Code Reuse Attacks in PHP: Automated POP Chain Generation

Johannes Dahse, Nikolai Krein, and Thorsten Holz
Ruhr-University Bochum

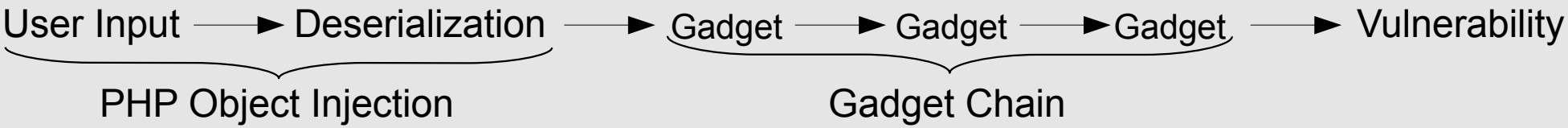
ACM CCS ’14, 3-7 November 2014, Scottsdale, AZ, USA

1. Code Reuse Attacks in PHP

- Code reuse attacks are known for memory corruption vulnerabilities
- They base on reusing existing code fragments
- Also a viable attack vector against PHP applications
- First demonstrated by Stefan Esser, 2009
- Using a *PHP Object Injection* (POI) vulnerability to trigger gadget chains

WordPress
(23.2% of all websites)

Joomla!™
(3% of all websites)



1.1 PHP's (De)Serialization

- PHP built-in functions
  serialize() / unserialize()
- Transform any data type
  to an unified string format
- Allows to store PHP values
  without loosing the structure

1.1 PHP's (De)Serialization

- PHP built-in functions
  serialize() / unserialize()
- Transform any data type
  to an unified string format
- Allows to store PHP values
  without loosing the structure

```
1  class Text {
2      public function __construct($data) {
3          $this->data = $data;
4      }
5  }
6
7  $object1 = new Text('CCS14');
8  $string = serialize($object1);
```

```
O:4:"Text":1:{s:4:"data";s:5:"CCS14";}
```

```
9  $object2 = unserialize($string);
10 echo $object2->data;
```

```
CCS14
```

1.2 PHP Object Injection

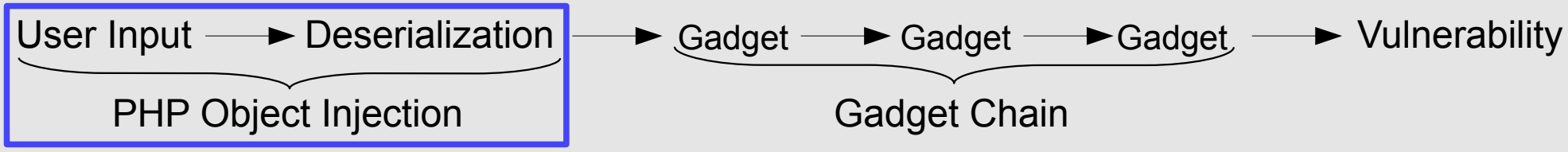
- *PHP Object Injection* (POI) Vulnerability occurs when user data is unserialize'd

```
1  class Text {
2      public function __construct($data) {
3          $this->data = $data;
4      }
5  }
6
7  $object1 = new Text('CCS14');
8  $_COOKIE['text'] = serialize($object1);
```

```
O:4:"Text":1:{s:4:"data";s:5:"CCS14";}
```

```
9  $object2 = unserialize($_COOKIE['text']);
10 echo $object2->data;
```

```
CCS14
```



1.2 PHP Object Injection

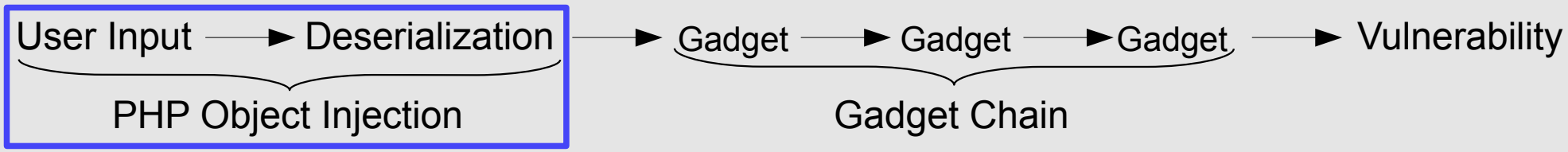
- *PHP Object Injection* (POI) Vulnerability occurs when user data is unserialize'd
- Attacker can inject arbitrary data types and values
- Severity depends on flow of injected data and *gadgets*

```
1  class Text {
2      public function __construct($data) {
3          $this->data = $data;
4      }
5  }
6
7  $object1 = new Text('CCS14');
8  $_COOKIE['text'] = serialize($object1);
```

```
O:4:"Text":1:{s:4:"data";s:5:"CCS14";}
O:6:"FooBar":1:{s:4:"data";s:3:"XSS";}
```

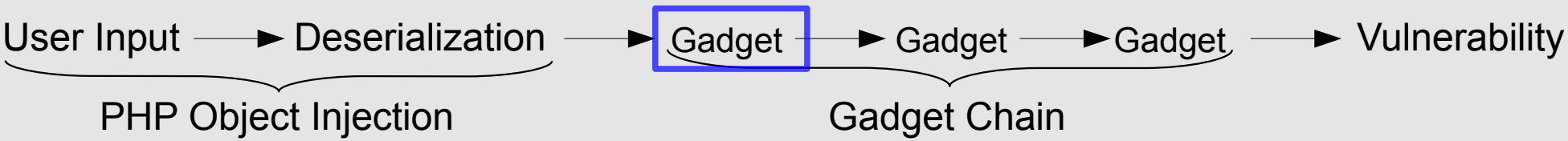
```
9  $object2 = unserialize($_COOKIE['text']);
10 echo $object2->data;
```

```
XSS
```



1.3 Magic Methods

- 15 special purpose methods starting with __
- For example __construct(), **__destruct**(), __toString(), **__wakeup**(), __isset()
- Some magic methods are invoked **automatically** on deserialization !

