New Methods in Automated XSS Detection & Dynamic Exploit Creation

A Multi-deck Presentation

Kenneth F. Belva, CISSP, CEH
xssWarrior.com
Contents & Deck Content

• Contact Information / Bio
• Slide Deck 1: Methods and Techniques Overview
  – Describes the overall picture of how things work
• Slide Deck 2: OWASP AppSecUSA 2015 Presentation
  – Gives more details about methods and variations
• Slide Deck 3: xssWarrior & XSS: A Basic Introduction
  – Non-Technical Introduction with screenshots of product showing this is not just theory / vaporware
Ken's Contact Information

Email: contact @ xssWarrior.com
Product: http://xssWarrior.com
Twitter: http://twitter.com/xssWarrior
Me: http://twitter.com/infosecmaverick

Research: http://securitymaverick.com
Essays: http://www.bloginfosec.com

Stop by and say, 'Hi'!
Bio of Kenneth F. Belva

Kenneth F. Belva is the Publisher and Editor-in-Chief of bloginfosec.com. He is currently developing xssWarrior, currently the only scanner that can automate testing for Stored XSS, for commercial use at xssWarrior.com. In addition, he is an independent penetration tester and security researcher.

For the past 15 years he worked in Cyber Security mainly in the financial services vertical, most recently at a multinational conglomerate, conducting both technical and non-technical risk assessments at the application and network layers. From 2005 - 2013 he managed an Information Technology Risk Management Program for a bank whose assets are Billions of dollars.

At the OWASP AppSec2013 conference BugCrowd validated three of his 0-day vulnerabilities he found in Yahoo, Yandex and Angelist within the first two days of BugBash2013. He has since been credited with finding a number of other vulnerabilities on sites such as Netflix and OKCupid.

He was previously on the board of the New York Metro Chapter of the Information Systems Security Association (ISSA) where he served in various capacities over the past 9 years. He has spoken and moderated at the United Nations as well as presented on AT&T’s Internet Security News Network (ISNN) on discovering unknown web application vulnerabilities as well as being interviewed on security enablement.

ITsecurity.com recognized him as one of the top information security influencers in 2007.


He recently co-authored a paper entitled “Creating Business Through Virtual Trust: How to Gain and Sustain a Competitive Advantage Using Information Security” with Sam Dekay of The Bank of New York. of security breaches on stock prices.

Mr. Belva frequently presents at information security conferences around the US as well as globally. He writes on day-to-day information security experiences in a non-essay format at SecurityMaverick.com when time permits and can be followed on twitter @infosecmaverick.
Slide Deck 1
Methods and Techniques Overview
New Methods in Automated XSS Detection & Dynamic Exploit Creation

Kenneth F. Belva, CISSP, CEH
xssWarrior.com
Overview of Methods and Techniques
Presented at OWASP AppSecUSA 2015
Points of Interest

- Please note: This presentation is a very simple explanation to communicate the method and concepts
- See OWASP presentation for more in-depth ideas and examples
- Not vapor-ware: Advanced Scanner Exists
- Links on second to last slide for more information
- Please visit: xssWarrior.com
Part 1:
The Current Automated Methodology
Most Popular XSS Detection Methodology:
The Exploit String Includes the Payload/Token

<script>alert(12345)</script>

Scanners Slam Strings into Application Hoping for a Callback or Event to Fire for Validation

Inefficient and Inaccurate
One Major Problem is Transformations

"script"alert12345"/script";

Most Popular XSS Detection Methods Cannot Account for Different Exploit Situations
Part 2:
The New Testing Methodology

Applies to All XSS:
Reflected, Reflected Stored, Stored, DOM
Step 1: Tracing Data and Building Cases: Inputs and Outputs

The goal:
Track where the data goes into the application and where it comes out
We assign a unique slug value to each field and load it into the application

Assign unique slug value to a field and submit

Spider site to see where unique slugs come out in HTML/JS /DOM/etc.
In this way we build cases of input and output
Page 1 ---> Page 2 / Page 3 / Page 4

Example of Slug in HTML Output
<img src="12345">some text</img>
<a href="a">12345</a>

We can inject custom script into DOM and search for our slug
Step 2: Parse source where slug found to get MINIMUMUM characters needed for each context

"\<img src="12345" >some text</img>" is needed for Case 1 Exploit and None Needed for Case 2 Exploit

Case 1:
\<img src="12345">[exploit]</img>

Case 2:
\<img src="[exploit.js]">some text</img>
Step 3:
Use Sandwich Method to Determine Potential Vulnerability and Build Table of Characters that Pass through App/Filter

Sandwich Method:

Enclose string to search between two unique slugs

12345"12345
12345<12345

As these unique strings are searchable we will know if they come out the other side for our cases built in Step 1

http://website?parm=12345"12345
http://website?parm=12345<12345

Potential Vulnerability:
<img src="12345"12345" >some text</img>
<a href="a">12345<12345</a>

Not Vulnerable (in modern browsers):
<img src="12345&quot;12345" >some text</img>
<a href="a">12345&lt;12345</a>
Step 4: If potential vulnerability exists check for exploit characters that fit the context

Case 1 HTML:
<img src="12345"12345" >some text</img>

Exploit 1:
<img src="http://website/EvilJS.js" >some text</img>

Exploit 2:
<img src="EvilJS.js" >some text</img>

Potential Exploits & Special Characters:
http://website/EvilJS.js --> ./
EvilJS.js --> .

