XSS-GUARD : Precise Dynamic Prevention of Cross Site Scripting (XSS) Attacks

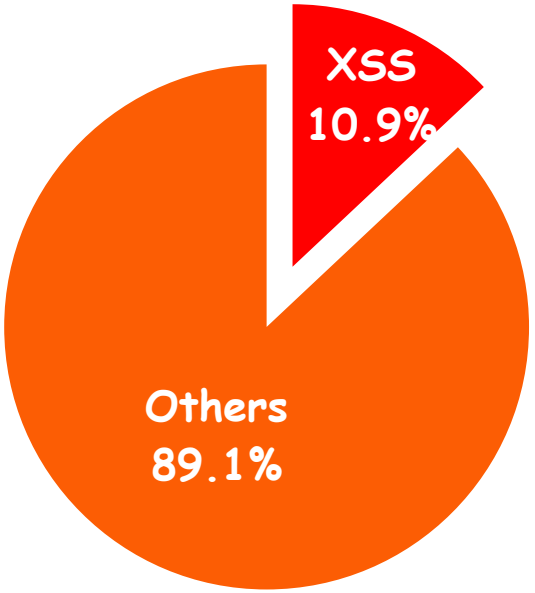
Prithvi Bisht (http://cs.uic.edu/~pbisht)
Joint work with : V.N. Venkatakrishnan

Systems and Internet Security Laboratory
Department of Computer Science
University of Illinois, Chicago
USA

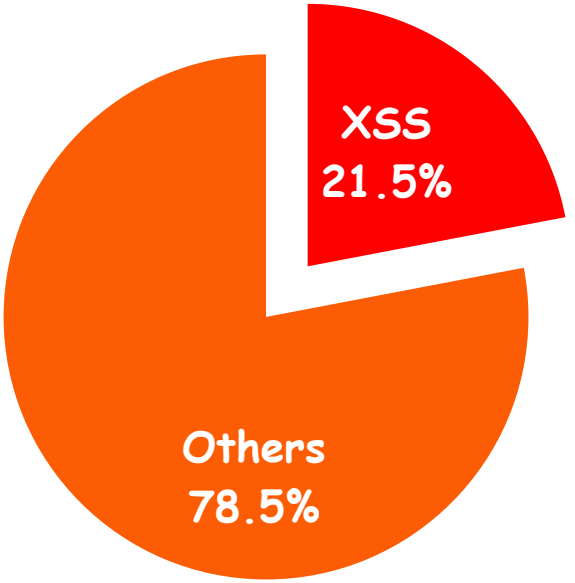
UIC UNIVERSITY OF ILLINOIS AT CHICAGO

XSS attacks : number one threat

CVE Vulnerabilities 2004



CVE Vulnerabilities 2006

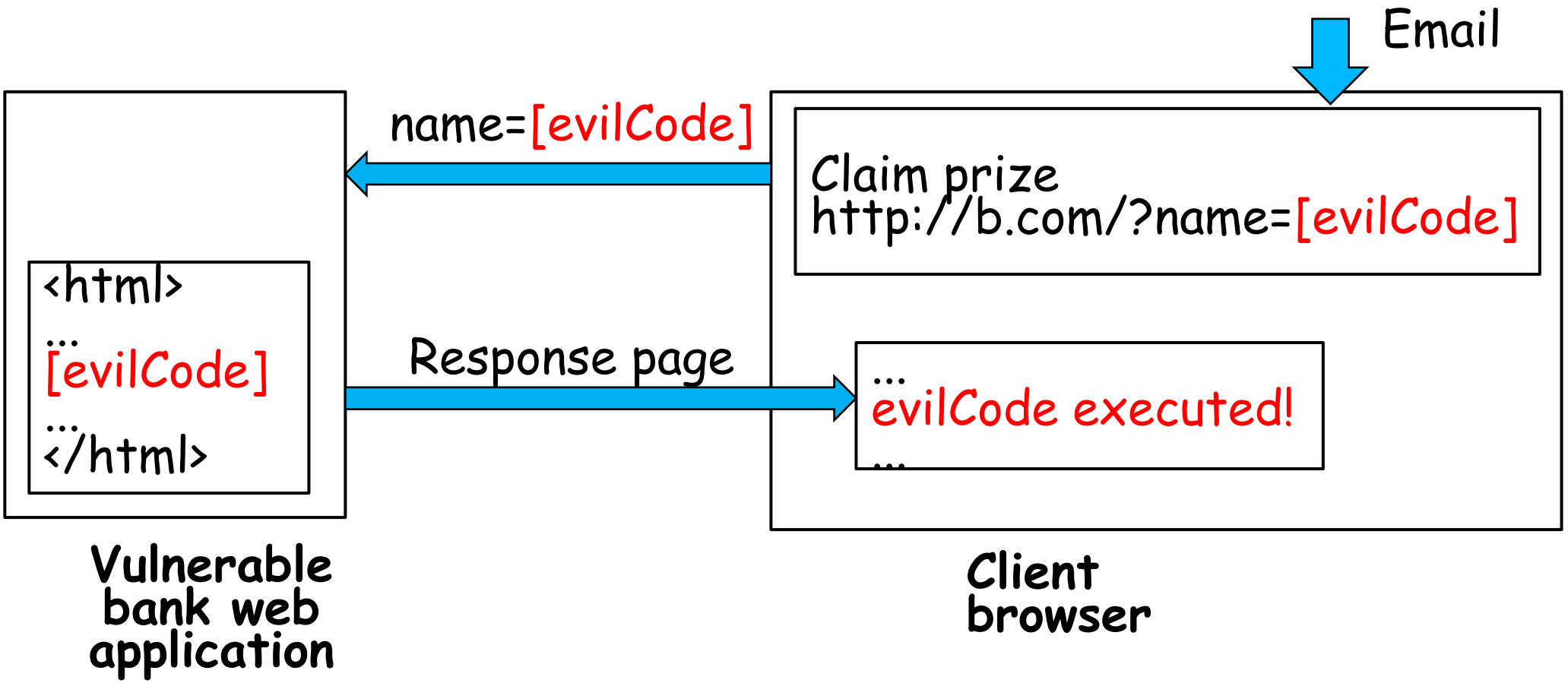


- …and the trend continues…
  - Second half of 2007 : 80% of all attacks were XSS
  - January 2007 : 70% web applications are vulnerable

    [source : http://en.wikipedia.org]
- Simple attacks lucrative targets
  - <script> alert('xss');</script>