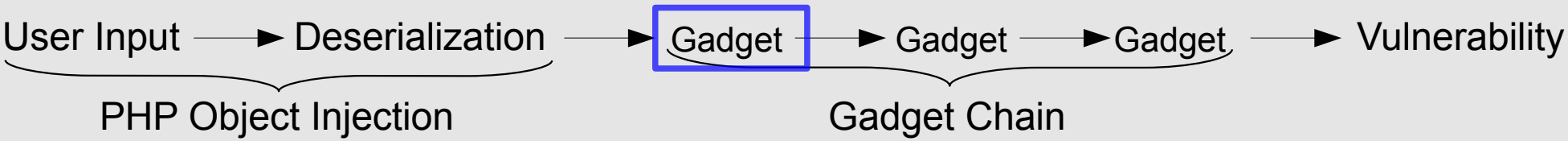


1.3 Magic Methods

- 15 special purpose methods starting with __
- For example __construct(), **__destruct**(), __toString(), **__wakeup**(), __isset()
- Some magic methods are invoked **automatically** on deserialization !

```
O:4:"Text":1:{s:4:"data";s:5:"CCS14";}
```

```
 9  $object2 = unserialize($_COOKIE['text']);
10
11      ...
```



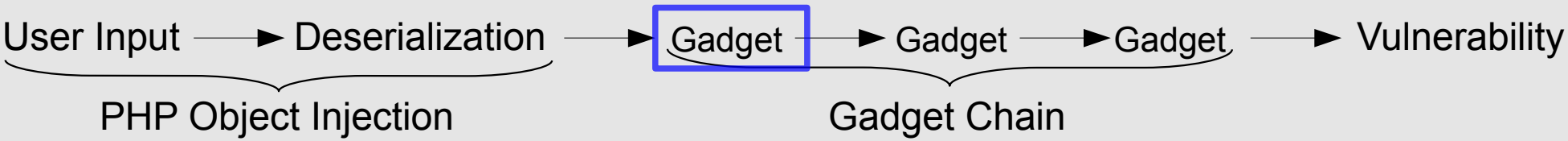
1.3 Magic Methods

- 15 special purpose methods starting with __
- For example __construct(), **__destruct**(), __toString(), **__wakeup**(), __isset()
- Some magic methods are invoked **automatically** on deserialization !

```
1   class TempFile {
2
3       ...
4       public function __destruct() {
5           unlink($this->file);
6       }
7       ...
8   }
```

```
O:4:"Text":1:{s:4:"data";s:5:"CCS14";}
```

```
9   $object2 = unserialize($_COOKIE['text']);
10
11      ...
```



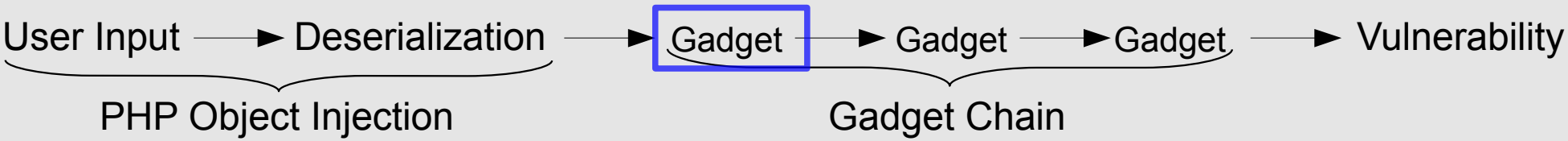
1.3 Magic Methods

- 15 special purpose methods starting with __
- For example __construct(), **__destruct**(), __toString(), **__wakeup**(), __isset()
- Some magic methods are invoked **automatically** on deserialization !

```
1   class TempFile {
2
3       ...
4       public function __destruct() {
5           unlink($this->file);
6       }
7       ...
8   }
```

```
O:4:"Text":1:{s:4:"data";s:5:"CCS14";}
O:8:"TempFile":1:{s:4:"file";s:9:".htaccess";}
```

```
9   $object2 = unserialize($_COOKIE['text']);
10
11      ...
```