Case 2 HTML:
<a href="a">12345<12345</a>

Exploit 1:
<script>alert(10)</script> --> <>()/

Exploit 2:
<script>String.fromCharCode(88,83,83)</script> --> <>()/.,
Step 5:
From Built Table We Can Further Determine Exploit Selection: Which Should Work & Which Should Fail Based on Which Characters Make it Through Filter (Accurately Determine Transformations)

<table>
<thead>
<tr>
<th>Translation Name</th>
<th>Value-Originally</th>
<th>Value-Submit</th>
<th>Value-Detect</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>HTML</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&amp;x3c;</td>
</tr>
<tr>
<td>HTML-NoSemi</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&amp;x22</td>
</tr>
<tr>
<td>HTML-pre</td>
<td>&lt;</td>
<td>&amp;x3c;</td>
<td>&amp;x3c;</td>
</tr>
<tr>
<td>HTML-pre</td>
<td>&quot;</td>
<td>&amp;x22;</td>
<td>&amp;x22;</td>
</tr>
</tbody>
</table>

Value-Submit = Value Submitted to Application
12345&lt;12345
12345&lt;12345
12345&amp;x22;12345

Value-Detect = Value Searched in HTML/JS/DOM by Scanner
12345&lt;12345
12345&amp;x3c;12345
12345&amp;x22;12345

When Submitted The Character Should be tested with and without URL encoding since older browser do not encode before submission
12345%2212345 → 12345"12345
Step 6: Build Exploit with Proper Syntax and Test
(A Simple Example)

Assume Proper Characters Passed Filter and in our Table
HTML Case:  <img src="12345" >some text</img>

Syntax from parsing:  " >
Exploit:  <script>alert(1)</script>
Dynamic Exploit:  "><script>alert(1)</script>

Test / Submit & Scan:  12345"<script>alert(1)</script>12345

Result 1 (Valid):  <img src="12345" ><script>alert(1)</script>12345"some text</img>

An Invalid Might look Like:  <img src="12345" >>alert1</s>12345"some text</img>

12345">>alert1</s>12345 Does not Match 12345"<script>alert(1)</script>12345

Since we can parse the HTML/JavaScript/DOM (syntax) and know what gets through the filter
we can build complex dynamic XSS exploits
Additional Notes

All Other String Combinations are Searchable. For Example, Anti-XSS Libraries:

12345<script>12345
12345<script>12345
Part 3: Additional Automated XSS Exploit Techniques
Item 1: New Exploit Validation Method without Callbacks or Event Trigger

If data is assigned a variable by definition the code has executed

Assume our exploit is:

```html
<script> sploitValidationField = 12345 </script>
```

If we search for `sploitValidationField` in the DOM and find the value in it is 12345

We will know our exploit will work

(Call backs and event triggers are still valid too)
Item 2: Privilege Escalation Testing

Build Case in following way:

Authenticate and Load Slugs as User of one Level (Input)

Authenticate as Higher Level user and Scan for Slugs (Output)

Once Mapped from Lower to Higher User Test using Above Methods
Closing Remarks & Links

- Support Our Cyber Security Industry Independent Researchers:
  - Please License: Don't Steal
- Currently Available as API and Service Offering
  - http://xssWarrior.com
- LinkedIn Application Business Page
  - https://www.linkedin.com/company/xsswarrior
- Contact information for Engagements and Speaking
  - speak@xssWarrior.com
- Linkedin Profile
  - https://www.linkedin.com/in/kenbelva
- xssWarrior YouTube Video:
  - https://youtu.be/CxHvr9Et3lo
- OWASP AppSecUSA 2015
  - https://appsecusa2015.sched.org/event/b3bf7e553d06f523704697068f0adedc
  - https://www.youtube.com/playlist?list=PLpr-xdpM8wG93dG_L9QKs0W1cD-esQEu
Thank You Much For Your Time
Slide Deck 2
OWASP AppSecUSA 2015 Presentation
Table of Contents

- Introduction / Background
  - What this presentation is and what it is not....
  - Some History: Discovering the Dynamic XSS Methodologies