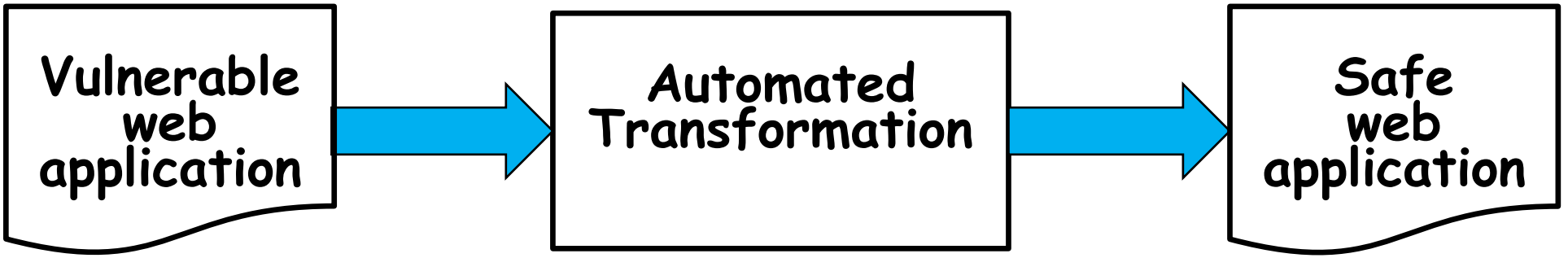
UIC UNIVERSITY OF ILLINOIS AT CHICAGO

A typical XSS attack



- Attacker controlled code can steal sensitive information or perform malicious operations.

Objective

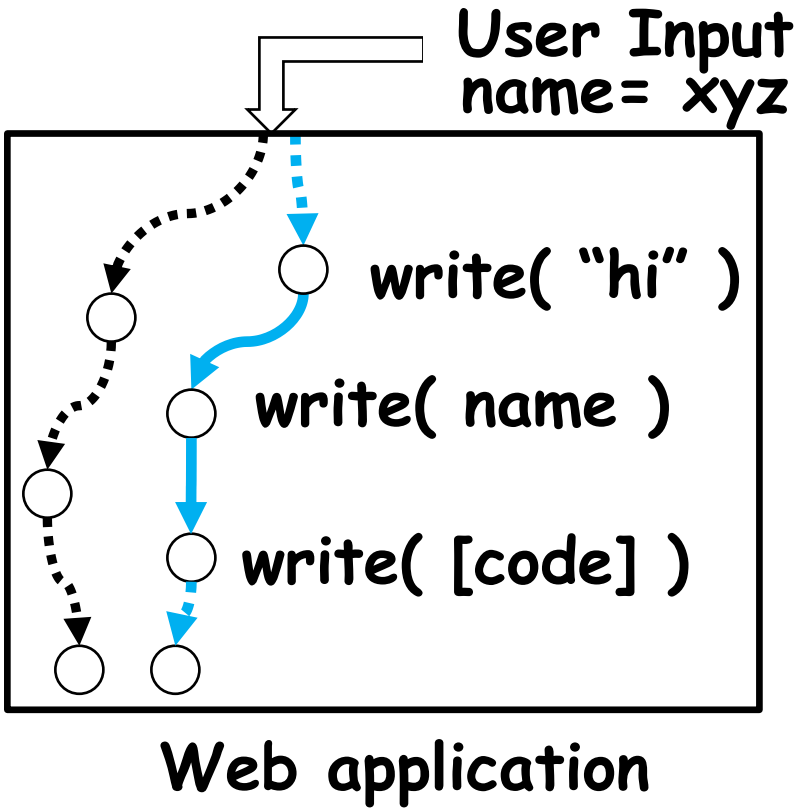


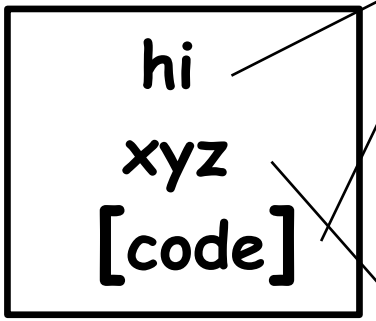
- **Automated** prevention of XSS attacks : server side
- **Robust** against subtle attacks
- **Efficient**

Outline of this talk

- Introduction
- Web application transformation technique
- Robust script identification at server side
- XSS-GUARD
  - Examples
  - Evaluation results
- Related work and summary

HTML page : A web application's view



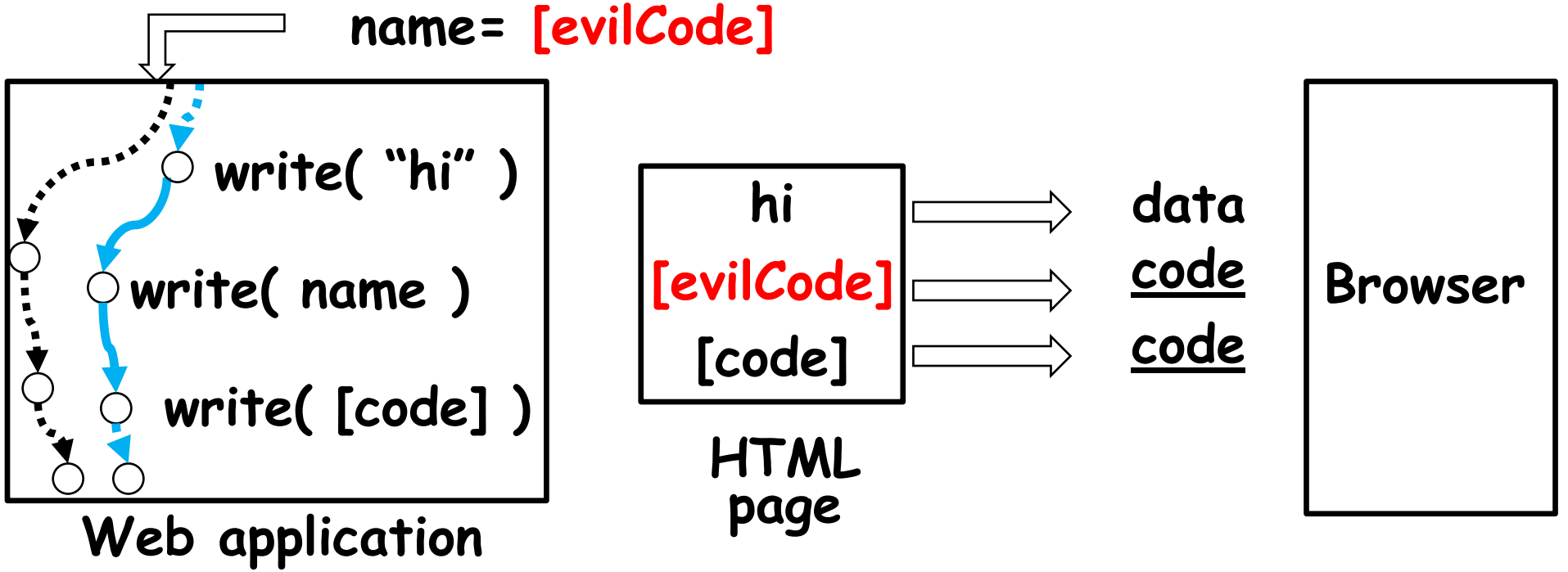


Output statements **not influenced** by user inputs produce programmer **intended** script code/data