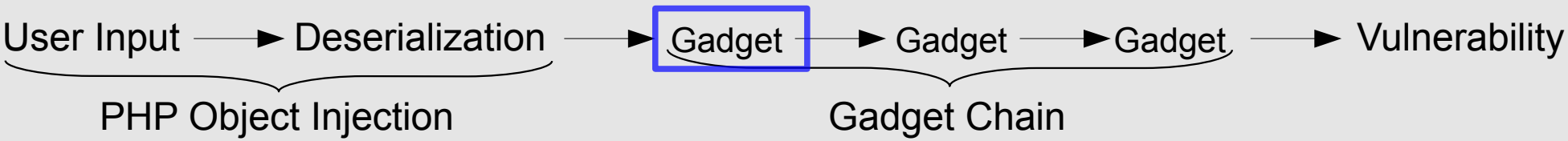
1.3 Magic Methods

- 15 special purpose methods starting with __
- For example __construct(), __destruct(), **__toString**(), __wakeup(), **__isset**()
- Some magic methods are invoked on specific **events**

```
1   class TempFile {
2
3       ...
4       public function __destruct() {
5           unlink($this->file);
6       }
7       ...
8   }
```

```
O:4:"Text":1:{s:4:"data";s:5:"CCS14";}
O:8:"TempFile":1:{s:4:"file";s:9:".htaccess";}
```

```
9   $object2 = unserialize($_COOKIE['text']);
10  if(isset($object2)) {
11      ...
```

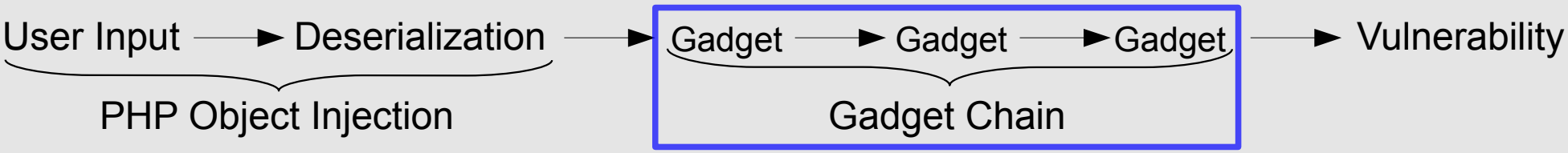


1.4 Property Oriented Programming

- Magic methods are the *initial gadgets*
- They might call other methods (*gadgets*)
- We control all properties

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

```
1  class Process {
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```



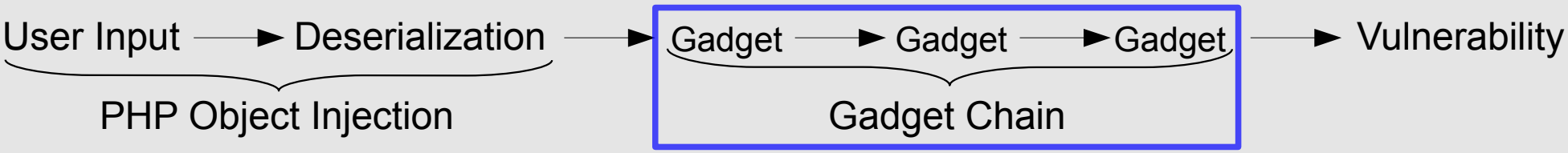
1.4 Property Oriented Programming

- Magic methods are the *initial gadgets*
- They might call other methods (*gadgets*)
- We control all properties

```
O:8:"TempFile":0:{};
```

```php
class TempFile {
    public function __destruct() {
        $this->shutdown();
    }
    public function shutdown() {
        $this->handle->close();
    }
}
```

```php
class Process {
    public function close() {
        system('kill ' . $this->pid);
    }
}
```



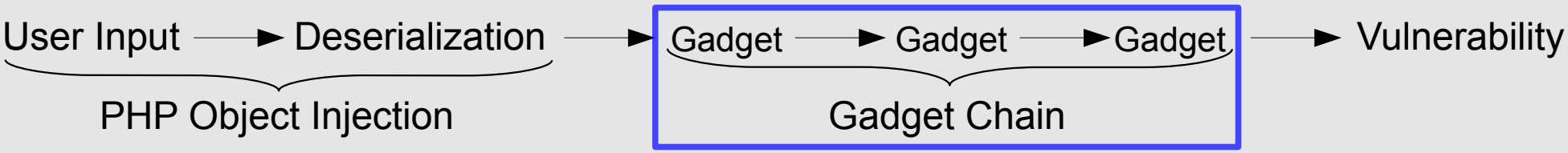
1.4 Property Oriented Programming

- Magic methods are the *initial gadgets*
- They might call other methods (*gadgets*)
- We control all properties

```
O:8:"TempFile":0:{};
```

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

```
1  class Process {
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```



1.4 Property Oriented Programming

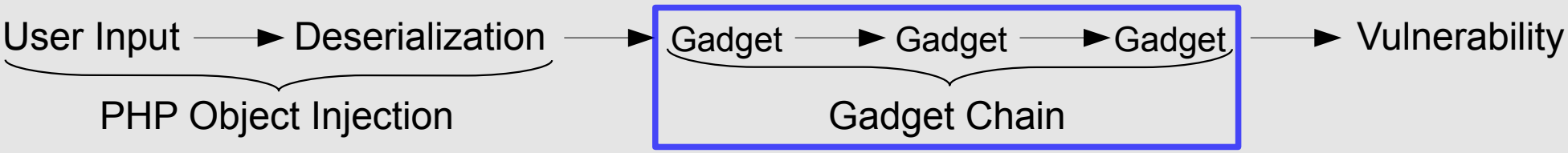
- Magic methods are the *initial gadgets*
- They might call other methods (*gadgets*)
- We control all properties

```
O:8:"TempFile":1:{
    s:5:"handle";O:7:"Process":0:{};
};
```