Part 1: The State of Automated XSS Discovery Today

- On Payloads: Static / Signature Analysis
- Current Known & Popular Automated XSS Testing Methods
- Issues with Payloads: Syntax and Transformations

- The Payload “Slam”
- The Tracing Payload
- The Trace and then Payload Replace

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
Part 2: New Methods -
Dynamic Analysis of XSS
Vulnerabilities: The Theory & (Some) Practice

- Change of Focus from Payloads to Characters
- Application Component Review: Filters, DB, Memory, Source & DOM
- Let's briefly talk about slugs and fields
- Tracing and parsing for needed characters
- The Sandwich Method
- The New XSS Detection Logic
- A Quick Reflected XSS Example
- Sandwich Method Extended: Brute-Force, Special Strings, Various Encodings & more
- Filtering in the field: A Real-life Pen Test Example
- The Questions of Accuracy and Efficiency
- Browser Considerations
- Goodbye Payloads! XSS is now about Characters, Slugs, Parsing & Filtering

http://xssWarrior.com
All Material and Methods Contained Here Patent Pending. All Rights Reserved
Table of Contents (pt3)

- Part 2: New Methods
- Dynamic Analysis of XSS Vulnerabilities: The Practice

- Spidering for Slugs and XSS
- HTTP Methods: GET / POST / HEADER / COOKIES
- Another Simple Reflected XSS Example
- A Simple Stored XSS Detection Example
- A brief word on DOM-Based XSS
### Table of Contents (pt4)

- Part 3: New Methods - Dynamic XSS Exploitation

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue with Current Static XSS Exploit Payloads</td>
</tr>
<tr>
<td>Introducing Dynamic XSS Exploit Analysis and Generation</td>
</tr>
<tr>
<td>Brief Review: Change of Focus from Payloads to Characters</td>
</tr>
<tr>
<td>Finding our trace or slug value in the source</td>
</tr>
<tr>
<td>Getting the HTML Syntax</td>
</tr>
<tr>
<td>Writing the Dynamic Exploit</td>
</tr>
<tr>
<td>Additional Validation Methods: Callbacks, etc.</td>
</tr>
<tr>
<td>Q&amp;A</td>
</tr>
</tbody>
</table>

[http://xssWarrior.com](http://xssWarrior.com)  

All Material and Methods Contained Here Patent Pending. All Rights Reserved
Introduction / Background
<table>
<thead>
<tr>
<th>What this presentation is &amp; what it is not…</th>
</tr>
</thead>
</table>

This presentation is a starter introduction to a new way of doing Dynamic XSS vulnerability detection.

This presentation shows SIMPLE examples in order to communicate the UNDERLYING CONCEPTS of Dynamic XSS Discovery.

It does NOT cover every iteration of the methods described.

- I briefly cover DOM-based XSS in this presentation but the methods described here can be extended for this as well – I will cover some of these verbally.

It does NOT cover more complex ideas and XSS cases but it should be understood from the presentation how these may be pragmatically solved and implemented.

The presentation covers straight HTML / JavaScript but it should also be understood that the methods contained herein also apply to additional technologies such as Flash and ActiveX.

It is NOT a product pitch.

xssWarrior: The methodology presented herein is not theory. A real application exists that embodies this presentation and it is continuing to be enhanced to add more and more functionality described here.

[http://xssWarrior.com](http://xssWarrior.com)
Some History

Discovering the Dynamic XSS Methodologies

I used major and open source scanners in large scale environments (2013-2014)

When the current automated scanners finished processing I would review the sites manually and I could almost always find additional XSS vulnerabilities not found by the scanners

I noticed that the exploits returned back from these scanners did not always function properly; namely, I needed to correct the syntax to get them to execute

Valentines Day 2014 - Yahoo! offers a doubles bounty for sports.yahoo.com. Found XSS Across 17 domains and every page on those domains. Why didn't their scanner(s) catch it?

This lead me to create an improved automated XSS vulnerability scanning detection system that can find the types of vulnerabilities I was finding manually before

I wrote a quick prototype scanner and found a bunch of XSS in bounties using the method I developed

I subsequently turned prototype into a full fledged scanner xssWarrior which included expanding my original method to include Stored XSS & DOM-based XSS

All material contained within is patent pending

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
Part 1:
- The State of Automated XSS Discovery Today
Almost all automated scanners today use a payload methodology. These strings consist of:

- Sample exploits
- Syntax
- Sometimes these strings contain an identifier or tracer value
- Sometimes callback/debugging payloads

Problem: The big issue is that one needs a high volume of use cases to satisfy every single variation.
- Satisfying all variations is not possible
- Cannot handle complex or unique XSS issues

Problem: If it doesn't fit something predefined it isn't found.
- This is the XSS equivalent to antivirus signatures

Let's turn to a few Open Source examples:

Please note I am a big fan of OWASP and their projects.

