

Magento Security

How to break the code

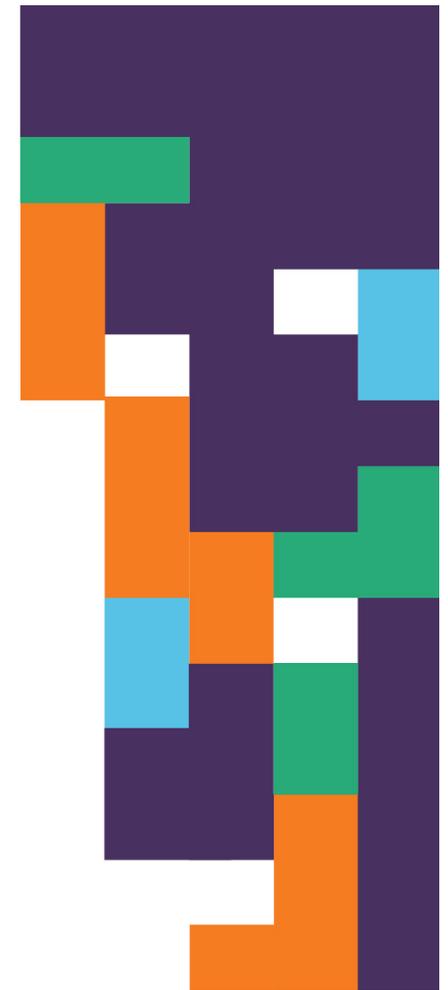


Bastian Ike
Webdeveloper

_bastian ike

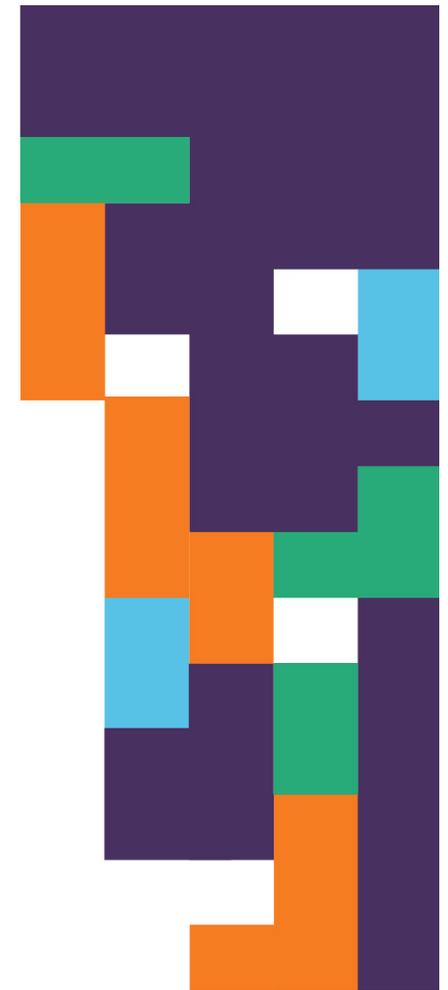
- ❑ Magento since 08/2010
- ❑ Certified Developer since 11/2011
- ❑ IT-Security (as a hobby) since 2007

- ❑ Apprenticeship at Hucke Media



_magento security

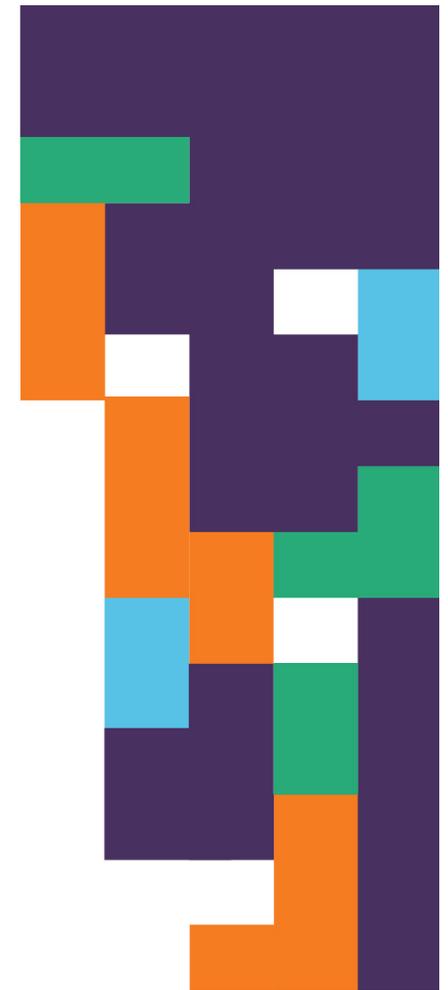
- Hide your admin
- Encrypt your data
- Use strong passwords
- Set file permissions
- And so on...



_main topics

- Definition of „Security“
- Security in E-Commerce and Magento
- Common issues
- Best practices

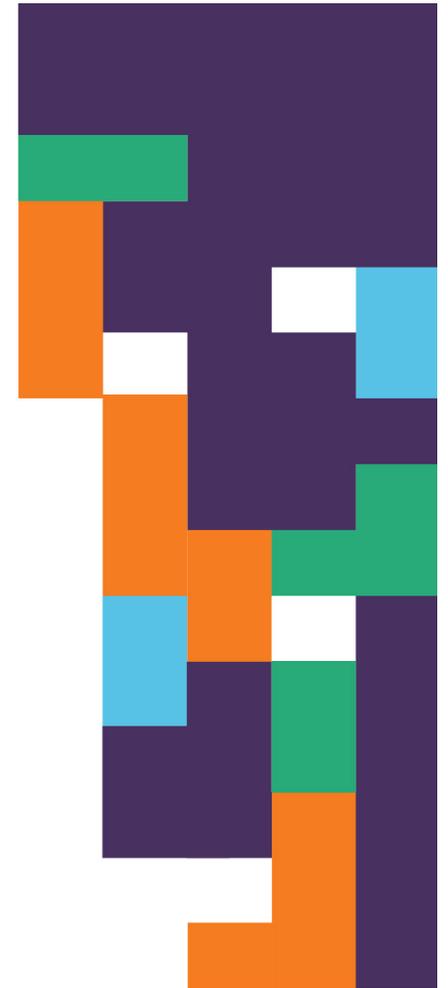
- Example attacks



_what does „security“ means?