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

Process Object

```
1  class Process {
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```



1.4 Property Oriented Programming

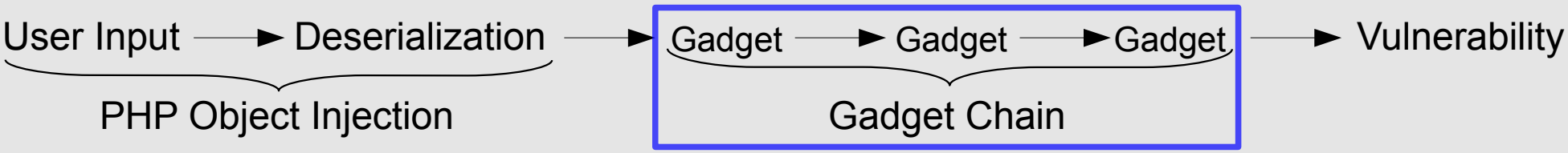
- Magic methods are the *initial gadgets*
- They might call other methods (*gadgets*)
- We control all properties

```
O:8:"TempFile":1:{
    s:5:"handle";O:7:"Process":0:{};
};
```

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

Process Object

```
1  class Process {
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```



1.4 Property Oriented Programming

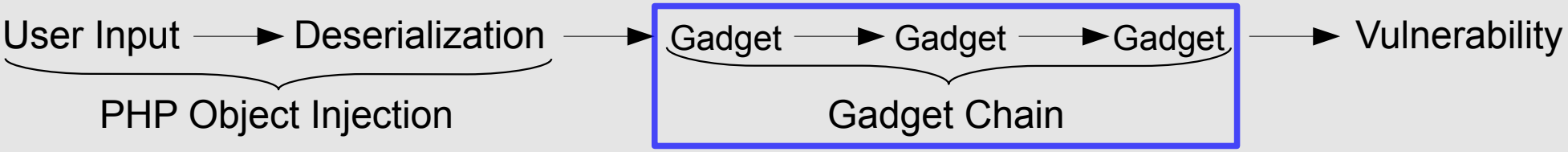
- Magic methods are the *initial gadgets*
- They might call other methods (*gadgets*)
- We control all properties

```
O:8:"TempFile":1:{
    s:5:"handle";O:7:"Process":1:{
        s:3:"pid";s:10:";touch ccs"
    };
};
```

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

Process Object

```
1  class Process {
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```



1.4 Property Oriented Programming

- Magic methods are the *initial gadgets*
- They might call other methods (*gadgets*)
- We control all properties

```
O:8:"TempFile":1:{
    s:5:"handle";O:7:"Process":1:{
        s:3:"pid";s:10:";touch ccs"
    };
};
```

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

Process Object

```
1  class Process {
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```

kill ;touch ccs

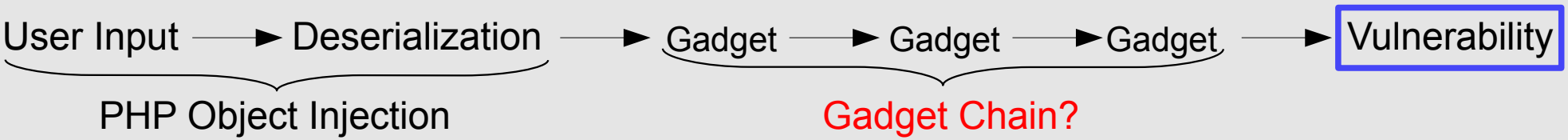
User Input → Deserialization → Gadget → Gadget → Gadget → Vulnerability

PHP Object Injection

Gadget Chain

1.5 POI Vulnerability Examples

CVE	Software	Version	Severity
CVE-2014-2294	Open Web Analytics	1.5.6	unknown
CVE-2014-1860	Contao CMS	3.2.4	PHP Code Execution
CVE-2014-0334	CMS Made Simple	1.11.9	unknown
CVE-2013-4338	Wordpress	3.5.1	unknown
CVE-2013-1465	CubeCart	5.2.0	SQL injection
CVE-2013-1453	Joomla!	3.0.2	SQL injection, File Delete



1.6 Our Approach

- Static code analysis for PHP code
- Automatically detect POI vulnerabilities
- Automatically generate POP gadget chains
- Extend prototype using *block* and *function summaries*
- Challenge: modeling object-oriented PHP code

2. Implementation

2.1 Overview

- *Block* and *function summaries*
- Independet **one-time** analysis of blocks and functions
- *Summary* stores data flow result of each unit and can be reused
- We analyze data flow between connected units *backwards-directed*

```
1   function getCookie() {
2     if(isset($_COOKIE['text'])) {
3       $cookie = $_COOKIE['text'];
4       $s = $cookie;
5     }
6     else {
7       $cookie = null;
8       $s = $cookie;
9     }
10    return $s;
11  }
12
13  $c = getCookie();
```

2.1 Overview

- *Block* and *function summaries*
- Independet **one-time** analysis of blocks and functions
- *Summary* stores data flow result of each unit and can be reused
- We analyze data flow between connected units *backwards-directed*

```
function getCookie()

 3        $cookie = $_COOKIE['text'];
 4        $s = $cookie;

 7        $cookie = null;
 8        $s = $cookie;

10      return $s;

13   $c = getCookie();
```

2.1 Overview

- *Block* and *function summaries*
- Independet **one-time** analysis of blocks and functions
- *Summary* stores data flow result of each unit and can be reused
- We analyze data flow between connected units *backwards-directed*

```
function getCookie()

        $s = $_COOKIE['text'];

        $s = null;

10      return $s;

13  $c = getCookie();
```