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
OWASP Xenotix XSS Payloads

OWASP Xenotix XSS Exploit Framework

OWASP Xenotix XSS Exploit Framework is an advanced Cross Site Scripting (XSS) vulnerability detection and exploitation framework. Xenotix provides Zero False Positive XSS Detection by performing the Scan within the browser engines where in real world, payloads get reflected. Xenotix Scanner Module is incorporated with 3 intelligent fuzzers to reduce the scan time and produce better results. If you really don't like the tool logic, then leverage the power of Xenotix API to make the tool work like you wanted it to be. It is claimed to have the world's 2nd largest XSS Payloads of about 4800+ distinctive XSS Payloads. It is incorporated with a feature rich Information Gathering module for target Reconnaissance. The Exploit Framework includes real world offensive XSS exploitation modules for Penetration Testing and Proof of Concept creation. Say no to alert pop-ups in PoC. Pen testers can now create appealing Proof of Concepts within few clicks.

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
Sample Xenotix Payload Variations

```html
<FRAMESET>
  <FRAME SRC="javascript:alert(1);"></FRAMESET>

');alert(1);
');alert(1);
";alert("KCF");"
";alert(String.fromCharCode(75,67,70));"
';alert("KCF");'
';alert(String.fromCharCode(75,67,70));'
";alert("KCF")
";alert(String.fromCharCode(75,67,70))
';alert("KCF")
';alert(String.fromCharCode(75,67,70))
<script>var var = 1; alert(var)</script>
<script type=text/javascript>alert(1)</script>
```

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
Other Tool Payloads
(XSSer)

## XXSer.py @ fuzzing vectors @ psy

```python
# This file contains different XSS fuzzing vectors to inject in payloads and browser supports.
# If you have some new vectors, please email me to [root@lordepsylon.net - epsylon@riseup.net] and will be added to XSSer framework.
# That's all.
##
## Happy Cross Hacking! ;)
```

```python
vectors = [
    {'payload': 'PAYLOAD''
    , 'browser': '''IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [09.02]'''
    , 'payload': ''''<SCRIPT>alert('PAYLOAD')</SCRIPT>'''
    , 'browser': '''IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [09.02]'''
    , 'payload': '''"<PAYLOAD>"'''
    , 'browser': '''IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [09.02]'''
    , 'payload': '''"<TITLE>PAYLOAD"'''
    , 'browser': '''IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [09.02]'''
    , 'payload': '''"<img src="x:x" onerror="PAYLOAD">"'''
    , 'browser': '''IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [09.02]'''
    , 'payload': '''"<BODY onload="#&()...;@\|\"="PAYLOAD>"'''
    , 'browser': '''IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [09.02]'''
    , 'payload': '''"<PAYLOAD>"'''
    , 'browser': '''IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [09.02]'''
    , 'payload': '''"<IMG SRC="PAYLOAD">"'''
    ]
```

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
Other Tool Payloads
(W3af)

```python
return l.get_id()

def get_string_payloads(self):
    """Give the list of payloads.
    Taken from: http://ha.ckers.org/xss.html"
    
    return [
        """;--""""<XSS>=&{()));//\\\';alert(String.fromCharCode(88,83,83));
        """";alert(String.fromCharCode(88,83,83));
        """";alert(String.fromCharCode(88,83,83));
        """";alert(String.fromCharCode(88,83,83));
    ]
```

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
Automated XSS Testing Methods

When searching out “in the wild” for XSS detection, all methods found used payloads to some degree.

Generally speaking there are only three distinct methods:
• The rest appear to be a variation of the three
• Some combine different elements of the three
• This would include added predefined / static characters strings into the front for syntax
• Clearly some of these methods will yield better results

When we examine the methods we will look at:
• The underlying ideas behind the method
• The logic
• The elements / components of the payload
• How it all fits together in order to test for XSS

http://xssWarrior.com
The Payload “Slam”

- Underlying Idea:
  - Assign the variable's data value with a known payload without anything else to it. Notice: no trace value

- The logic:
The Tracing Payload

Underlying Ideas:

- Put a tracer value in a known payload so the payload can be tracked
- If we determine the payload executes, we know which one did

The Logic:

- http://vulnsite.com?param=DATAVALE
  (signature)

- The trace value, such as 12345, is embedded in the predefined payload. Example:
  - http://vulnsite.com?param=<script>alert(12345)</script>
The Trace and then Payload Replace

Underlying Idea:
- Similar to the “Slam” but puts a tracevalue into the logic
- Assign the variable’s data value with a tracevalue to see if user supplied data return to the application
- If the tracevalue is returned, assigned a payload and determine if vulnerable

The Logic:
Issue with Payloads

✓ Syntax and Transformations

Often times a filter will:
- Eliminate anything to right of the “bad” character
- Reject the entire string if it contains a “bad” character