„Security is the degree of protection against danger, damage, loss and crime. Security as a form of protection are structures and processes that provide or improve security as a condition“

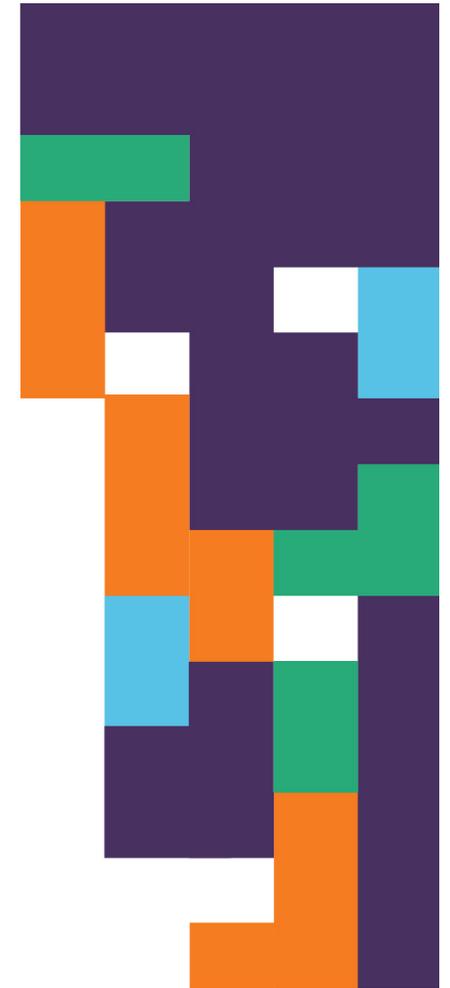
~Wikipedia



imagine 2012
Magento Conference

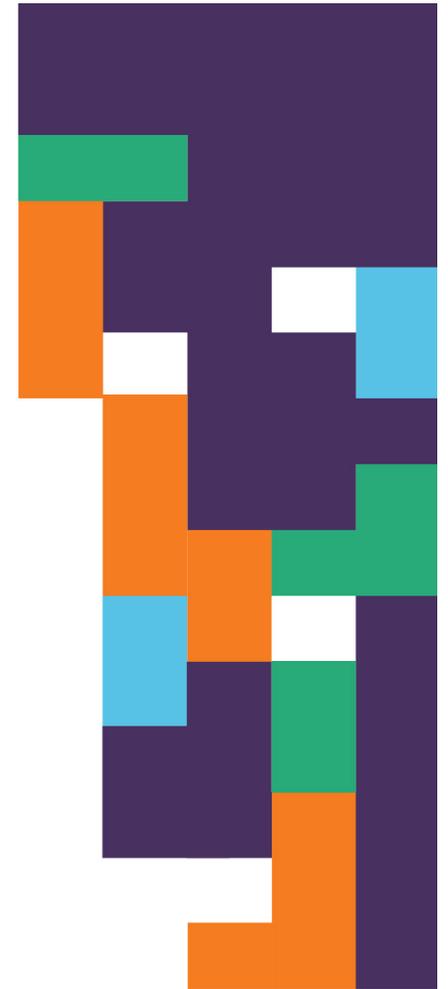
_computer security

- Very new topic
- First attacks by the Morris worm in 1988
- Based on logical problems, not „weak“ infrastructure
- Absolutely not as shown in movies like
 - Matrix
 - Password:Swordfish
 - Hackers



_thebod's golden giftcard

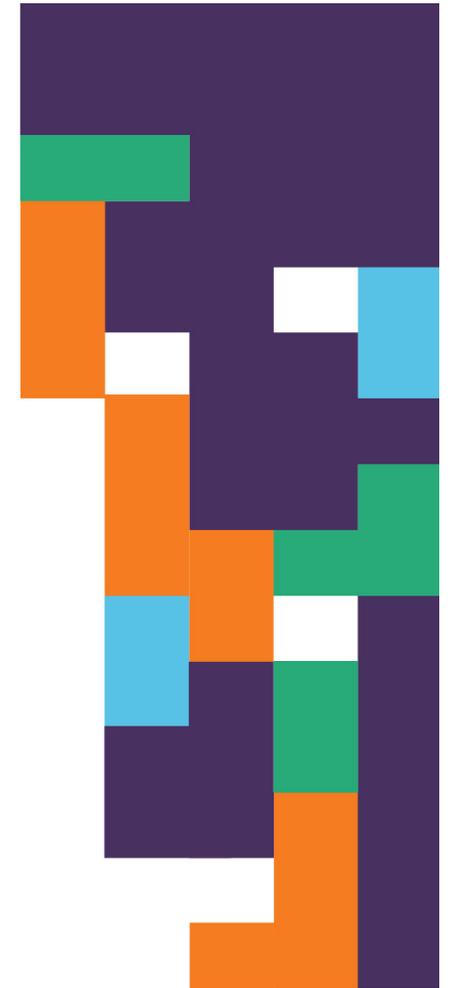
Example



imagine 2012
Magento Conference

_ecommerce security

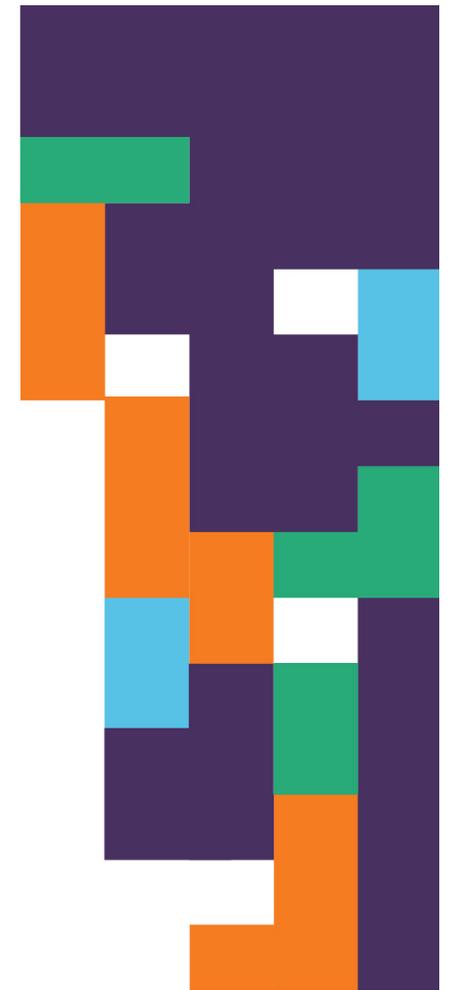
- ❑ Worldwide accessible
- ❑ An attacker needs a computer, not a gun
- ❑ Possible loot: credit cards, private data, etc...



