WebShield: Enabling Various Web Defense Techniques without Client Side Modifications

Zhichun Li, Tang Yi, Yinzhi Cao, Vaibhav Rastogi,
Yan Chen, Bin Liu, and Clint Sbisa

NEC Laboratories America, Inc.

Northwestern University

Tsinghua University

Web Has Become a Primary Target

Drive by Download

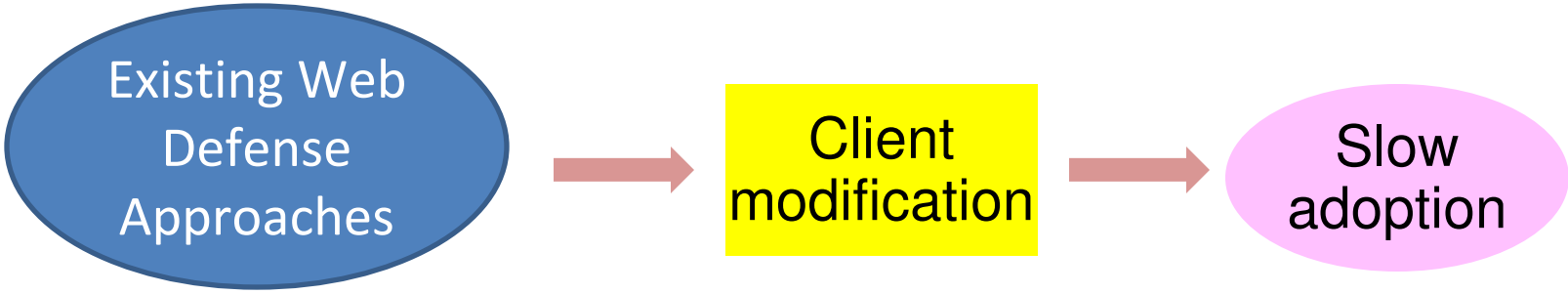
Cross site scripting

Cross Site Request Forgery

Cross-Origin JavaScript Capability Leaks

Desire a General Middlebox

- Existing web defense techniques need browser/client modification



- Advocate middlebox approaches

Client-side	Middlebox
heterogenous & co-exist with other software	clean installation
high maintenance overhead	centralized control
user voluntary update	easy update and VM management

General Design Principles for Middlebox

- Principles
  - Principle I: general middlebox should enable various protection mechanisms
  - Principle II: avoid client-side deployment
  - Principle III: containment of untrusted script execution
  - Principle IV: should not sacrifice user experience

Existing Middlebox Approaches

- BrowserShield
  - Code rewriting: rewrite HTML and JavaScript code with policy checking wrappers
  - Only applies to known browser vulnerabilities
  - Hard to be extended to support other defense mechanisms
- SpyProxy
  - Actively execute the web pages in a proxy sandbox
  - Applies to both known and unknown vulnerabilities
  - But only detect deterministic exploits

Evade Existing Approaches

```
function attackX() {
// exploit an unknown vulnerability,
// so BrowserShield cannot be applied
    ...
}
var attackcalled=false;
function loadAttack() {
    var el=document.getElementById("Evil");
    // use user events to bypass SpyProxy
    el.addEventListener("mouseover"
}
fu
}
```

