Non-alphanumeric code
With JavaScript & PHP by Gareth Heyes
How did it begin?

- Yosuke Hasegawa posted to the sla.ckers message board
- We were all amazed that this executed code without the need of alphanumeric characters
How does it work?

• JavaScript is a loosely typed language
• true+true == 2
• The key to non-alpha code is using string indexes to obtain individual letters
• ‘abc’[0] == ‘a’
• Objects can be converted to strings using the concatenation “+” operator and the toString value of that object can be used
  • "+{} == ‘[object Object]’
  • We can use each letter of the generated string ‘[object Object][1] == ‘o’
  • But we need 1 right? That is alphanumeric
Getting numbers

- `+[] == 0`
- `[]` array returns a blank string when `valueOf/toString` is called
- `+` (infix) operator converts an object to a number
- Because the value is a blank string the result is 0
- To get the number 1 we can use the not “!” operator
- `[] == false; !![] == true`
- Reverse of false is true then use + infix to convert to one
- `+!![] == 1`
- But that’s just 0 and 1 right?
Getting bigger numbers

- @oxotnick from sla.ckers came up with a cool trick
- Using object accessors you can increment values without a variable
  
  ```javascript
  ++[[]][+[]] == 1
  ```
- Works with an array inside an array:
  
  ```javascript
  [ ];
  ```
- `[ ][+[]]` then accessing the first element of the array `+[[]]` == 0
- Then finally incrementing the value `++[[]][+[]]`
- Notice `++[]` is illegal but via accessor it works
- The `[]` is converted to 0 then incremented by 1
- You can increment further by concatting arrays
  
  ```javascript
  ++[[]][+[]]+[++[[]][+[]]][+[]]
  ```
Getting Objects

- Firefox 2 and older browsers allowed you to remove the object from function call and returned window instead. This is older ES behaviour.
- `(1,[].sort)() == window`
- This works on IE9 still 😊
- `(1,[].reverse)() == window`
- You are removing the “this” value of the array
- `[3,1,2].sort() // works as expected
- We need window to call other functions such as alert with non-alpha code
Generating sort

- We need to generate the string “sort” in order to get our reference to window
- Any ideas?
- false
- [object Object]
- true
- true
- We need the string index for each of the letters
- For false we need the 3rd index starting from 0
- [object Object] requires 1 and so on
Generating sort cont.

• First we generate false with `![]+[]` and convert it to a string
  `![] == false`  
  `+ [] // converts to string using a blank array`
• Getting the letter is trickier then you first think:
  `![]+[][0] == NaN`
• JavaScript is getting the first element of the array instead of concating
• Enclose with another array and access the first element and you get the string “false”
  `[false] [0]`  
  `['false'][0][0]`  
  `['false'][0][0]`  
  `['false'][0][0]`  
  `['false'][0][0] == 'f'`
Generating sort cont.

• We need to increment the value to access “s” as shown earlier
  
  ```javascript
  ++[[[]][+[]]+[+[[]][+[]]][+[]]+[+[[]][+[]]][+[]]+[+[[]][+[]]][+[]]] == 3
  ```

• Using our string false we access the third element which is “s”
  
  ```javascript
  ![[]]+[]][+[]][+[]]+[+[[]][+[]]][+[]]+[+[[]][+[]]][+[]] == ‘s’
  ```

• To generate “o” we only need the number 1 and an object string
  
  ```javascript
  []+{} == [object Object] as string
  ```

• `[[[]][+[]]][+[]]` access first element of array which is the string

• We’ll use a shortcut I mentioned earlier
  
  ```javascript
  [[]+{}][+[]][+!![]]
  ```

• `[[[]][+[]]][+[]][+!![]]` accesses 1st element from 0 of our string “o”
Generating sort cont.

- The same techniques can be applied to get “r” from true
- ![[]+[[]]+[]+[!![]]] == “r”
- And ![[]+[[]]+[]+[[]]] == “t”
- Put it all together and what have you got?
- ![[]+[[]]+[]+[[]]+[]+[++]][+[]][++] == ‘sort’
- Using the trick from before we can use an array to get window
- ()([[[]]+[]]+[]+[++]][++[]][++]|[++][]+[++][]+[++][]+[++][]+[++][]+[++][]+[++][]+[++][]+[++][]+[++]() // kaboom! Window on IE9!
Calling alert(1)

- Making “a”, “l”, “e” follows the same process, we already have “r” and “t”
- ![Expression](expression.png) // “a” 1st position from false
- ![Expression](expression.png) // “l” 2nd position from false
- ![Expression](expression.png) // “e” 4th position from false

Lets combine it all together

- ![Expression](expression.png) // alert(1)
Executing arbitrary JavaScript

- Generating every single character is hard work
- Using Function you can generate a character from a octal escape
- “a” looks like \141 in octal
- Function('return"\141"'); // returns “a”
- We need the letters in “Function”, “return” and generate the required octal number
- The conversion function converts the input into octal
- A reference to window is added, along with the a range of numbers 0-9
- Any code can be executed using this method
The great char wall