_thebod's golden giftcard

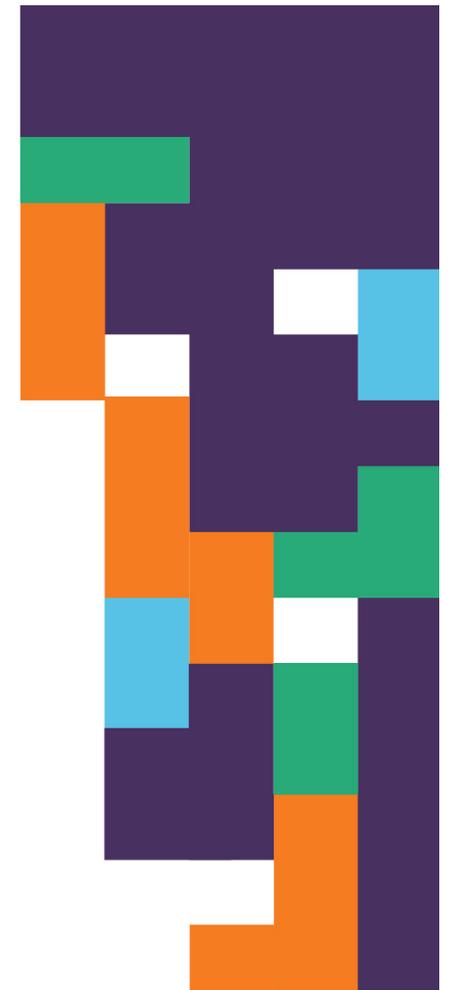
Abstract description

- ❑ `Payment_Module_Model_Event`
`::_validateData()`
- ❑ `$merchant`, `$orderid`, `$secret`, `$amount`,
`$currency`, `$status`
- ❑ `$merchant` and `$secret` are empty by default



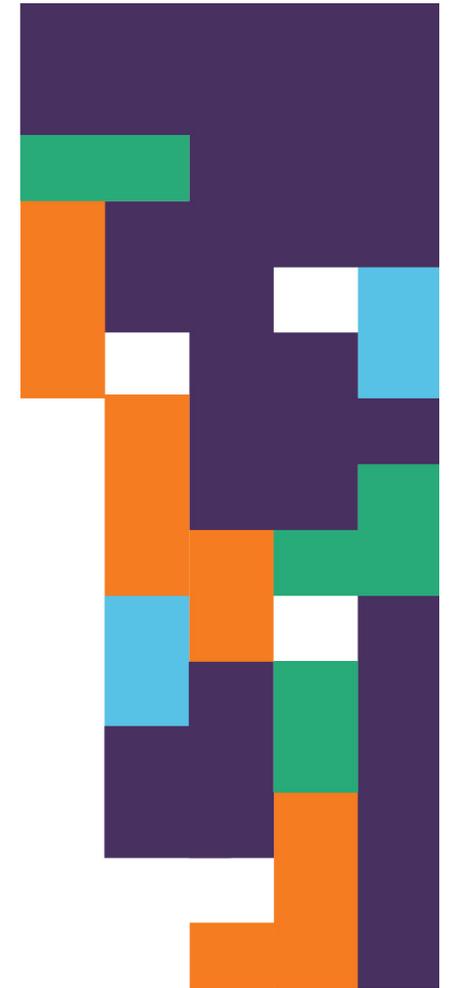
_thebod's golden giftcard

```
$key = md5(md5($secret));  
$key .= md5($key . $merchant);  
$key .= md5($key . $orderid);  
$key .= md5($key . $amount);  
$key .= md5($key . $currency);
```



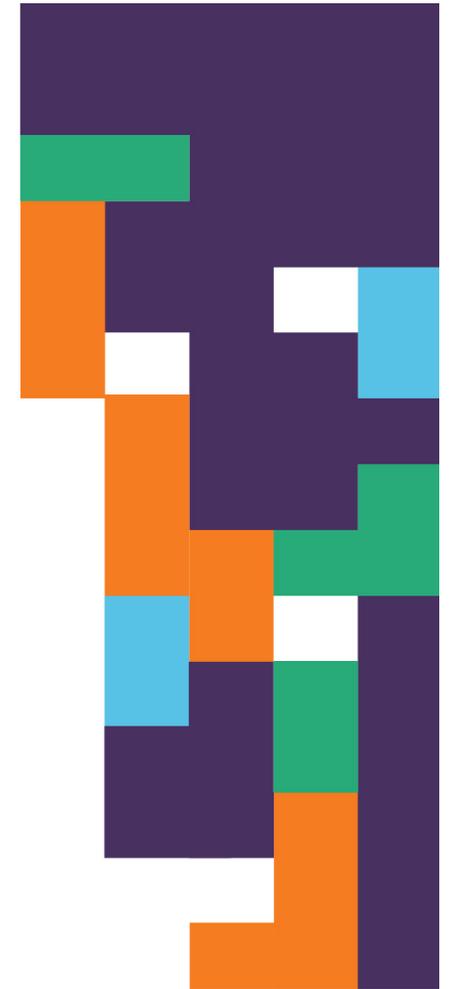
_thebod's golden giftcard

```
$key = md5(md5(md5('')));  
$key .= md5($key . $orderid);  
$key .= md5($key . $amount);  
$key .= md5($key . $currency);
```