Very Easy to Implement

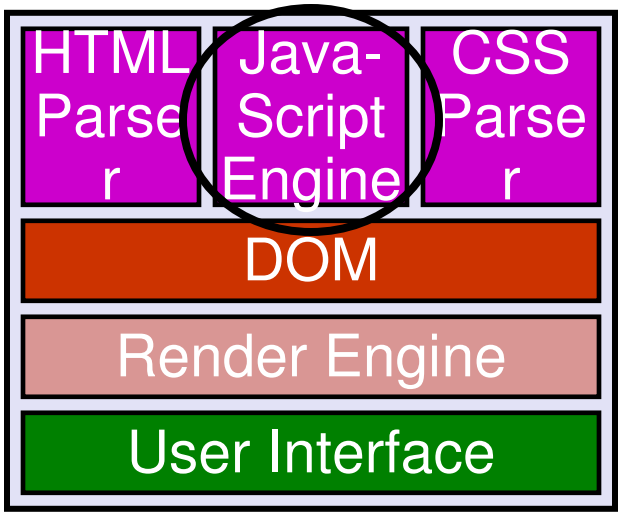
checkMouse → attackX

Outline

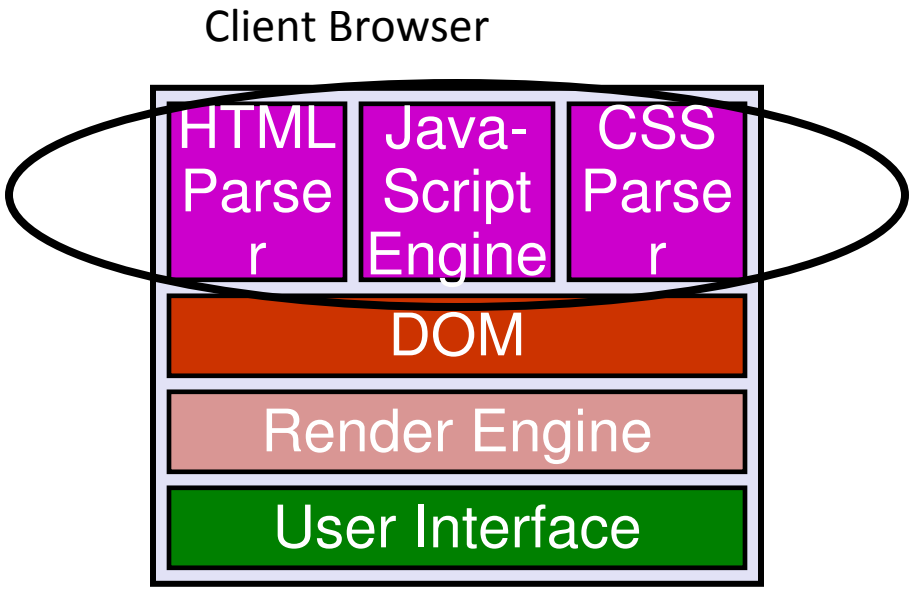
- Our Design
- Implementation
- Evaluation
- Conclusion