Others may produce **unintended** script code

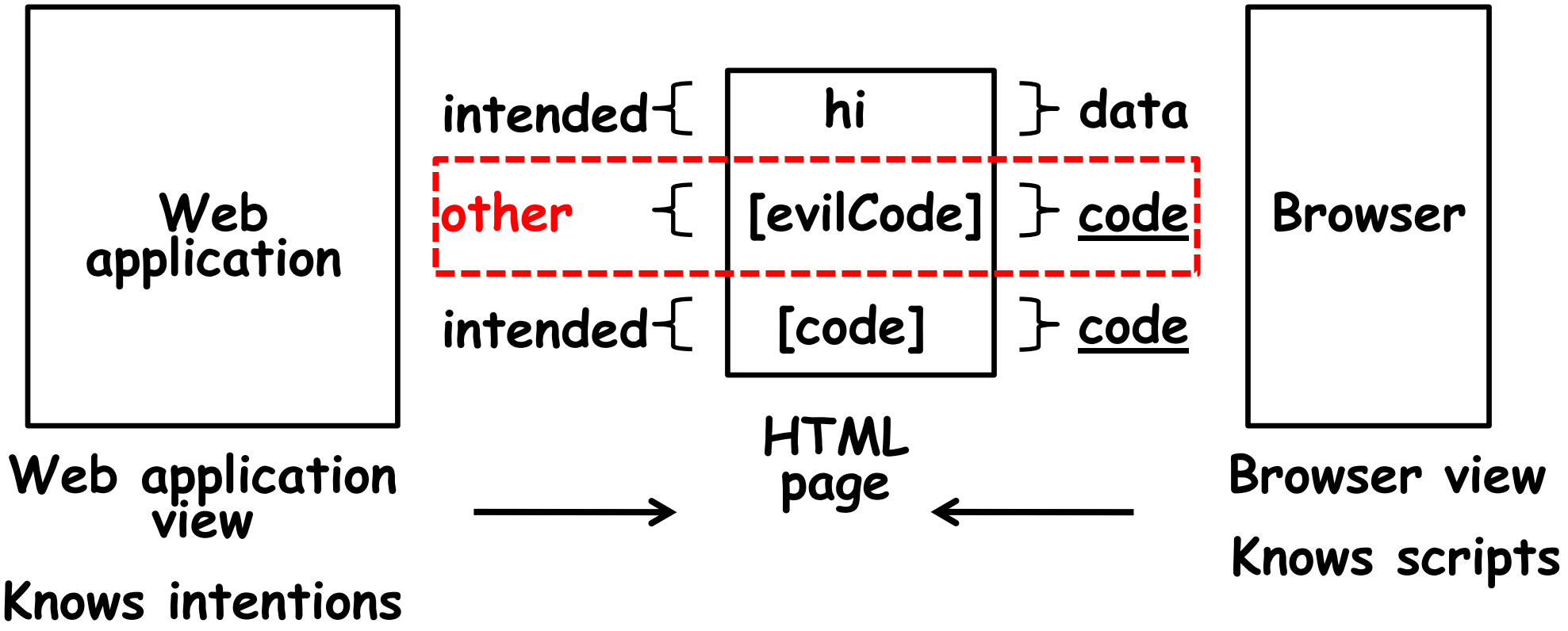
- Page generated by output statements in a control path.
- Web application's view : **intended regions** & **others**
  - **Other regions** could lead to unintended script code.

HTML page : A browser's view



- Browser does not differentiate between injected and programmer intended scripts.
- Browser's view : a collection of script **code** & **data**.

A complete view



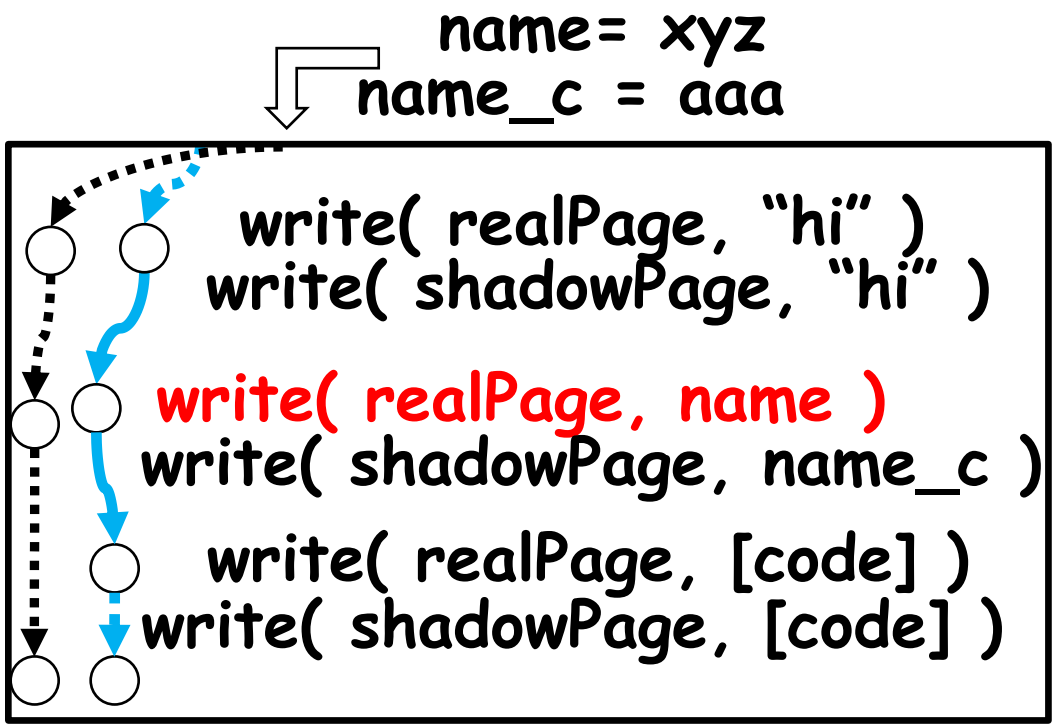
- An effective defense would require both these views!