2.1 Overview

- *Block* and *function summaries*
- Independet **one-time** analysis of blocks and functions
- *Summary* stores data flow result of each unit and can be reused
- We analyze data flow between connected units *backwards-directed*

```
function getCookie()




return ($_COOKIE['text'] | null)
```

```
13   $c = getCookie();
```

2.2 Challenge: OOP

- No prototype in related work supports object-oriented PHP code analysis
- Object-oriented code disallows *independent* analysis of units

```
1  class UserInput {
2      public function __construct() {
3          $this->c = $_COOKIE['test'];
4      }
5      public function get() {
6          return $this->c;
7      }
8  }
9
10 $input = new UserInput();
11 if(is_object($input)) {
12     $c = $input->get();
13 }
```

2.2 Challenge: OOP

- No prototype in related work supports object-oriented PHP code analysis
- Object-oriented code disallows *independent* analysis of units
- **Our approach:** Assist *backwards-directed* data flow analysis with *forwards-directed* data propagation

```
1  class UserInput {
2      public function __construct() {
3          $this->c = $_COOKIE['test'];
4      }
5      public function get() {
6          return $this->c;
7      }
8  }
9
10 $input = new UserInput();
11 if(is_object($input)) {
12     $c = $input->get();
13 }
```

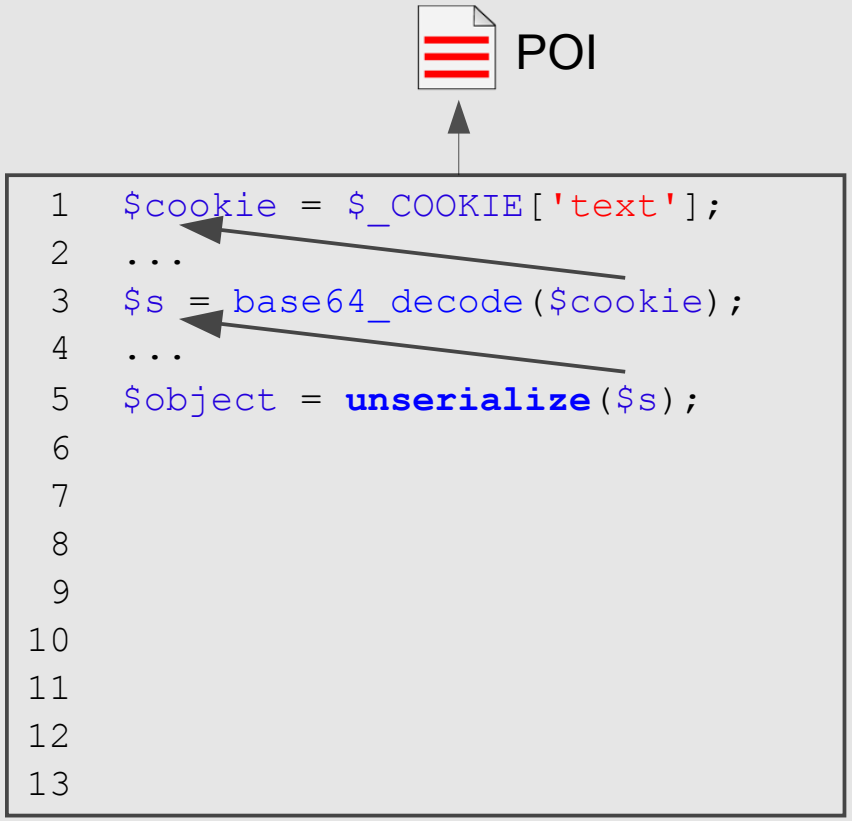
2.2 Challenge: OOP

- No prototype in related work supports object-oriented PHP code analysis
- Object-oriented code disallows *independent* analysis of units
- **Our approach:** Assist *backwards-directed* data flow analysis with *forwards-directed* data propagation
- For full details, please refer to our paper

```
1  class UserInput {
2      public function __construct() {
3          $this->c = $_COOKIE;
4      }
5      public function get($key) {
6          return $this->c[$key];
7      }
8  }
9
10 $input = new UserInput();
11 if(is_object($input)) {
12     $c = $input->get('test');
13 }
```

2.3 POI Analysis

- Regular backwards-directed *taint analysis* for unserialize()
- If argument is resolved to user input, report POI vulnerability