_security in magento

Overview

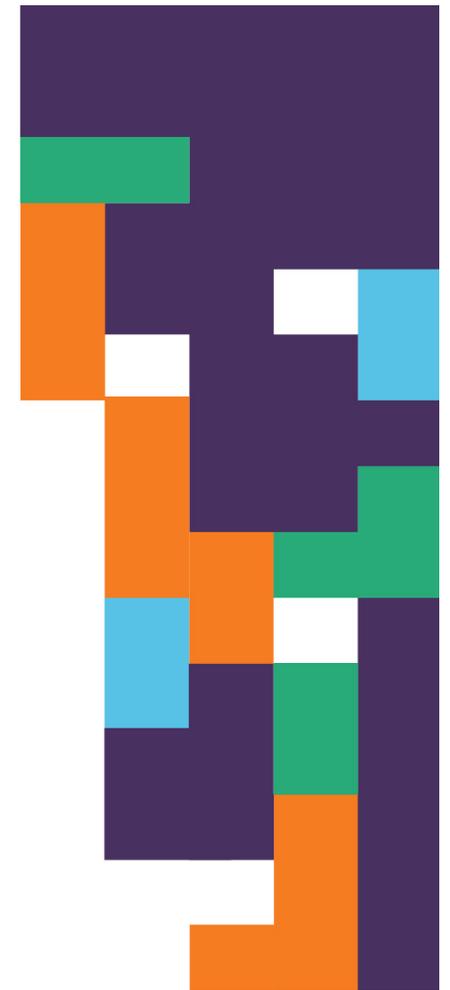


imagine 2012
Magento Conference

_complex sourcecode

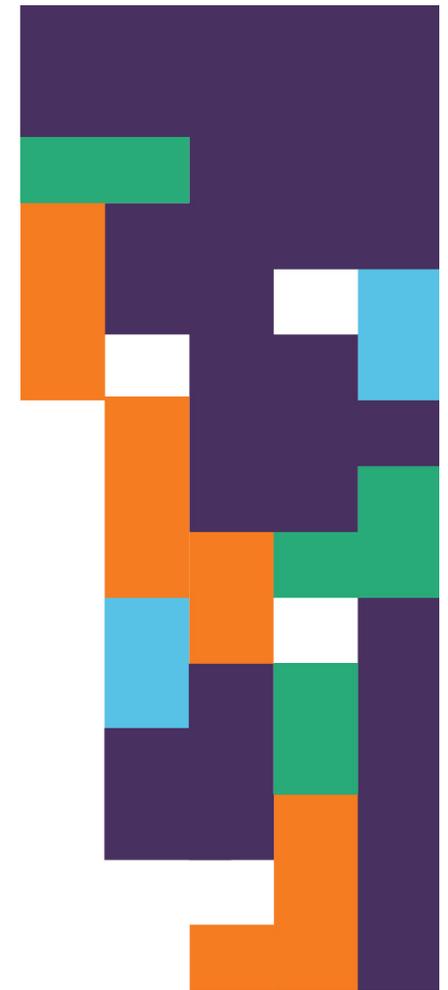
Magento 1.7.0.0-rc1

- ❑ 1,978,267 lines of code
- ❑ 11,891 files
- ❑ 337 database tables



_merchants

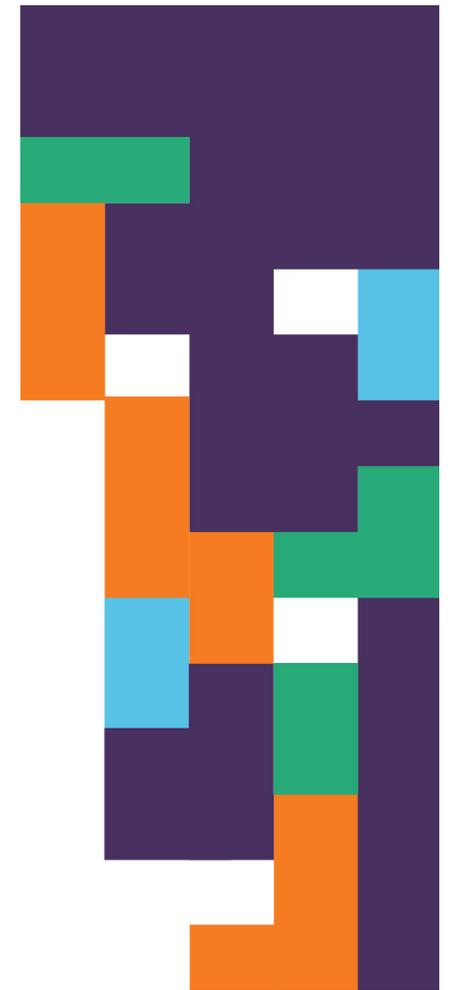
- ❑ Many possible targets
- ❑ A lot of transaction data (credit card data)
- ❑ A lot of customer data (matching names)



_developers

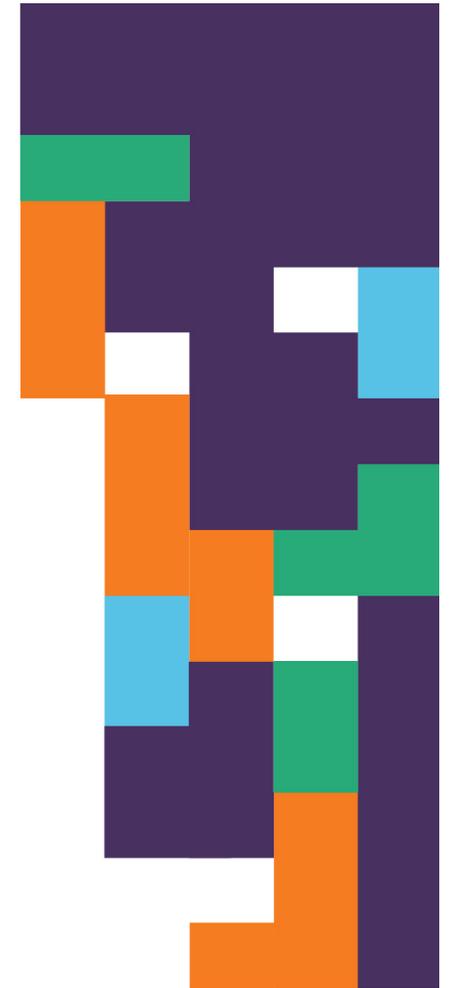
- Security is a very unknown topic
- Not many known attacks as references
- Very small information about advanced security

- Magento 2 will be checked against OWASP top 10
- Security team since 02/2011



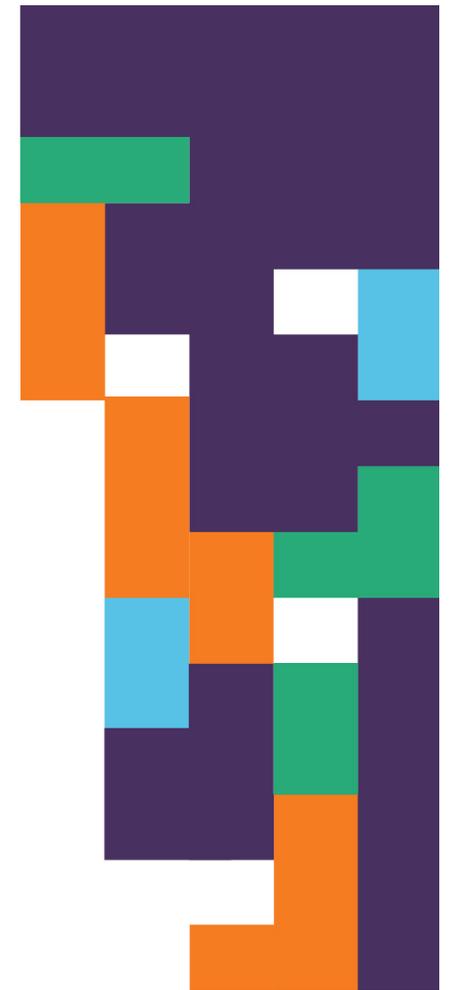
_magento connect

- ❑ +5,000 extensions
- ❑ A lot of (insecure) payment modules
- ❑ Checking everything is impossible
- ❑ Encrypted sourcecode