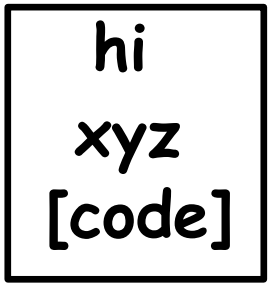
Idea

If a web application knows
intended scripts
and
all the scripts (including injected)
for a generated HTML page, it can
remove unintended scripts.

Question : How to compute intended scripts?

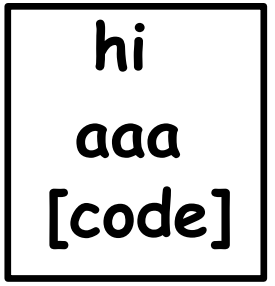
Computing intended code





data
data
code

Real HTML page



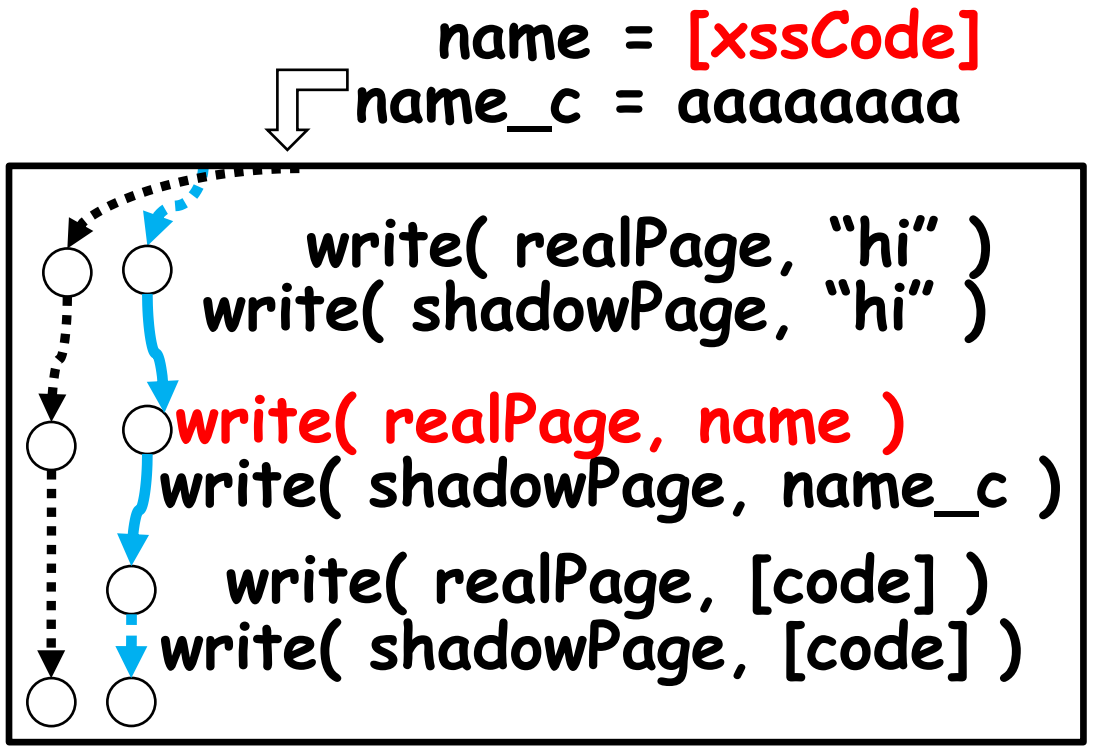
data
data
code

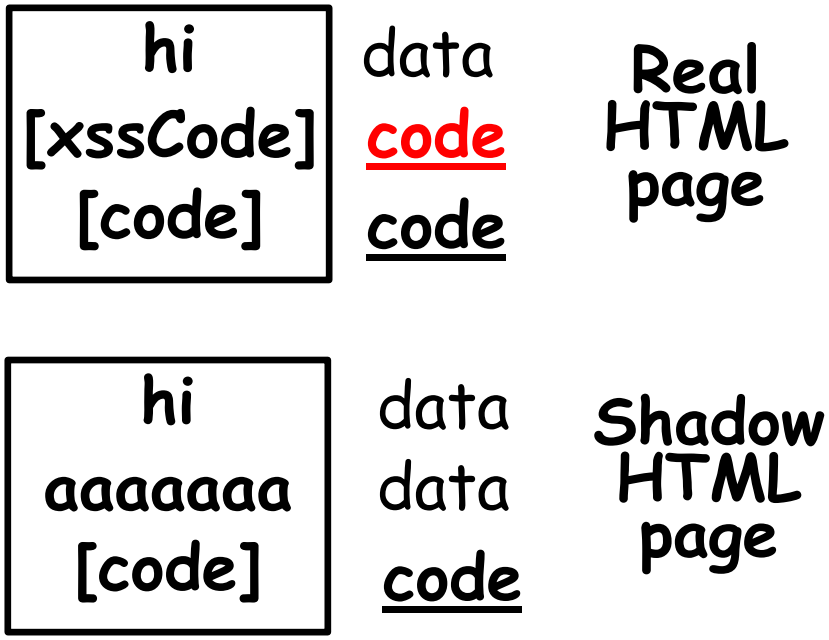
Shadow HTML page

- Replicate output statements uninfluenced by user inputs to create **shadow page**.
- Other output statements replicated but act on benign inputs (**as intended**).

UIC UNIVERSITY OF ILLINOIS AT CHICAGO

Computing intended code…cont.





- Real page contains injected script, but shadow retains only intended script.
- For a real page, **its shadow page has intended scripts.**

Shadow page captures intended code

- Real HTML page = output statements with **user inputs**
- Shadow HTML page = **mirror** above output statements with **benign user inputs**
- Transform web application to create shadow (intended) page for each real page
  - Define "benign input" for each "real input".
  - Mirror the "actual input" processing on "benign input".
  - Replicate output statements with above processed inputs.

For details on transformation, please refer to

- CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations, S. Bandhakavi, P. Bisht, P. Madhusudan, V.N. Venkatakrishnan, ***ACM CCS 2007***, **submitted to *ACM TISSEC* 2008**

Idea Revisited

If a web application knows

intended scripts

and

all scripts (including injected)

for a generated HTML page, it can

remove unintended scripts.

Question : How to compute intended scripts?