Complex Script Tag Syntax
- It needs to fit the exact payload syntax

We need to be able to account for when data is transformed: example, from %27 to ' or \x27 to '

Payloads will often fail because they cannot account for filtering variations

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
Part 2: New Methods

Dynamic Analysis of XSS Vulnerabilities:
  The Theory & Practice
The Change of Focus

from Payloads to Characters

The idea is that instead of using payloads we test each situation individually based on it's specific circumstances

- We do this by figuring out which characters need to be tested in any given situation (context and syntax)

The move from Payloads to Characters gives some distinct advantages

- We can figure out how the application interprets characters that are passed to it and, should there be filtering, figure out the rules of the filter.
- We can narrow our requirements to exactly what the situation calls for and test only for those characters needed (derived from the context and syntax)
- We can account for more complexity when the application does not fit a per-defined set of assumptions: we can figure out the unique combination of characters and the correct syntax to define proper HTML/JavaScript/JSON/XML/etc. For example, a complex script tag.
- It allows for more fine grained testing

This process may be used in an automated system

With the characters and syntax information can dynamically discover XSS vulns, especially complex ones

With the character and syntax information can write custom exploits too

The key points:

- if we know what characters are needed for correct syntax and we know which characters get through the filter (and how to get them through) there is an extremely high probability there is an vulnerability and in some cases we can know it 100%
- With this information we can then turn to validation of the vulnerability and test different ways (browser / character encodings / specific strings / etc.) it may come about as well as write specific tests for the XSS issue found

http://xssWarrior.com
Let’s briefly talk about slugs and fields

- Assignment, Tracing, Tracking & Syntax Parsing

Our goal will be to track these slugs, especially for Stored XSS. We need to know where the slug enters and exists in order to test for which characters get through the application.

By keeping track of where the slugs are inputted and figuring out where they are outputted (context), we can then parse the HTML for syntax.

Ideally one unique slug per field.

We can even get fancy and use a unique slug per load variation per field.

We can use this data (input, output, context, syntax) to create test cases for our characters.

http://xssWarrior.com
Application Component

Review: Filters, DB, Memory, Source & DOM

- In the wild we find various “application” filters:
  - WAFs
  - Filters may be at the server, application level and/or DB level

- Our slugs will wind up either in the “HTML” source [Reflected XSS], or

- They could remain in the memory of the DOM [DOM-based XSS], or

- They could also be stored temporarily in the memory of the application and exit elsewhere in the app (on a different page or process) [In-Memory XSS], or

- They could become stored in the database (and come out on different pages) [Stored XSS]
The Sandwich Method

- Remember:
  - can advantageously be automated
  - tests any and every character and string combinations!
- Instead of using a single slug (such as 123456789), we use two in concert with one another
- Between the two trace slugs we can then place any additional character or string creating a new unique string
- Examples (no spaces normally):
  - 123456789 A 123456789
  - 123456789 " 123456789
  - 123456789 <script> 123456789
  - 123456789 & #39; 123456789
  - 123456789 %27 123456789
  - Etc / etc / etc....
- If we detect the unique string in the output of the application we know our character or string has made it through the application. For example, we test a URL encoded character:
  - We submit to app string A: 123456789 %27 123456789
  - We search output for string B: 123456789 ' 123456789
  - We know if we find string B in the output we know the ` has made it through the application

http://xssWarrior.com
The New Automated Dynamic XSS Detection Logic

Underlying Ideas:
- The goal is to determine the characters needed to complete the syntax needed for XSS.
- We can then determine if the characters and strings needed for XSS make it through the application.
- We can create variations based on specific scenarios and get accurate testing results instead of firing "blind".
- We can create encoding variations for different characters and determine if the output would be vulnerable when interpreted by specific browser versions.

The Logic:
- http://vulnsite.com?param=WEBSITEVALUE
  - <-- If tracervalue is returned somewhere in the application or found in the DOM we have a potential vulnerability.
  - <-- Parse for syntax & determine HTML/script/etc. characters needed.
  - <-- Parse for other elements such as tags to generate XSS exploits specific for that specific scenario.
  - <-- Now we can test for special characters to see what gets through the filter.
  - <-- There can be a lot of variations on characters/strings that get tested/passed (character encodings, known strings, etc.).
- http://vulnsite.com?param=tracervalue<payload (custom)>tracervalue
  - <-- Payloads get created based on results of character and string testing.
  - <-- Possible but not always needed.
- http://vulnsite.com?param=payload (custom)
  - <-- Final result.