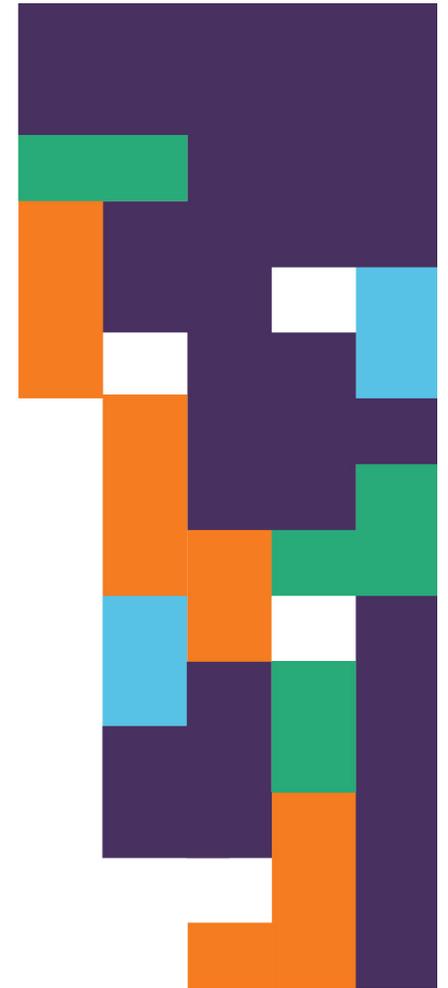
_best targets

- Payment modules
- Price rules / Reward modules
- Core infrastructure



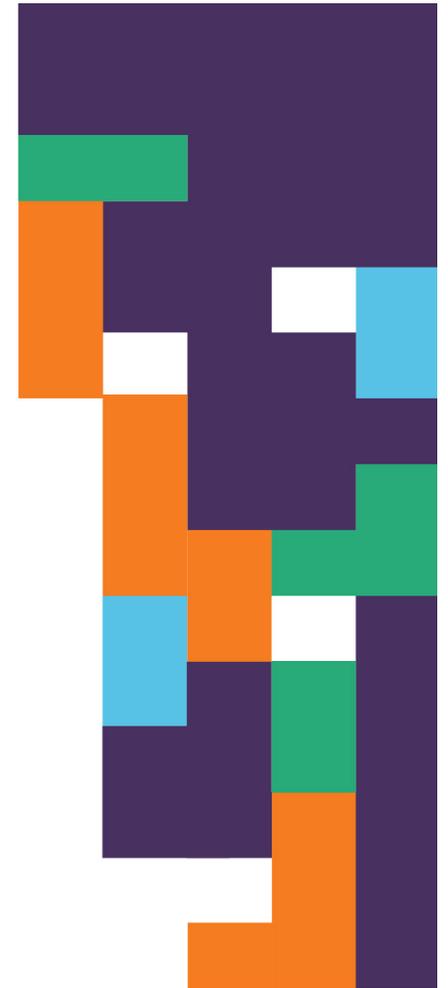
_api exploit

Example API attack



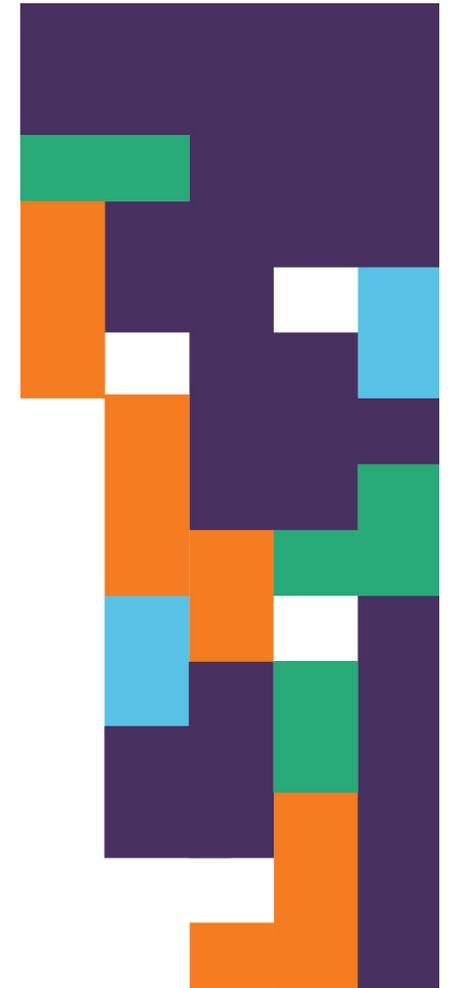
`_api exploit`

Abstract explanation



_common issues

Common attacks on
PHP websoftware



imagine 2012
Magento Conference

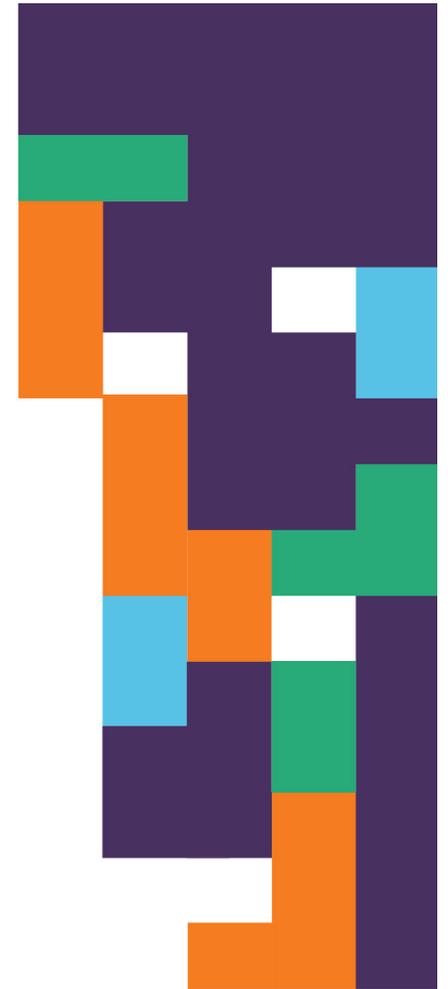
_mysql injection

- ❑ Well-known, but still very dangerous
- ❑ Based on the injection of special-crafted MySQL code into a query
- ❑ Example:

```
<?php
```

```
// [mysql connect...]
```

```
mysql_query('SELECT field1, field2 FROM table1  
WHERE id=' . $_GET['entry']);
```



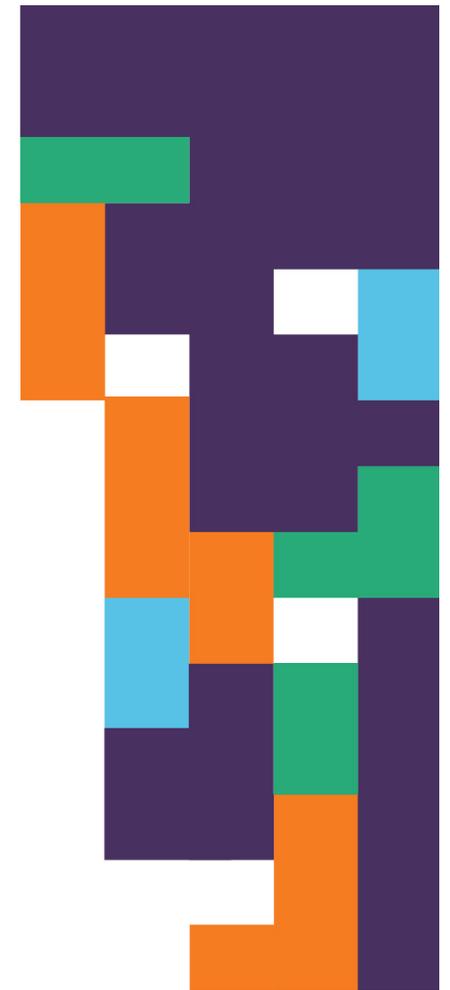
_mysql injection

- ❑ Well-known, but still very dangerous
- ❑ Based on the injection of special-crafted MySQL code into a query
- ❑ Example:

```
<?php
```

```
// [mysql connect...]
```

```
mysql_query('SELECT field1, field2 FROM table1  
WHERE id=1');
```