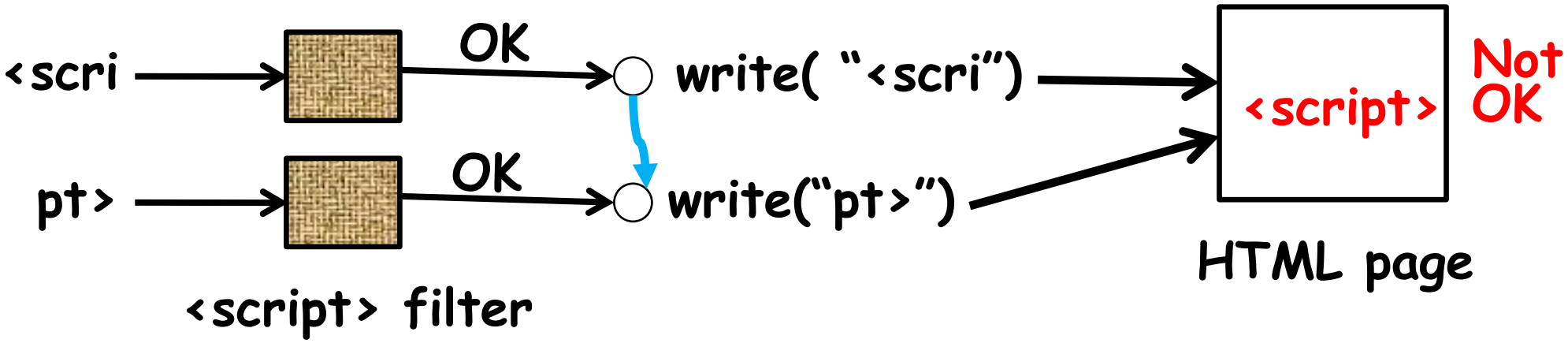
- By computing shadow pages.

Question : How can application identify all scripts?

What about filters?

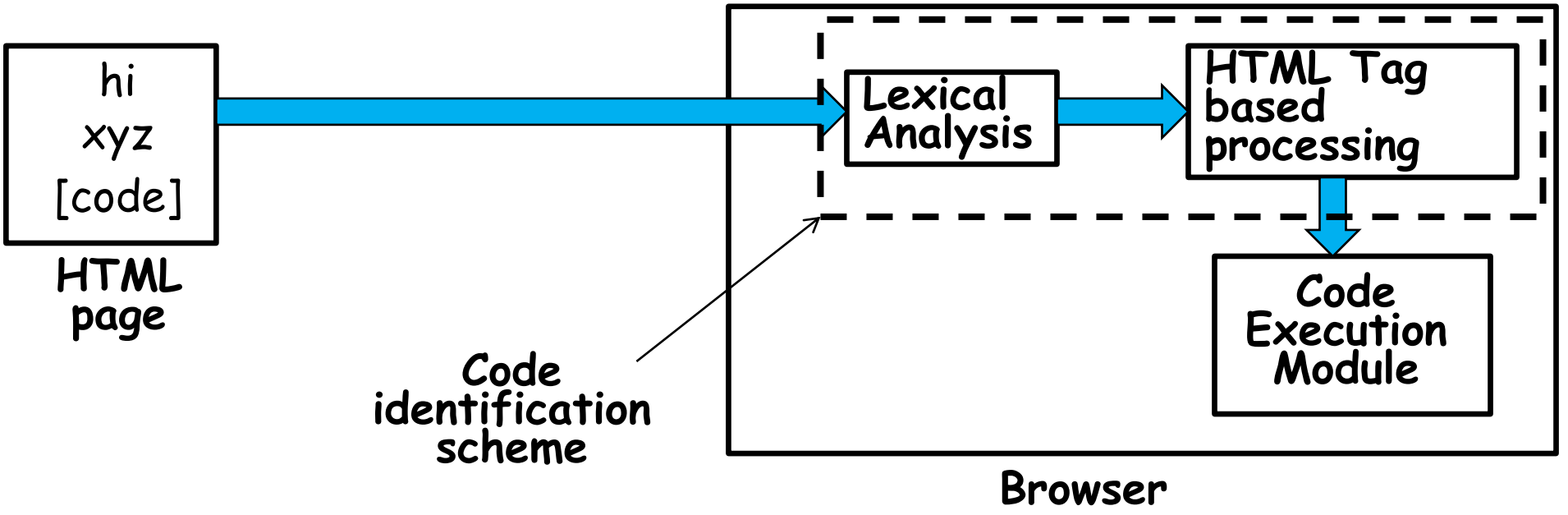
Filters effective first layer…but lack context

- Ineffective against subtle cases
  - MySpace Samy worm used eval('inner' + 'HTML') to evade "innerHTML" filter.
- Large attack surface
  - tags and URI schemes, attributes, event handlers, alternate encoding…
- Filters analyze inputs without their context of use.



- **Alternate scheme: find scripts in output (HTML page)**
  - **Inputs embedded in context of use in HTML page**

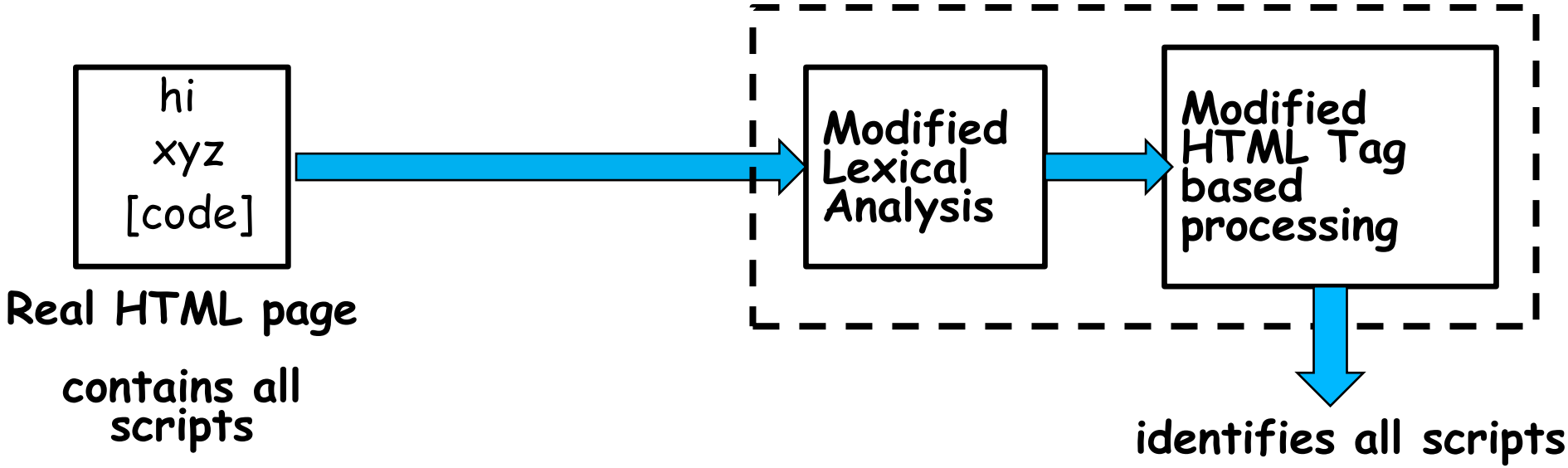
How Firefox identifies scripts?