(Note we are now using custom values instead of payload signatures.)
Sandwich Method
Extended

- Brute-Force, Special Strings, Various Encodings & more

(In reality: no spaces in the examples below)

- 6ea261c8 `<script` 6ea261c8
- 6ea261c8 `<script>` 6ea261c8
- 6ea261c8 `%3c` 6ea261c8 (URL encode >)
- 6ea261c8 `&`x39; 6ea261c8 (Decimal: '
- 6ea261c8 `&`x27; 6ea261c8 (HTML Hex: '
- 6ea261c8 `\u0027` 6ea261c8 (Unicode: '
- 6ea261c8 `\x27` 6ea261c8 (Straight Hex: '')
Filtering in the field

A Real-life Pen Test Example

- Case 1:
  - `<` did not work
  - `%3c` did not work
  - `%%3c` WORKED

- Case 2:
  - `javascript` did not work (it was filtered)
  - `'` did not work (it was filtered)
  - `java'script` did work: turned into → `javascript`

- And we can test for these cases because we are testing for characters and strings without using payloads!
The Questions of Accuracy and Efficiency

For most fields we only need to check the characters that make up the syntax (and any encoding variations we choose to run)

Therefore: we check fewer characters than the payload method which usually checks all payloads for a parameter

- This is especially true if we determine that one of the essential characters needed for the syntax fails: we don't need to continue checking the additional characters. Example: a double quote needed in an HTML attribute

If we like we can add additional characters we plan to use in our exploit to determine which exploit to use or how we need to build it (based on the context / syntax analysis). Examples:

- If we use String.fromCharCode we may want to add , ()
- Or if we decide to use data:text/html;base64 in an href we may need to add ;/

Extremely accurate

- If the strings don't match we know character didn't make it through
- If we don't find that the essential syntax characters, strings and / or our exploit characters pass we know it will not be vulnerable

We can analyze more complex issues
Browser Considerations

Once we know the characters that pass through the application, we can build strings that are browser specific if we know that &#; will make it through but something like < will not.

We can get strings through that would be interpreted differently on different browsers.

This means we can test for XSS per browser and not just generic, perhaps IE8 is vulnerable but not IE10 or Firefox 35, etc.

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved.
Key Takeaways!

- Figure out how the application works via character determination is more advantageous than “blindly” submitting payload strings
- We can figure how the application behaves by using the sandwich method to trace character and string data to figure out how the application will behave: filter and / or transform data
- Using the character & syntax data is more accurate and efficient
- We can use the character & syntax data to determine if a vulnerability or potentially vulnerability exists and then create custom exploits especially when the syntax is complex.
- We can use the sandwich method to test for characters and strings in other circumstances even if we cannot parse the source: Flash, ActiveX, etc.
Part 2: New Methods

Dynamic Analysis of XSS Vulnerabilities:
The Practice
In reality any number of methods can be used to get URLs (especially for “AJAX URLs”)—for ease of discussion we will stick with spidering.

Whatever method is used, when spidering the application the components search for slugs:

- If they are immediately found after the page submission we have a Reflected XSS
- If they are submitted but found on another page (in the same session) we have InMemory XSS
- If they are found after the session is cleared and a new one is formed we have Stored XSS
- We find our slugs referenced in the immediate page in the client memory (DOM-based)

If these slugs are found, they are recorded and associated with the location they were inputted.

The goal is to find places to input but also find where slugs are outputted.

We map the input to output of the slugs: this may be a 1 to Many relationship, especially when dealing with Stored XSS (think a name field).

Once we have the input and then the output we can test which characters go in and come out using the Sandwich Method.

We can then track the results and the one’s that have vulnerabilities based on characters and syntax we can be generating exploits.
Testing Application Methods & Synataxes

We can use the sandwich and detection methods described above to test different methods and parts of the application
- GET / POST / HEADER / COOKIES

The application can also test for different syntax formats and test those
- JSON / HTML / XML / Etc.

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
A brief word on DOM-Based XSS

We can search through the DOM for the slug

We can then search through the DOM for the slug sandwich and determine the characters can be represented / not filtered or transformed

We can determine what strings / exploits can be represented in the DOM

We can then use various validation methods – such as callbacks, debug, etc. – to test exploits
Part 3: New Methods

A Brief Method for Dynamic XSS Exploitation
Issues with Current Static XSS Exploit Payloads

- The issue is that the payload is the exploit
  - It is not customized for the context / syntax
  - It could transform due to a filter but there still may be a vulnerability

- Introducing Dynamic XSS Exploit Analysis and Generation

- By knowing the characters and the context a customized exploit may be developed for specific situation, including accounting for transformations of characters through the filter
  - (see pen testing example earlier %%%3c)
Recall our testing logic:

- http://vulnsite.com?param=WEBSITEVALUE
  - If tracervalue is returned somewhere in the application or found in the DOM we have a potential vulnerability
  - Parse for syntax & determine HTML/script/etc. characters needed
  - Parse for other elements such as tags to generate XSS exploits specific for that specific scenario
  - Now we can test for special characters to see what gets through the filter
  - There can be a lot of variations on characters/strings that get tested/passed (character encodings, known strings, etc.)
- http://vulnsite.com?param=tracervalue<payload (custom)>tracervalue
  - Payload based on results of testing
  - Possible but not always needed
- http://vulnsite.com?param=payload (custom)
  - Final result
Method to Determine and Create Custom XSS Exploit (pt2)

- A Simple Dynamic Custom XSS Exploit Method

- Step 1: Find Slug in HTML
- Step 2: Parse HTML to determine where CheckSum exists / syntax check
- Step 3: Determine characters needed to pass through filter based on HTML Syntax
- Step 4: Use XSS Test Method to determine characters that pass through filter
- Step 5: If characters pass through filter, build exploit string based on characters and context and then check if exploit string passes through filter
- Step 6: (optional) Exploit string can be out of band callback for extra validation
- Step 7: Remove MD5 Check Sum and Save Exploit

- Based on the characters and syntax needed, we may decide to add special characters to test which we most likely would use in the exploit we plan to use
  - We can technically make this determination either after we test the preliminary characters
    - That is to say, after we determine if the necessary characters get through via step Step 3
  - Or, we can "guess" and add them to Step 3 and test everything "at once"

http://xssWarrior.com

All Material and Methods Contained Here Patent Pending. All Rights Reserved
We can make it more complex depending on the different exploits for the context: – html tag / text or attribute / script / etc.

- For instance, in the body tag:
  - If we can pass `="() we might be able to exploit`
  - `onload="exploit()"
  - Where we might not be able to pass `"<>()`
  - `"><script>alert(10)</script>"`

- We can account for the transformation and / or filtering mechanisms in place in the application
  - `< will not make it through but `%%` does
Ken's Contact Information

Email: contact@xssWarrior.com
Product: http://xssWarrior.com
Twitter: http://twitter.com/xssWarrior
Me: http://twitter.com/infosecmaverick

Research: http://securitymaverick.com
Essays: http://www.bloginfosec.com

Stop by and say, 'Hi'!
Kenneth F. Belva is the Publisher and Editor-in-Chief of bloginfosec.com. He is currently develops xssWarrior, currently the only scanner that can automate testing for Stored XSS, for commercial use at xssWarrior.com. In addition, he is an independent penetration tester and security researcher.

For the past 15 years he worked in Cyber Security mainly in the financial services vertical, most recently at a multinational conglomerate, conducting both technical and non-technical risk assessments at the application and network layers. From 2005 - 2013 he managed an Information Technology Risk Management Program for a bank whose assets are Billions of dollars.

At the OWASP AppSec2013 conference BugCrowd validated three of his 0-day vulnerabilities he found in Yahoo, Yandex and AngelList within the first two days of BugBash2013. He has since been credited with finding a number of other vulnerabilities on sites such as Netflix and OKCupid.

He was previously on the board of the New York Metro Chapter of the Information Systems Security Association (ISSA) where he served in various capacities over the past 9 years. He has spoken and moderated at the United Nations as well as presented on AT&T’s Internet Security News Network (ISNN) on discovering unknown web application vulnerabilities as well as being interviewed on security enablement.

ITsecurity.com recognized him as one of the top information security influencers in 2007.


He recently co-authored a paper entitled “Creating Business Through Virtual Trust: How to Gain and Sustain a Competitive Advantage Using Information Security” with Sam Dekay of The Bank of New York. Mr. Belva frequently presents at information security conferences around the US as well as globally. He writes on day-to-day information security experiences in a non-essay format at SecurityMaverick.com when time permits and can be followed on twitter @infosecmaverick

http://xssWarrior.com
Slide Deck 3

xssWarrior & XSS: A Basic Introduction
xssWarrior & XSS: A Basic Introduction

Kenneth F. Belva, CISSP, CEH
xssWarrior & XSS

- Presented at One of World's Top Cyber Sec Conferences
- What are some of the consequences of XSS?
- How is it different? What are some benefits?
- Some Public Results
- Graphical Interfaces
- Conclusion
- Who Am I?
- Contact Information
What are some of the consequences of XSS?

- Log in as another person (session stealing)
- Install malware such as APTs (Advanced Persistent Threats) on the user visiting the compromised website
- Redirect users to a fake / malicious website under attacker's control
How is it different?
What are some benefits?