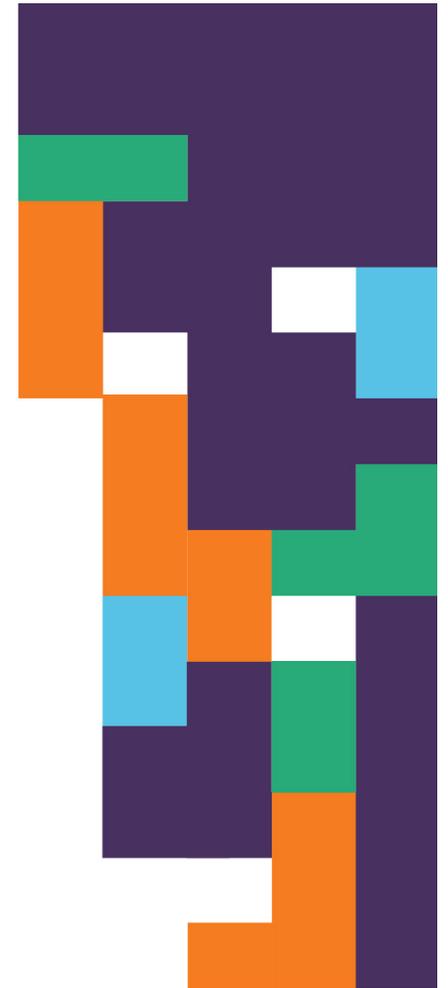
_mysql injection

- ❑ Well-known, but still very dangerous
- ❑ Based on the injection of special-crafted MySQL code into a query
- ❑ Example:

```
<?php
```

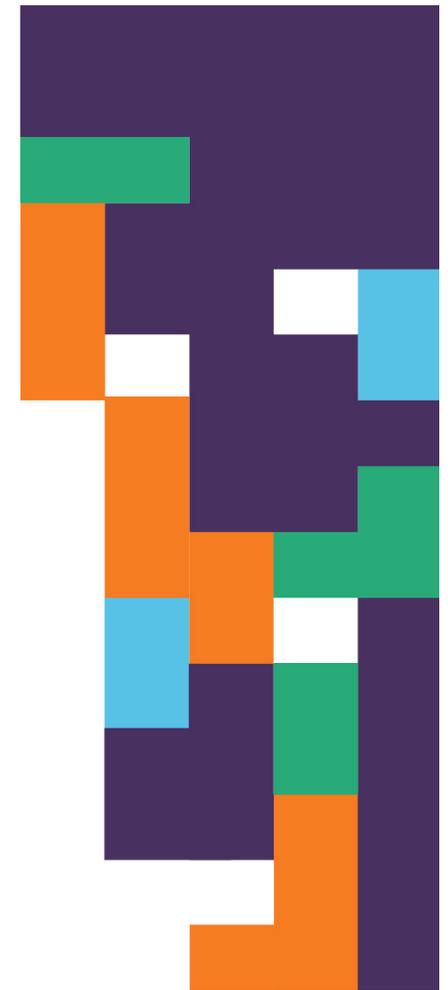
```
// [mysql connect...]
```

```
mysql_query('SELECT field1, field2 FROM table1  
WHERE id=-1 UNION SELECT username AS field1,  
password AS field2 FROM users WHERE admin=1 LIMIT  
1');
```



_XSS

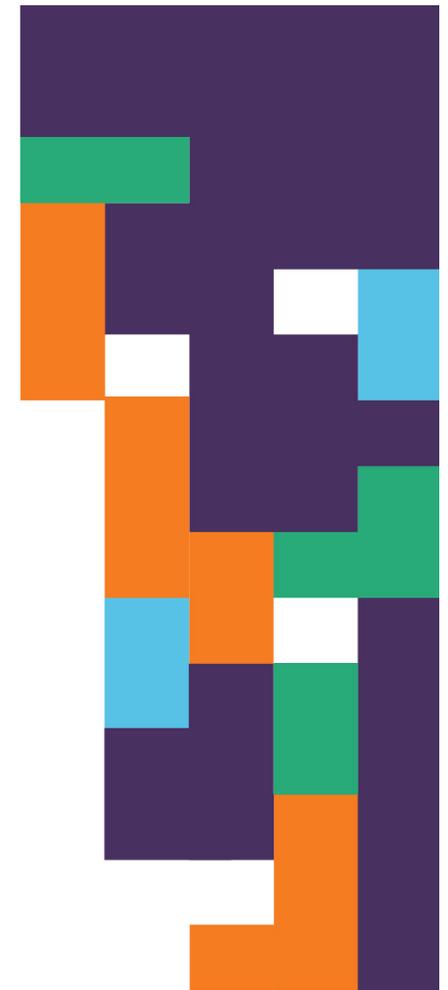
- Cross Site Scripting
- Based on JavaScript injection into homepages
- Non-Persistent:
 - Social engineering needed
- Persistent:
 - More dangerous, no SE needed
- Possible attack: creating new administrators by simulating browser requests



_XSS

□ Example:

```
<!-- html-header, form tags, etc... -->  
<!-- inside a search form: -->  
Search for: <input name="q" value="<?php  
Mage::getRequest()->getParam('q') ?>" />  
<!-- submit button etc... -->
```



_XSS

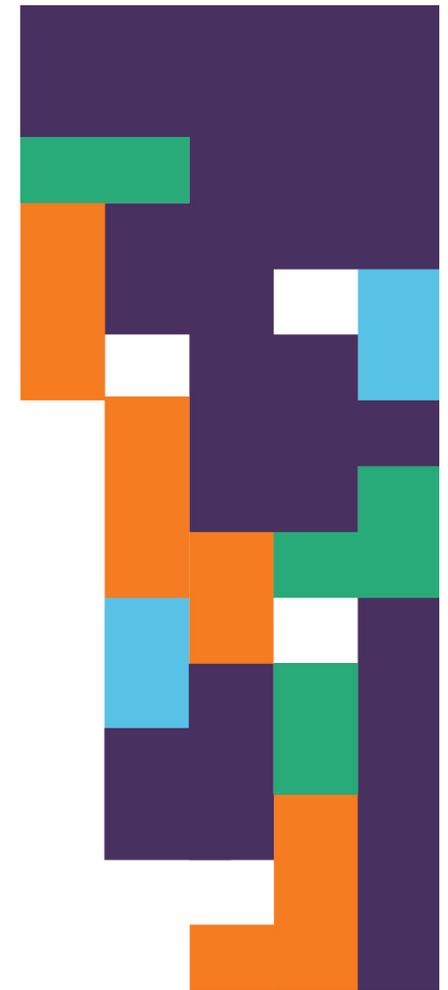
□ Example:

```
<!-- html-header, form tags, etc... -->
```

```
<!-- inside a search form: -->
```

```
Search for: <input name="q" value="test" />
```