- Lexical analysis component identifies tokens.
- HTML tag based processing identifies scripts in:
  - External resource download e.g., <script src=...>
  - Inlined scripts/event handlers e.g., <body onload=...>
  - URI schemes that can have scripts e.g., javascript/data

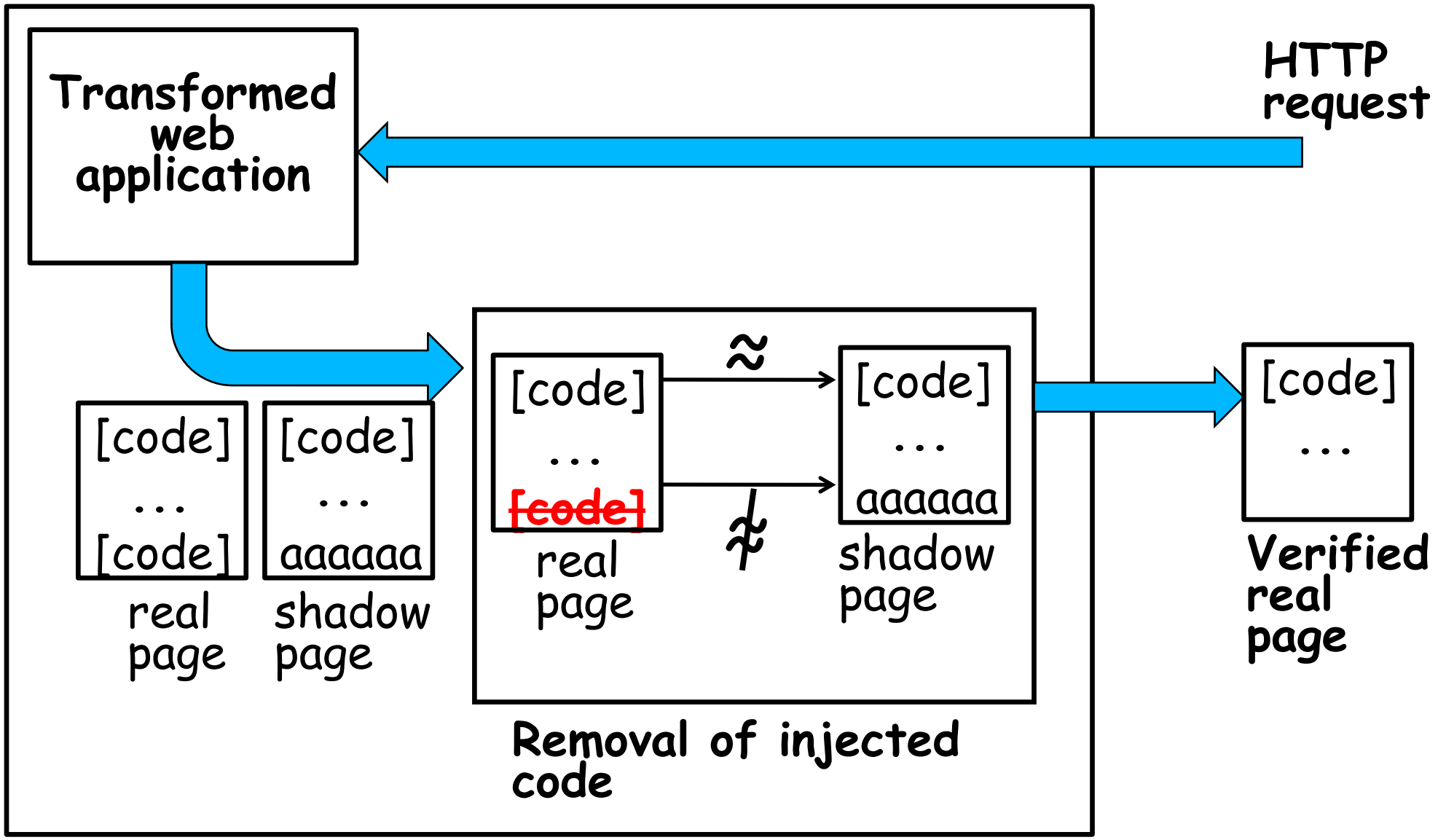
Leveraging browser's code identification mechanism

- A browser performs precise identification of scripts.
- Robust
  - alternate encodings
  - large attack surface
- **Our approach leverages this at the server side.**