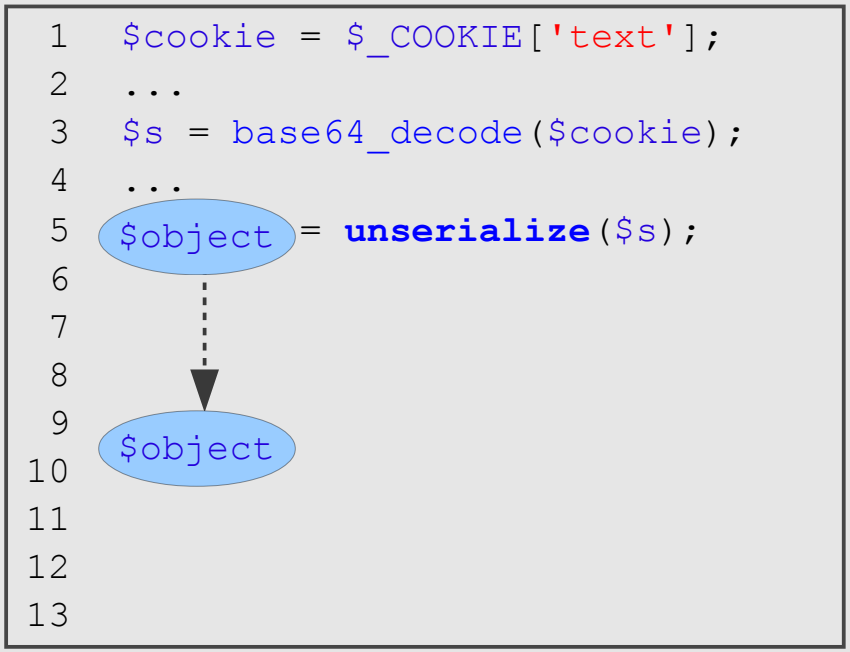
2.3 POI Analysis

- Regular backwards-directed *taint analysis* for unserialize()
- If argument is resolved to user input, report POI vulnerability
- Vulnerable unserialize() call returns *flagged* object

```
1   $cookie = $_COOKIE['text'];
2   ...
3   $s = base64_decode($cookie);
4   ...
5   $object = unserialize($s);
6
7
8
9
10
11
12
13
```

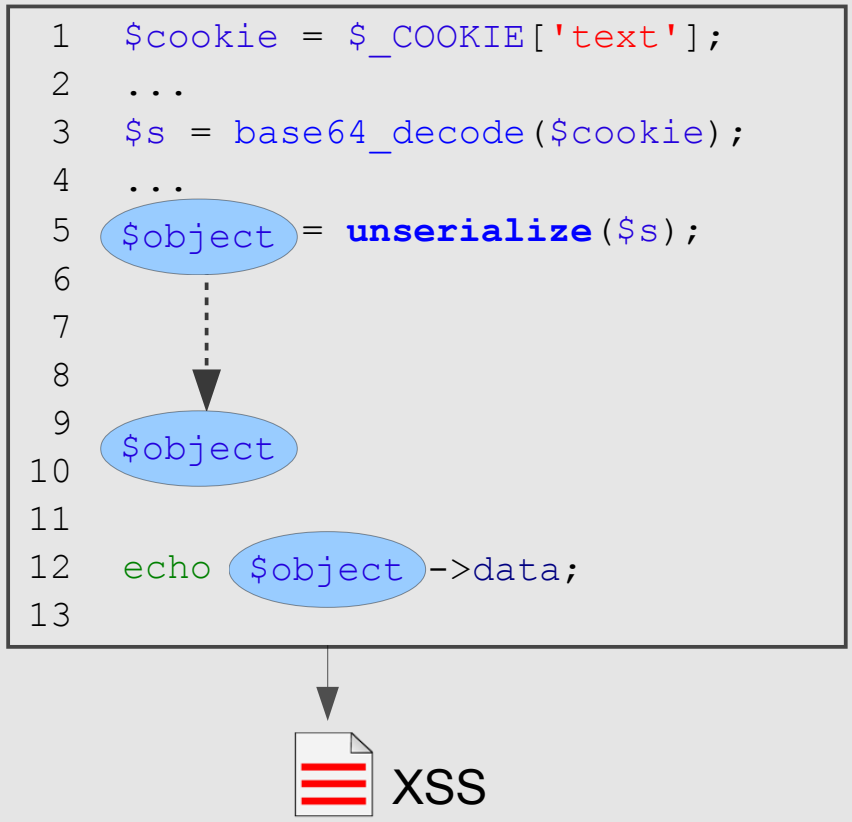
2.3 POI Analysis

- Regular backwards-directed *taint analysis* for unserialize()
- If argument is resolved to user input, report POI vulnerability
- Vulnerable unserialize() call returns *flagged* object
- Propagate *flagged* object forwards-directed



2.3 POI Analysis

- Regular backwards-directed *taint analysis* for unserialize()
- If argument is resolved to user input, report POI vulnerability
- Vulnerable unserialize() call returns *flagged* object
- Propagate *flagged* object forwards-directed
- All properties are tainted



2.4 Inter-procedural Analysis

- If *receiver* can be determined, analyze distinct method

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

```
1  class Process {
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```

```
1  class Database {
2      public function close() {
3          mysql_close($this->link);
4      }
5  }
```

2.4 Inter-procedural Analysis

- If *receiver* can be determined, analyze distinct method
- Otherwise, combine analysis for equally named methods

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

```
1  class Process {
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```

```
1  class Database {
2      public function close() {
3          mysql_close($this->link);
4      }
5  }
```

2.4 Inter-procedural Analysis

- If *receiver* can be determined, analyze distinct method
- Otherwise, combine analysis for equally named methods
- Arguments of a sensitive sink that are resolved to object properties are stored as the method's *sensitive properties*

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

```
1  class Process {                    $this->pid
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```

```
1  class Database {
2      public function close() {
3          mysql_close($this->link);
4      }
5  }
```

2.4 Inter-procedural Analysis

- If *receiver* can be determined, analyze distinct method
- Otherwise, combine analysis for equally named methods
- Arguments of a sensitive sink that are resolved to object properties are stored as the method's *sensitive properties*
- Sensitive properties are applied to each *receiver* at call-site