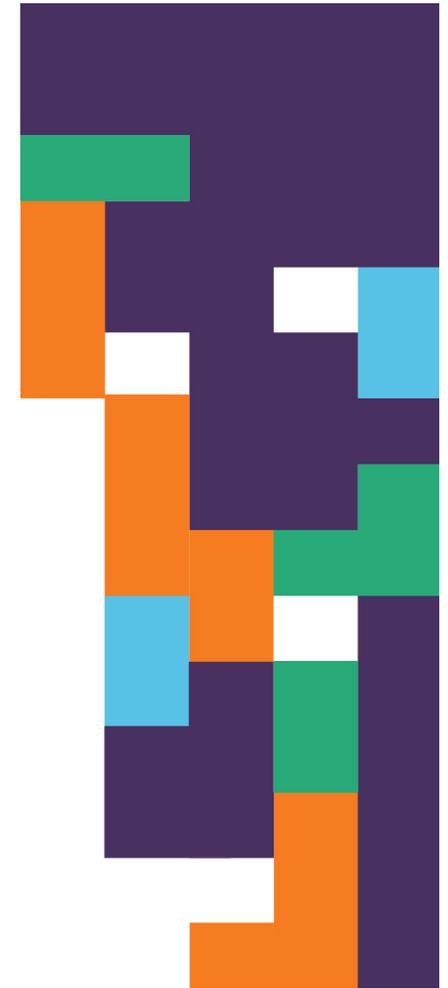
```
<!-- submit button etc... -->
```



_XSS

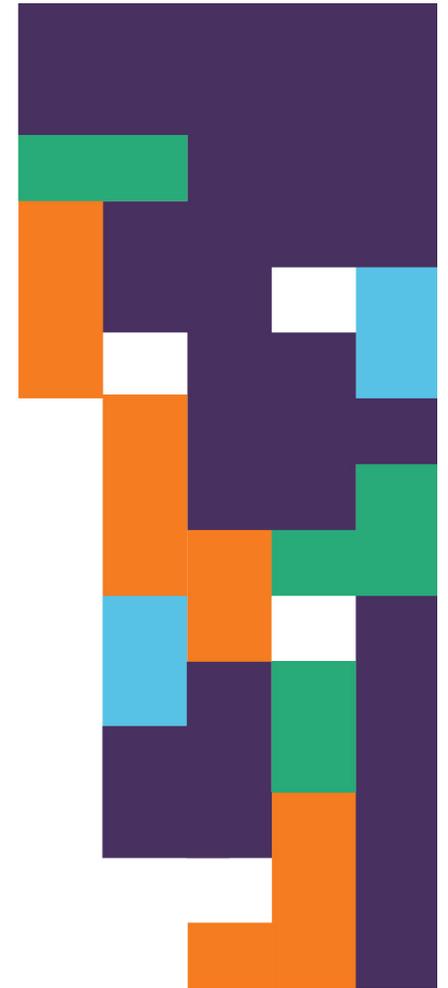
□ Example:

```
<!-- html-header, form tags, etc... -->  
<!-- inside a search form: -->  
Search for: <input name="q"  
value=""><script>alert("JS code injection")</  
script><" />  
<!-- submit button etc... -->
```



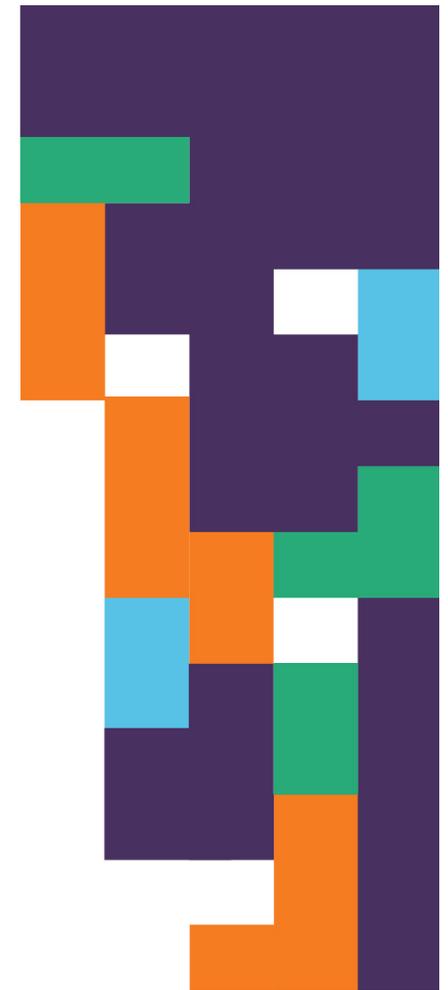
_unknown attack vectors

unserialize() and simplexml_load_string()



`_unserialize()`

- ❑ Transform serialized data from a string into an Array or an Object
- ❑ `unserialize()` on objects calls
 - `__wakeup()`
 - `__destruct()`
- ❑ Magento and Zend contain a few abusable destructors...
- ❑ ...but Magento don't use `unserialize()` on user input, but maybe extensions



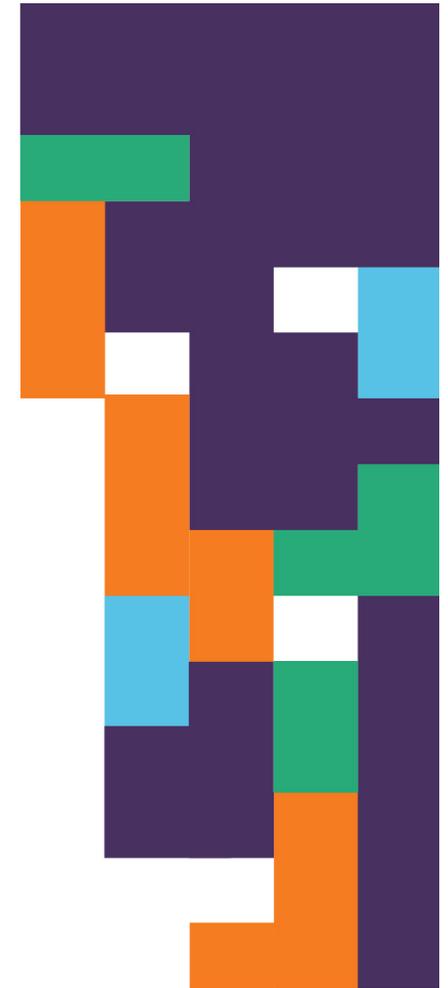
__simplexml_load_string()

