Dynamic Analysis

- Operates during execution
- Consider control paths actually taken
- Tracks taint in interpreter
- False positives are unlikely
- PHP has no native support; patched interpreter required

# Tainting the database

acquia

- Is data from the database tainted or not?
  - user.email: validated pre-write, so it's safe
  - node.title: validated post-read, so it's not safe
- Per-column taint information in schema
  - On read, set taint property on any data coming from a tainted column
  - On write, verify lack of taint on any data going to a tainted column
  - Preserves taint properties across the database
- Requires (?) dynamic, not static, analysis

# Taint Drupal

- Based on Taint PHP by Wietse Venema
  - Defines fixed set of taint bits
    - TC_HTML, _SQL, _PCRE, ..., _ALL
  - Taint-enables a subset of PHP and ext functions
- database.taintmysqli.inc does db tainting
- Schema changes:

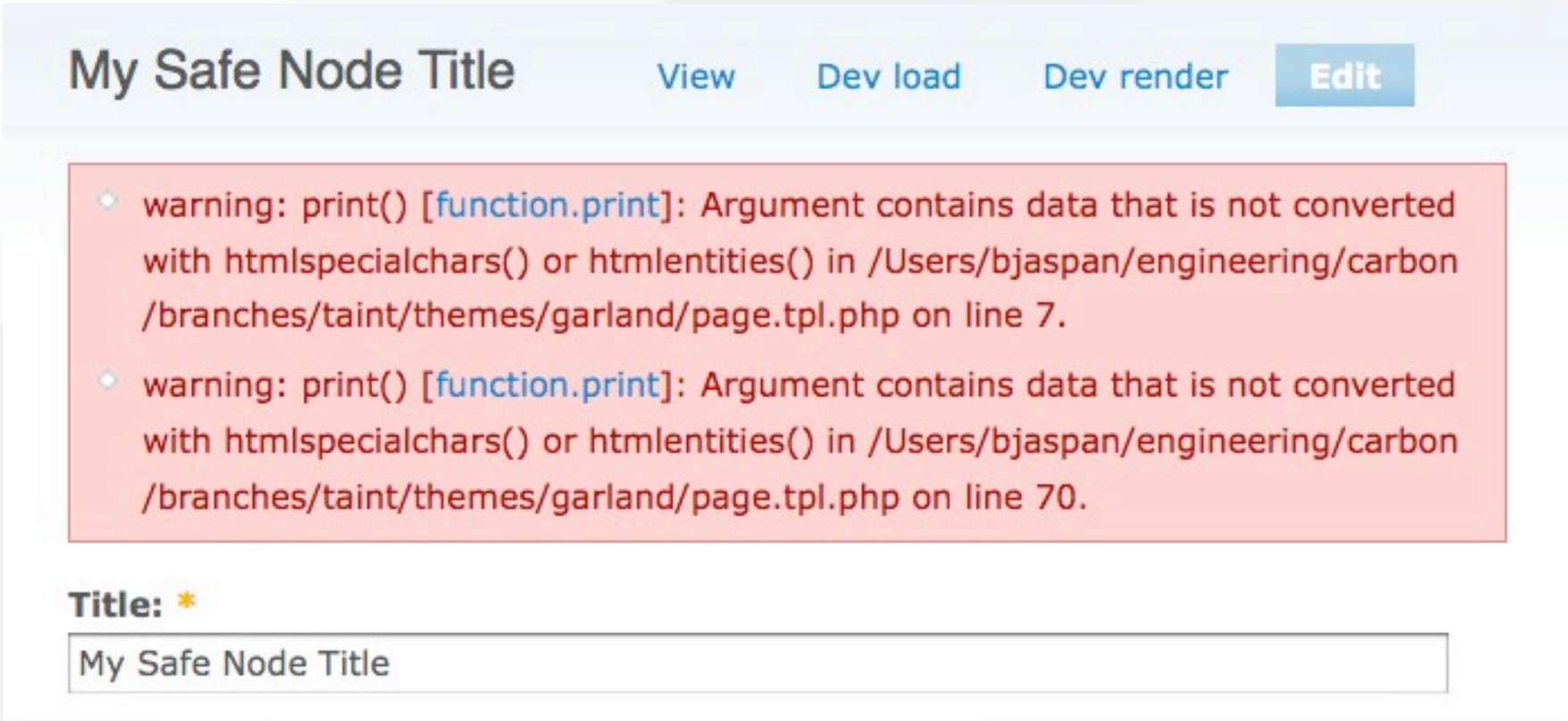
```
function taint_schema_alter($schema) {
  // title contains raw text from the user
  $schema['node']['fields']['title']['taint'] =
     TC_ALL;
  // message can contain the path but must be
  // check_plain()ed before INSERT
  $schema['watchdog']['fields']['message']['taint'] =
     TC_ALL & ~TC_HTML;
}
```

# Taint Drupal: Node title

- This bug was in Drupal 6.0:

```
function node_page_edit($node) {
  drupal_set_title($node->title); // XSS!
  return drupal_get_form($node->type .'_node_form',
                         $node);
}
```

- Taint Drupal catches it automatically:



# Taint Drupal: Simpletests

acquia

- Taint Drupal is integrated with Simpletest
  - Taint-checks all covered code paths automatically
  - Logs all taint errors as Fails
  - Simulated penetration testing not needed
  - Eventually, this can wipe out XSS, SQLi, and more
    - ... especially if used in production with hard failures

# Taint Drupal: Status

- Requires patched & re-built PHP interpreter
- Taint PHP is experimental and incomplete
  - Further work may require a fork
- Taint Drupal is also incomplete
  - Needs taint settings for columns of every table in the schema (core and contrib)
  - Assumes TC_NONE unless told otherwise, so you can "page in" taint bits as you get false errors
- Not yet publicly available

# Demo!

# Questions?
# (for real this time)

Barry Jaspan
Acquia, Inc.
barry.jaspan@acquia.com