```
1  class TempFile {
2      public function __destruct() {
3          $this->shutdown();
4      }                          $this->handle->pid
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

```
1  class Process {                          $this->pid
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```

```
1  class Database {
2      public function close() {
3          mysql_close($this->link);
4      }
5  }
```

2.4 Inter-procedural Analysis

- If *receiver* can be determined, analyze distinct method
- Otherwise, combine analysis for equally named methods
- Arguments of a sensitive sink that are resolved to object properties are stored as the method's *sensitive properties*
- Sensitive properties are applied to each *receiver* at call-site

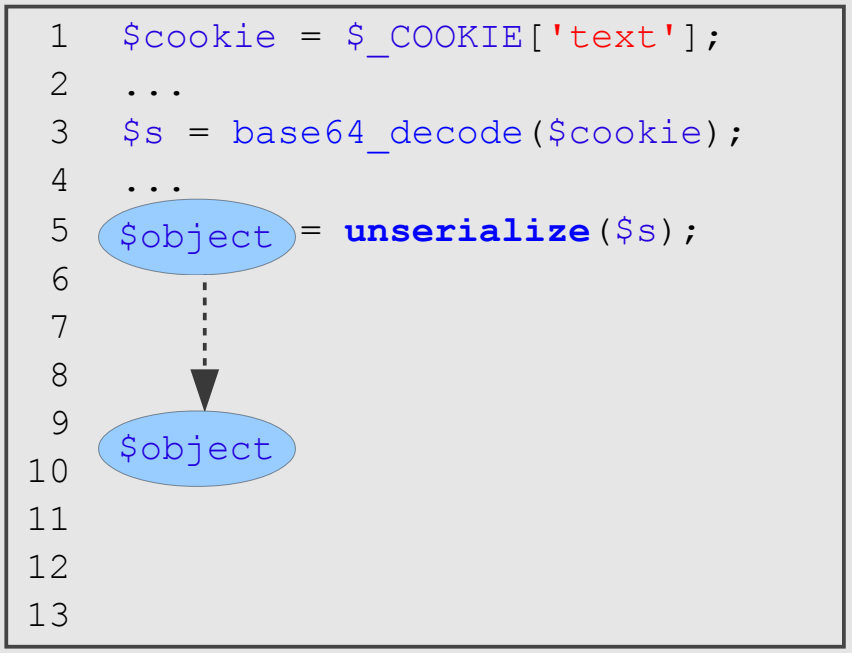
```
1  class TempFile {                    $this->handle->pid
2      public function __destruct() {
3          $this->shutdown();
4      }                               $this->handle->pid
5      public function shutdown() {
6          $this->handle->close();
7      }
8  }
```

```
1  class Process {                     $this->pid
2      public function close() {
3          system('kill ' . $this->pid);
4      }
5  }
```