- ❑ Doctype entities + protocol wrapper results in file disclosure
- ❑ Can be used to read files
- ❑ Good point: not applicable on the Magento core
- ❑ Example:

```
<?xml version="1.0" ?>
```

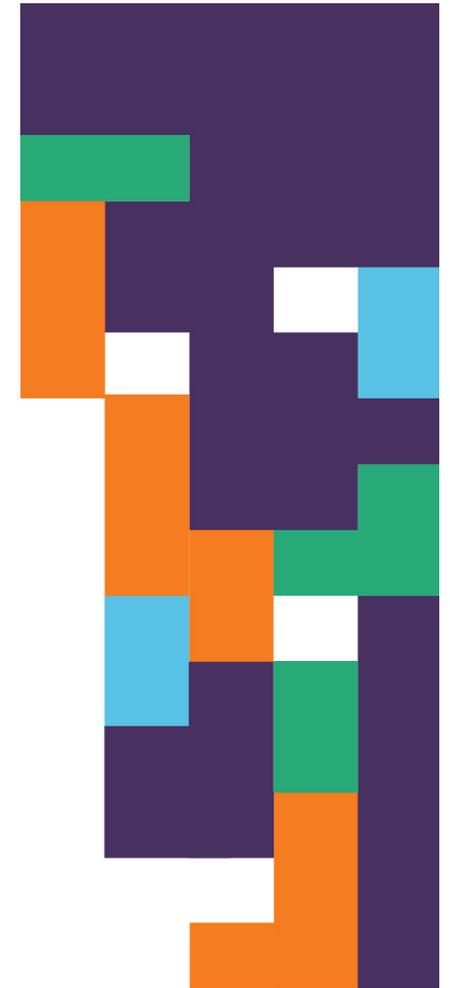
```
<!DOCTYPE foo [ <!ELEMENT foo ANY > <!ENTITY xxe  
SYSTEM "file:///etc/passwd" >]>
```

```
<foo>&xxe;</foo>
```



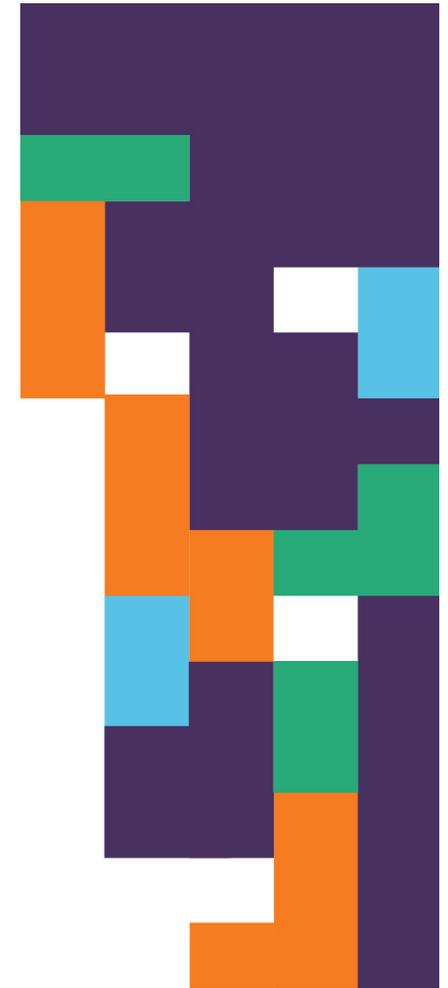
_best practices

- Review the code from the point of the attacker
- What would you do to hack your software?
- Four-eyes principle review
- An attacker doesn't care about complex exploits
- Use frameworks and follow there coding guidelines



_issue found?

- Full-, or half-disclosure?
- Would you like to see exploits for your software online?
- Inform the vendor...
- ...or ask me ;-)

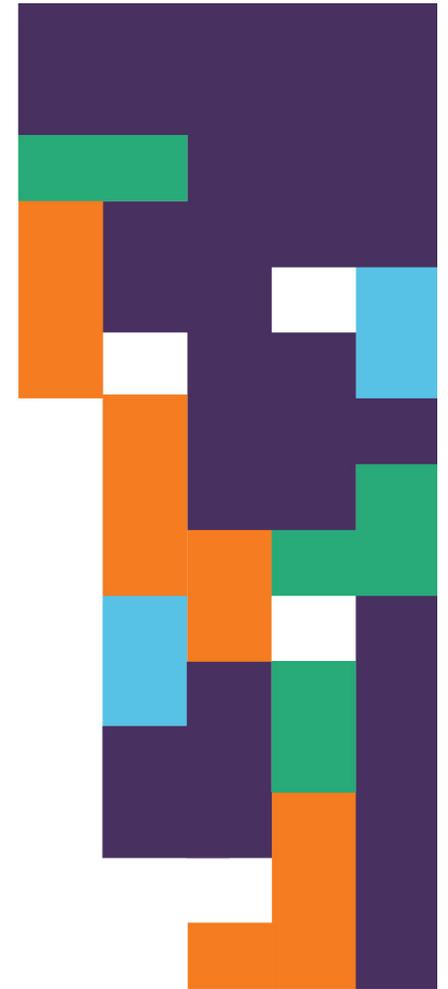


conclusion

**Software can't be safe, but we
can make it hard to exploit it**

and

Never trust the user input!



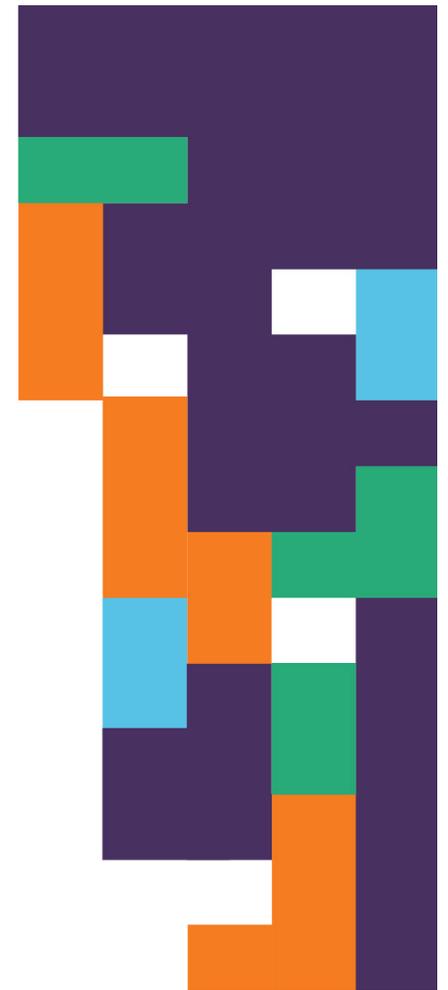
_thank you!

...questions?

Bastian Ike

Twitter: @b_ike

Mail: b-ike@b-ike.de



imagine 2012
Magento Conference