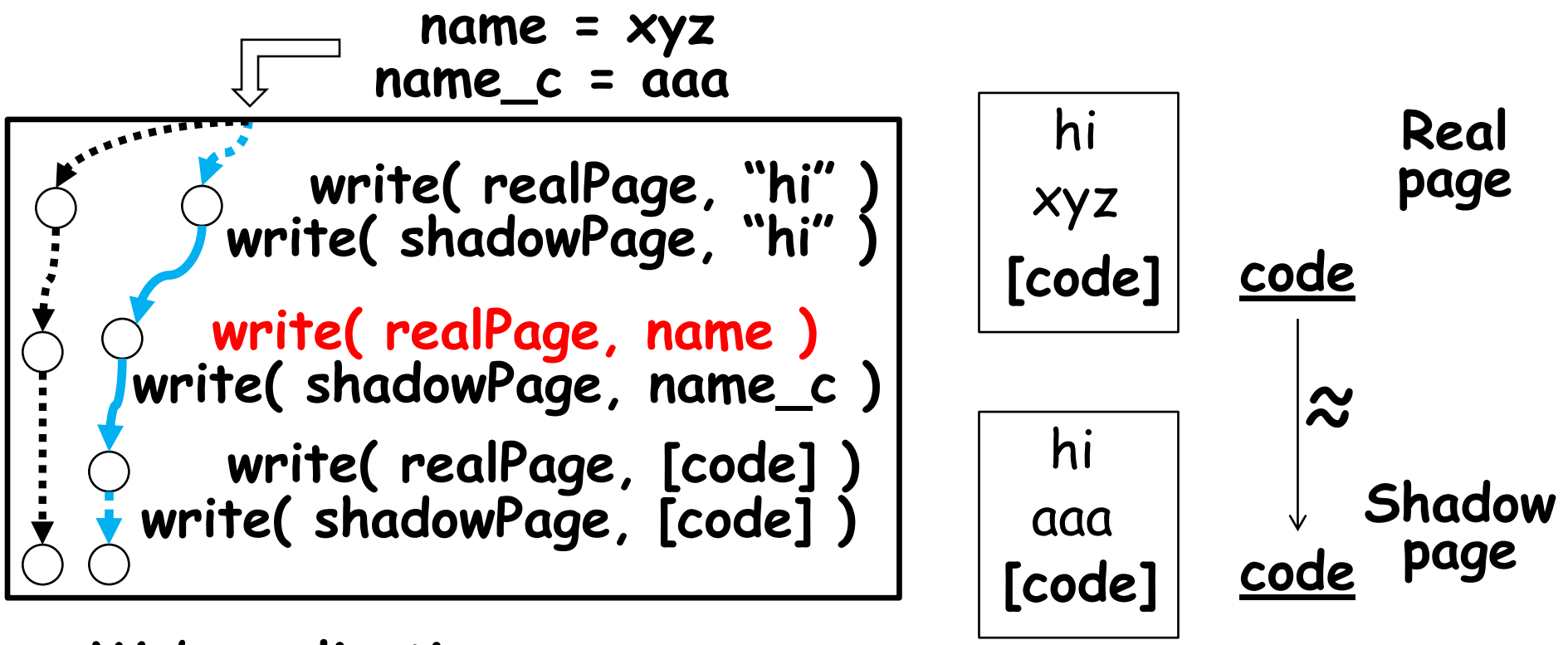
  - Modifications record all scripts in real HTML page.

XSS-GUARD : End - to - End



UIC UNIVERSITY OF ILLINOIS AT CHICAGO

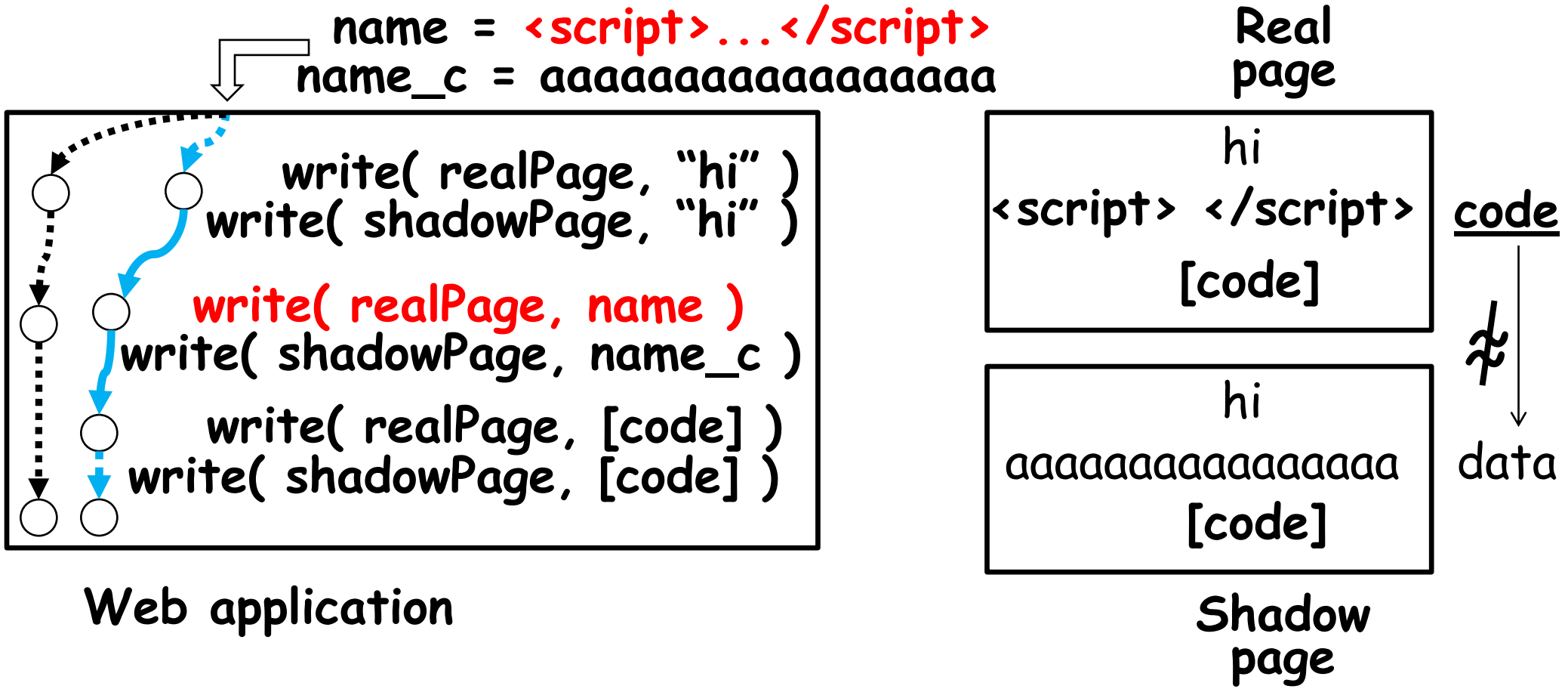
XSS-GUARD : intended scripts



- All intended scripts in real page have equivalent scripts in shadow page.