```
1  class Database {
2      public function close() {
3          mysql_close($this->link);
4      }
5  }
```

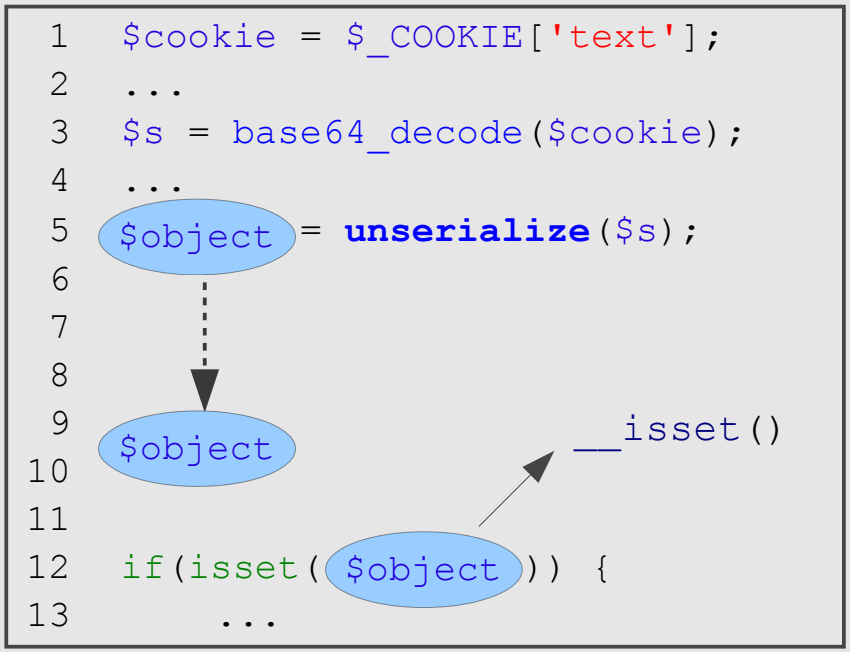
2.5 Magic Method Invocation

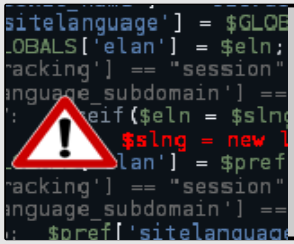
- Automatically invoke analysis for all __destruct() and __wakeup() methods
- *Flagged* object is *receiver*
- Sensitive properties trigger *POP gadget chain* report which is attached to the POI report



2.5 Magic Method Invocation

- Automatically invoke analysis for all __destruct() and __wakeup() methods
- *Flagged* object is *receiver*
- Sensitive properties trigger *POP gadget chain* report which is attached to the POI report
- Trigger other magic methods when propagated *flagged* object is used in related events





3. Evaluation

3.1 Selected CVE

CVE	Software	Version	POI	Chains
CVE-2014-2294	Open Web Analytics	1.5.6	1	0
CVE-2014-1860	Contao CMS	3.2.4	3	3
CVE-2014-0334	CMS Made Simple	1.11.9	1	0
CVE-2013-7034	LiveZilla	5.1.2.0	1	0
CVE-2013-4338	Wordpress	3.5.1	1	0
CVE-2013-3528	Vanilla Forums	2.0.18.5	2	1
CVE-2013-2225	GLPI	0.83.9	1	0
CVE-2013-1465	CubeCart	5.2.0	1	1
CVE-2013-1453	Joomla!	3.0.2	1	2
CVE-2009-4137	Piwik	0.4.5	1	3

3.2 POI Detection

CVE	Software	Version	POI			Chains
CVE-2014-2294	Open Web Analytics	1.5.6		1	-1	0
CVE-2014-1860	Contao CMS	3.2.4	+16	3		3
CVE-2014-0334	CMS Made Simple	1.11.9		1		0
CVE-2013-7034	LiveZilla	5.1.2.0	+1	1		0
CVE-2013-4338	Wordpress	3.5.1		1	-1	0
CVE-2013-3528	Vanilla Forums	2.0.18.5		2		1
CVE-2013-2225	GLPI	0.83.9	+14	1		0
CVE-2013-1465	CubeCart	5.2.0		1		1
CVE-2013-1453	Joomla!	3.0.2	+1	1		2
CVE-2009-4137	Piwik	0.4.5		1		3

3.2 POI Detection

CVE	Software	Version	POI		Chains
CVE-2014-1860	Contao CMS	3.2.4	+16	3	3
CVE-2013-7034	LiveZilla	5.1.2.0	+1	1	0
CVE-2013-2225	GLPI	0.83.9	+14	1	0
CVE-2013-1453	Joomla!	3.0.2	+1	1	2

- New POI vulnerabilities in *Contao*, *LiveZilla*, and *GLPI* are already fixed
- New POI vulnerability in *Joomla!* lead to RCE until version 3.3.4 (**CVE-2014-7228**)[1]

CVE	Software	Version	POI		Chains
CVE-2014-2294	Open Web Analytics	1.5.6	1	-1	0
CVE-2013-4338	Wordpress	3.5.1	1	-1	0

- False negatives due to *reflection* (OWA) or complex *second-order* data flow (Wordpress)
- No false positives

[1] http://websec.wordpress.com/2014/10/05/joomla-3-3-4-akeeba-kickstart-remote-code-execution-cve-2014-7228/

3.3 POP Chain Detection

CVE	Software	Version	IGadgets	Chains		
CVE-2014-2294	Open Web Analytics	1.5.6	24	+9	0	
CVE-2014-1860	Contao CMS	3.2.4	136	+11	3	
CVE-2014-0334	CMS Made Simple	1.11.9	41	+1	0	
CVE-2013-7034	LiveZilla	5.1.2.0	21		0	
CVE-2013-4338	Wordpress	3.5.1	41		0	
CVE-2013-3528	Vanilla Forums	2.0.18.5	14		1	-1
CVE-2013-2225	GLPI	0.83.9	77		0	
CVE-2013-1465	CubeCart	5.2.0	47	+2	1	
CVE-2013-1453	Joomla!	3.0.2	73	+3	2	
CVE-2009-4137	Piwik	0.4.5	111	+2	3	-1

3.3 POP Chain Detection

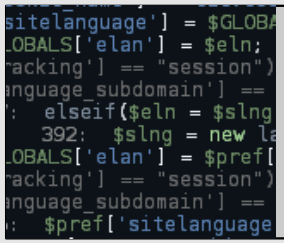
CVE	Software	Version	IGadgets	Chains
CVE-2014-2294	Open Web Analytics	1.5.6	24	+9 0
CVE-2014-0334	CMS Made Simple	1.11.9	41	+1 0
CVE-2013-1465	CubeCart	5.2.0	47	+2 1
CVE-2013-1453	Joomla!	3.0.2	73	+3 2

- New chains **define** severity of known POI, e.g., SQLi in *OWA*, File Delete in *CMSMadeSimple*
- New chains **refine** severity of known POI, e.g., Local File Inclusion in *Joomla!* [1]

CVE	Software	Version	IGadgets	Chains
CVE-2013-3528	Vanilla Forums	2.0.18.5	14	1 -1
CVE-2009-4137	Piwik	0.4.5	111	+2 3 -1

- False negative due to imprecise framework analysis
- 10 false positives due to dynamic class invocation

[1] http://websec.wordpress.com/2014/10/03/joomla-3-0-2-poi-cve-2013-1453-gadget-chains/



4. Conclusion

4. Conclusion

- Code reuse attacks are not only a threat for memory corruption
- We studied the nature of POI and POP in PHP
- We proposed and implemented an automated approach for detection
- We found previously unknown POI vulnerabilities
- We found new POP chains that determine the severity of a POI
- False positives and negatives can occur by imprecise analysis of dynamic OOP features
- Avoid serialize()/unserialize(), use json_encode()/json_decode()

Questions ?

johannes.dahse@hgi.rub.de

Thank you!
Enjoy the conference.