• How many characters are required to execute non-alphanumeric code?
  • 6 characters. ()[]!+
• The great char wall was named by me and sirdarckcat as a impossible barrier to break
• We tried many many times
• You need “(" and ")” to call functions “[“ and “]” to access string indexes and access properties and + to concat and convert into a number. “!” is important because you can generate true or false
• Can you break the char wall?
Attempts to break the wall

- Many JavaScript warriors have tried and failed to break the wall
- Without “!” you can’t get true or false with “[]+()”
- Without (“ and “) you can’t get window
- Without “p” and “_” you can’t get properties such as __parent__ on older browsers
- The wall is an impossible problem that cannot be solved without some sort of JavaScript quirk
- I tired and failed to break it again
A non-alpha demo
Decoding non-alpha code

- Properties are unknown and are calculated at run time
- E.g. Obj[x] but we don’t know “x” until the script runs
- You could convert basic patterns such as +[] to zero etc but subtle variations in the code could bypass this
- We don’t know the property and we don’t know the object either e.g. !obj true or false? We don’t know
How can you decode then?

- Sandbox the JavaScript code
- Proxy any calls to native functions and observe the result
- Remember Function?
- If we can change “Function” then we can have the decoded result
- The sandboxed code runs as normal but in a fake environment that we control
Decode demo
How decoding works

- Uses free JavaScript sandbox that I developed called JSReg

```javascript
parser.extendWindow("$sandbox$", function(code) {
var js = JSReg.create(), result;
js.setDebugObjects({doNotFunctionEval: true, functionCode: function(code) {
    code = code.replace("J.F();var $arguments$=J.A(arguments);","'"));
    result = code;
}});
try {
    js.eval(code);
} catch(e) {
    return e;
}
return result;
});
```

How decoding works cont.

- Extends the environment with a custom sandbox function
- Changes JSReg behaviour to only run Function but not execute the code
- Removes any unneeded sandboxing code that remains
- JSReg does the rest 😊
Fooling the decoder

• If you can detect you’re in a sandboxed environment you could alter the code’s behaviour
• You could break the sandbox
• You could find a syntax quirk inside the sandbox to prevent execution
• You could use eval instead of Function
• You could use a browser DOM object
Defending the sandbox

• You could change eval and related Functions behaviour and log the result
• Provide a fake DOM object that an attacker thinks is real
• I challenge awesome security researchers on sla.ckers to break JSReg to prevent sandbox escapes and syntax problems
• Make the environment seem real by overwriting toString/valueOf of every native object/function to return the expected result
Is non-alpha code evil?

- Without testing the boundaries of what is possible we cannot hope to provide adequate defences
- The attacker could figure it out anyway
- Anyone researching malicious non-alpha code will be forced to use tools that can decode the data
- Improves tools
PHP Non-alpha
You can do this in PHP?

- I wanted to emulate this stuff in PHP
- Nobody thought it was possible
- PHP lacks the toString/valueOf properties of JavaScript
- How can you generate characters from nothing?
Thinking about non-alpha PHP

• PHP does a string conversion for arrays and results in “Array”
• Maybe we can use those characters?
• Hmmm what can we call with that? Not much
• How can we generate other characters from Array to enable us to execute code
Bitwise operators on strings

- I figured out that PHP allows bitwise operators on strings
- $A \mid B == C$
- Generating "_" is difficult though
- Using two or more operations can result in different characters. E.g. Generate "C" use "C" to generate "D" etc
Process for gen underscore

- Create array, concat with self to string, for loop to find char to xor, second loop to gen first char with A
- $\text{$_=$;}$ //creates an array
- $\text{$_=$$_$;}$ // converts to string Array
- $\text{$_=$$_$;}$ // converts to zero
- $\text{$_[+$_$++];}$ // gets “A” from the string
- $\text{$_[+$+$+$_]+$_$;}$ // gets “A”
- “A” | (XOR “a” with 0x7f)
- Result: $\text{$_[+$_$++]$_[$+$$_]+$_$;}$^0x7f;
- 0x7f is the literal character
Clever but...

- Stefan Esser (security god of PHP) said nice but why not use ++ or --
- Increment/decrement works on strings!
- $x='a';$x++;echo $x; // "b"
- Generating underscore was fun but it is smarter to use ++ or --
Calling print “hello”

```php
<?php

$S[]==$S; $S=$S.$S; $Z=+$S; $X=$Z; $X++; $X=$X+$X; $O=$O+$X; $S=$S+$O; $T=$S[$X]; $U=$S[$Z];

$O=$O+$X; $R=$S; $R++; $R=$R+$R; $S=$S+$O; $S=$S+$O; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=$R+$O; $S=$S+$O; $O=$O+$X; $O=$O+$X; $R=
```

Generating non-alpha PHP

- Generate letters and numbers required
- Assert == eval in PHP
- Use chr to generate require letters
- Convert every character into their charcode then use chr to generate and assert to call
PHP Demo

```php
<?php

// Sample PHP code

// End of PHP code

```
Thanks

• Yosuke Hasegawa, Stefan Esser, Mario Heiderich, @Sirdarckcat, @Lever_one, @thornmaker, @rvdh, @oxotnick, @SW, @theharmonyguy and all my slackers friends
Questions?