UIC UNIVERSITY OF ILLINOIS AT CHICAGO

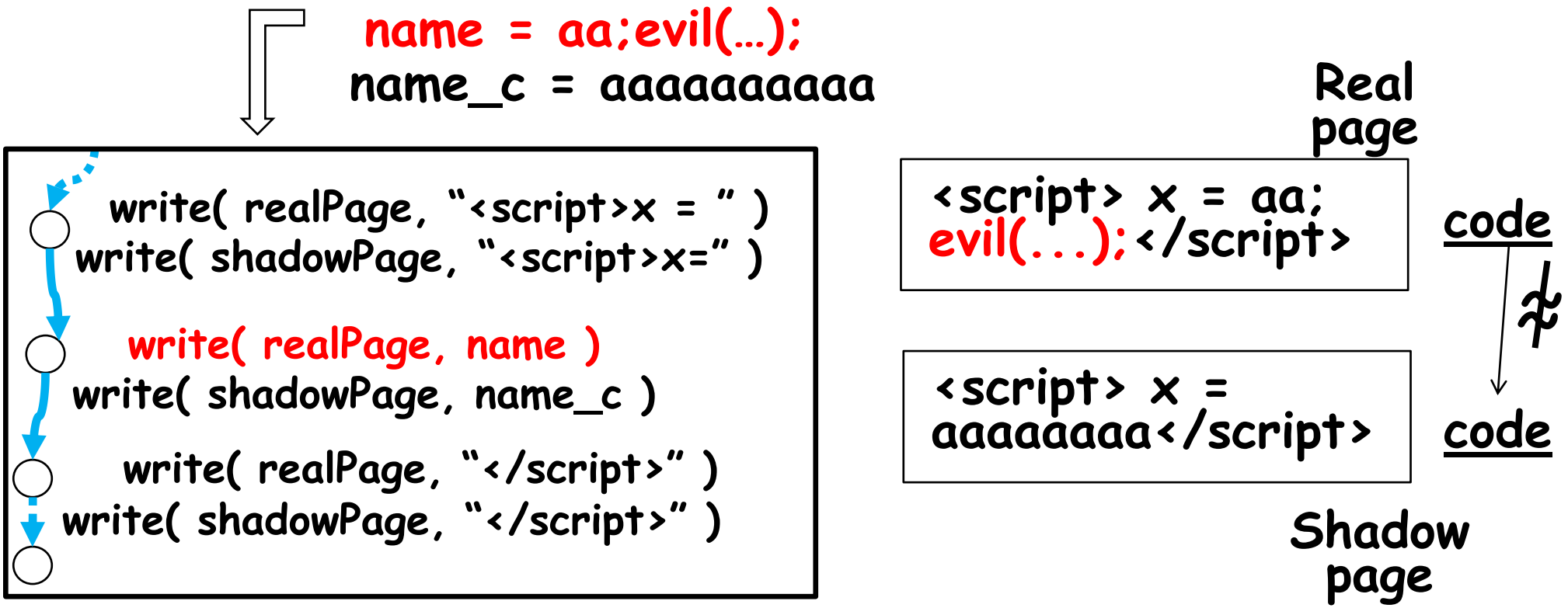
XSS-GUARD : Attack prevention



- Injected script in real page does not have equivalent script in shadow page, and is removed.

UIC UNIVERSITY OF ILLINOIS AT CHICAGO

XSS-GUARD : Subtle attack case



- Any unintended addition to existing scripts is successfully prevented.

UIC UNIVERSITY OF ILLINOIS AT CHICAGO

Effectiveness Evaluation

- Against real world exploits

CVE-2007-5120/5121	JSPWiki	defended
CVE-2007-2450	Tomcat HTML Manager	defended
CVE-2007-3386	Tomcat Host Manager	defended
CVE-2007-3383/3384/2449 CVE-2006-7196	Tomcat Example Web Applications	defended

- Defended 32 applicable exploits out of 92 : R. Hansen XSS cheatsheet.
- **False negatives :** non-Firefox attacks
  - Current implementation can be extended
  - initial experiments : Defended 35 / 56 non-Firefox attacks

UIC UNIVERSITY OF ILLINOIS AT CHICAGO

Performance Evaluation

- Performance overhead (response time)

Exploits from CVE	**5 – 24 %**
Varied response sizes (1KB – 75KB)	**3 – 14 %**
Parse tree comparison of scripts (1 – 5)	**37 – 42 %**

- Parse tree comparison is rarely done : in presence of attacks, or scripts embedding user inputs.
- These numbers indicate worst case performance –
  - Negligible network latency in experiments (LAN setup)
  - Can be further improved by limiting the transformation to only relevant statements.

Some Related Work

- **Vulnerability analysis:** find vulnerable source-sink pairs e.g., saner: Livshits et al. Usenix 2005, Pixy N. Jovanovic et al. S&P2006, Y. Xie et al. Usenix 2006, D. Balzarotti et al. CCS 2007...
  - Useful but limited to detection
- **Server side solutions:** filter based or track taint & disallow at sink : W. Xu et al. Usenix 2006, ...
  - Centralized defense but do not know all scripts
- **Client side solutions:** Firewall like mechanisms to prevent malicious actions at client
  - Noxes E. Kirda, et al. SAC 2006, P. Vogt et al. NDSS 2007
  - User controlled protection but do not know intended scripts
- **Client-Server collaborative solutions:** Clients enforce application specified policies
  - BEEP T. Jim, et al. WWW 2007, Tahoma R. Cox et al. S&P 2006, Browsershield C. Reis et al. OSDI 2006
  - Can determine intended and all scripts but deployment issues

Contributions and future work

- A robust server side solution to prevent XSS attacks.
- A mechanism to compute programmer intended code, **useful in defending other code injection attacks**.
- Leveraged browser's mechanisms at server side.

Thanks for your attention!

Questions?

UIC UNIVERSITY OF ILLINOIS AT CHICAGO

Backup Slides

Taint vs Candidate evaluations

- Taint tracking captures "trust" notion.
- Candidate computation captures "intended structure".
- We found taint tracking and limiting tainted constructs in parse trees, a very powerful idea. There are no differences in effectiveness but there are subtle differences –
- X = taintedX – taintedX + 100.5;
  - X will be treated as tainted : false positive?
- X_c = taintedX_c – taintedX_c + 100.5;
  - X_c will contain equivalent structure to X.

Miscellaneous examples

- Non-Firefox construct based false negative
  - <IMG SRC='vbscript:msgbox("XSS")'>
    Firefox does not understand *vbscript* URIs.

- Firefox quirk based exploit
  - <script/xss src="”> </script>

Changes done to handle non-Firefox quirks

- Others **(3)**
- Attacks based on **src** attributes **(2)**
  - <XML SRC="xsstest.xml" ID=I></XML> <SPAN DATASRC=#I DATAFLD=C DATAFORMATAS=HTML></SPAN>
  - <SCRIPT a=`>` SRC="http://ha.ckers.org/xss.js"></SCRIPT>
- Javascript / data **URI scheme** based exploits **(30)**
  - <img src="javascript:alert('XSS')">Is ignored in Firefox, but valid in other browsers.
  - We forced all the attributes to be parsed irrespective of Firefox applicability.

Candidate Transformation

Performance numbers (detailed)

Related work

- Filters : don't have adequate context
- Output encoding : may forbid all HTML
- Taint : Effective, focuses on taintedness, rather than semantics
- Server side solutions :
- Client side solutions : Firewall like behavior, may disallow legitimate scripts, or allow attacks on trusted servers.
- Server-client side solutions

Real page and shadow page comparison with offsets

False positive in subtle case

Example with existing filters - prevent attack