- xssWarrior uses a proprietary method to test and detect for XSS vulnerabilities
- Finds difficult XSS vulnerabilities in complex code
- The scanner excels at a notorious difficult XSS issue: Stored XSS
  - Up to now most scanners cannot test for this accurately due to the limitations of the current techniques

The Benefits:
- With the new automated process, the application lowers the total cost to find XSS vulnerabilities
- Tool easily fits into existing automated scanning processes and procedures
Some Public Results

- Patent-Pending Technique used to find XSS vulnerabilities on following Bug Bounty programs
  - Netflix
  - Yahoo
  - OKCupid
  - Yandex

- xssWarrior found XSS in below applications resulting in CVEs
  - CVE-2014-6635 – Exponent CMS
  - CVE-2014-6618 – Your online shop
  - CVE-2014-6619 – Pizza Inn
  - [To be assigned] – TomatoCart
  - CVE-2015-2043 – MyConnection Server 8.2b
FEATURES
XSS Warrior is perfect for novices and experts alike
API and Service Offerings Available Now
SaaS Service Arriving Soon: Follow us on Twitter

UNIQUE PROPRIETARY METHODOLOGIES
The methods used in XSS Warrior were presented at the world’s top web application cyber security conference OWASP AppSecUSA 2015. The outcome was that the XSS Warrior methods are faster and more accurate than the current methods deployed in the major commercial scanners. XSS Warrior uses a series of unique proprietary methodologies to find difficult XSS in an automated fashion.

All methods are patent-pending.

INTERESTING PRODUCT FEATURES
Some of our product features & methods:
1. Test for XSS privilege escalation attacks
2. Dynamically built JavaScript payloads customized to exploit unique vunls as well as standard situations
3. Automated URL Filter Tests for Character Set Types for browser exploit translations
4. Algorithmic Parameter Manipulation to Trigger Unique XSS Cases
5. Reports scenario specific dangerous characters that bypass filter for further research
6. Extremely Accurate Stored XSS Scanning Method

APPLICATION PROGRAMMING INTERFACE (API)
Our remote SaaS API allows for the XSS Warrior analytical engine to be integrated into 3rd party products for scanning Internet facing hosts. Our engine will report it’s status in real-time with a heartbeat. The vulnerability results may be received in real time or as a final result. The results report protocol is in XML and is easily parsed.

Please contact us in regards to ordering and implementation API requests.

SAAS EASE OF USE
Our intuitive SaaS interface allows even non-technical people to create XSS Warrior scans. The scan results are reported in an easy to read layout which may be directly printed/exported (pdf) for 3rd parties or exported in various formats (XML/CSV/TXT) for use in other applications.
Graphical Interface

- Default URL: http://test1.com
- Individual URLs: Enter domain urls
- Scanning Mode: Spider, Single
- Browser Request Types: GET, POST
- Checks To Run: PARAMETERS, HEADERS, COOKIES, DOM
- User Agent String: Enter user agent string
- Report Options: Vulnerabilities only, All Tests
- Real Time: True, False
- Depth Accuracy: Normal, High
- Request Throttling: None, Limit
- Request Throttle Timing: Throttle Upper Limit
- Authentication Checks: No Authentication, Authentication
- Run Privilege Attack Checks: No Privilege, Privilege
Results Part 1
Results Part 2

```
encoded_EXPLOIT_TESTS_number_6_found: {'value-name': 'ascii', 'value-orig': '<script src="/tiny.com">', 'value-from': '<script src="/tiny.com">'}
encoded_EXPLOIT_TESTS_number_5_found: {'value-name': 'ascii', 'value-orig': '<script>alert(String.fromCharCode(65, 65, 65))</script>', 'value-from': '<script>alert(String.fromCharCode(65, 65, 65))</script>'}
encoded_EXPLOIT_TESTS_number_4_found: {'value-name': 'ascii', 'value-orig': '<img src="/a" onerror="javascript:alert(10)">', 'value-from': '<img src="/a" onerror="javascript:alert(10)">'}
encoded_EXPLOIT_TESTS_number_3_found: {'value-name': 'ascii', 'value-orig': '<script>alert("xssWarrior")</script>', 'value-from': '<script>alert("xssWarrior")</script>'}
encoded_EXPLOIT_TESTS_number_2_found: {'value-name': 'ascii', 'value-orig': '<script>alert("xssWarrior")</script>', 'value-from': '<script>alert("xssWarrior")</script>'}
encoded_EXPLOIT_TESTS_number_1_found: {'value-name': 'ascii', 'value-orig': '<script>alert("xssWarrior")</script>', 'value-from': '<script>alert("xssWarrior")</script>'}
```

End of Vulnerability Results for URL ...
Conclusion

- Use xssWarrior to find common and hard to find XSS vulnerabilities in web properties
- Protect the infrastructure by finding security holes before bad guys do (defense)
- Find XSS holes in adversaries websites before they do (offense)
Who Am I?

- I am almost 20 year veteran in the cyber security field
- Had technical and managerial roles in the cyber space: currently developing xssWarrior for public release
- Active in NYC cyber scene: prior 8+ year board member of NYC chapter of ISSA
- Presented at NYC chapters of OWASP, ISSA, ISC2 and ASIS