Our Design

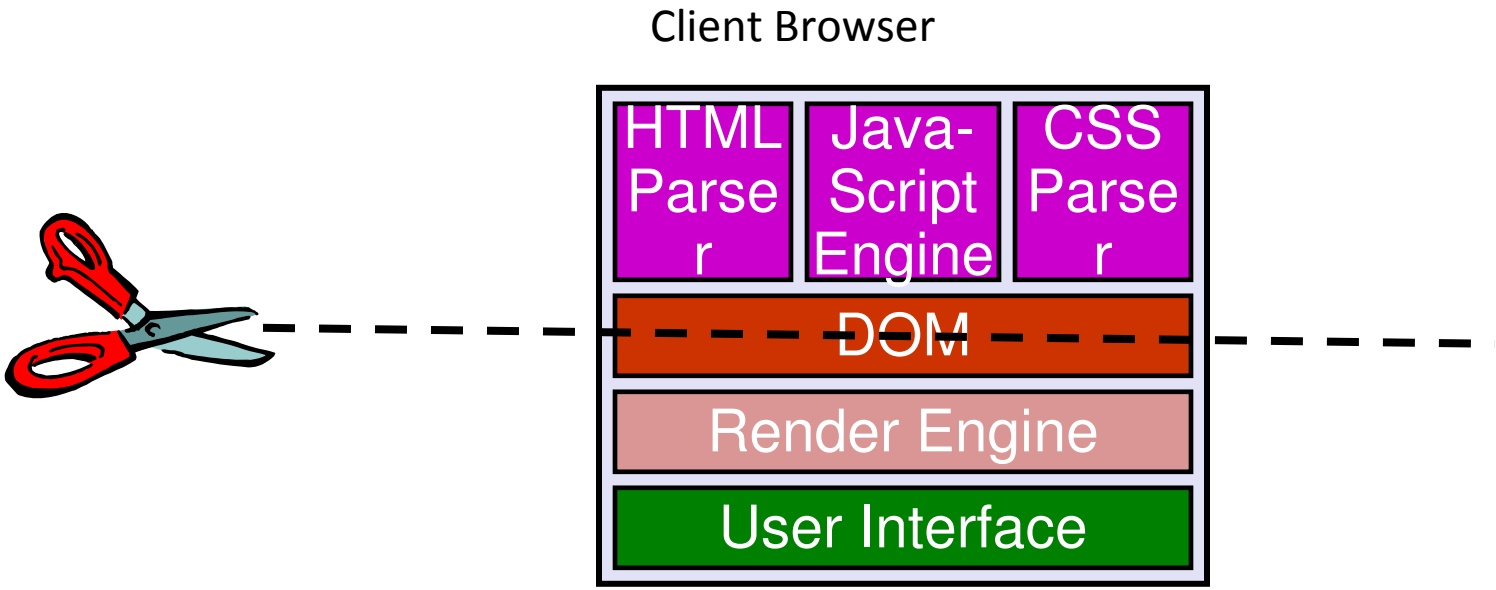
Client Browser



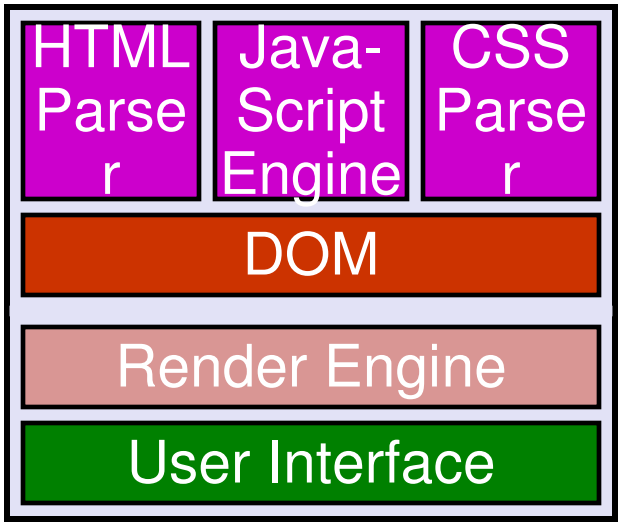
Our Design


Client Browser
HTML Parser
Java-Script Engine
CSS Parser
DOM
Render Engine
User Interface

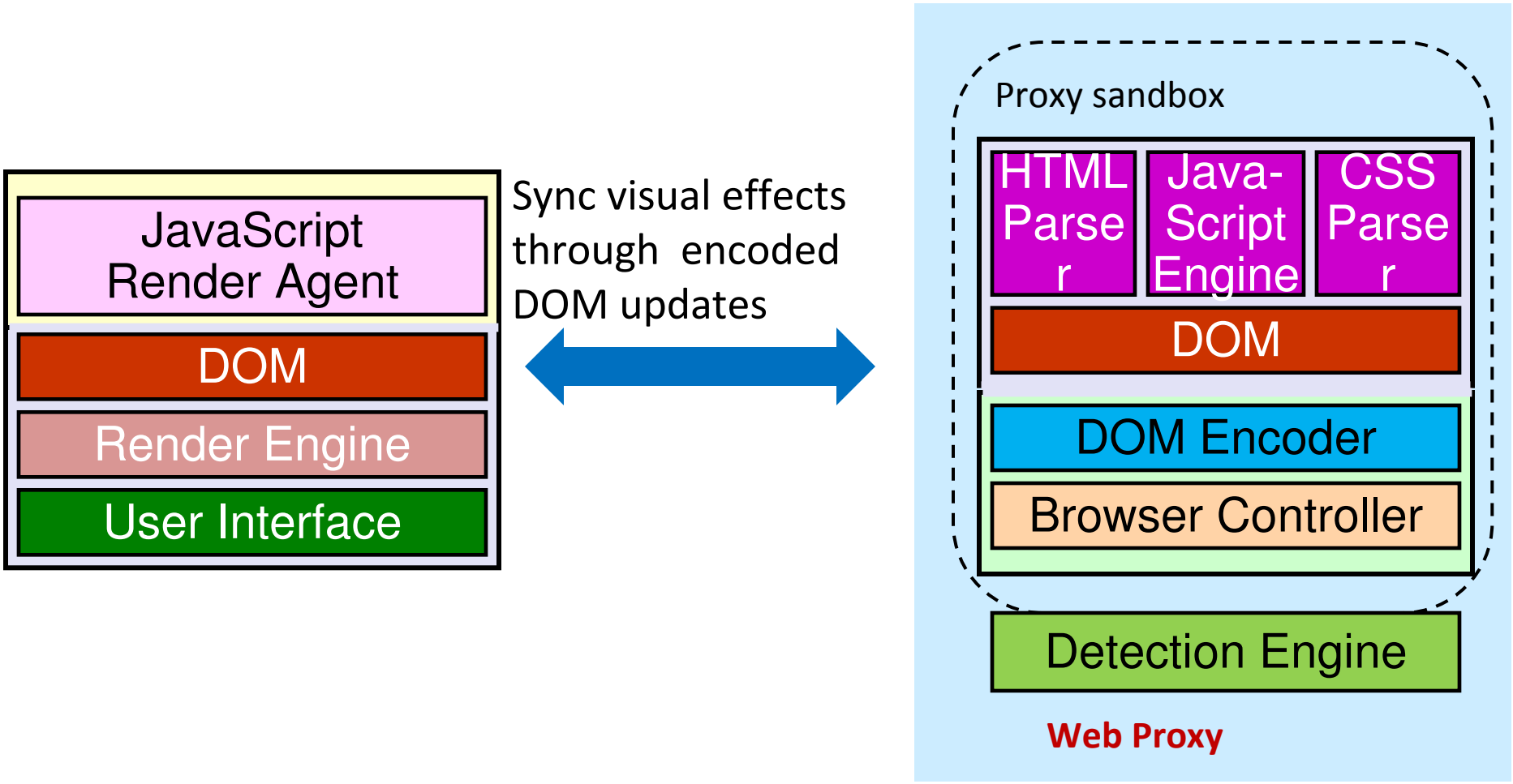
Our Design


Client Browser
HTML Parser
Java-Script Engine
CSS Parser
DOM
Render Engine
User Interface

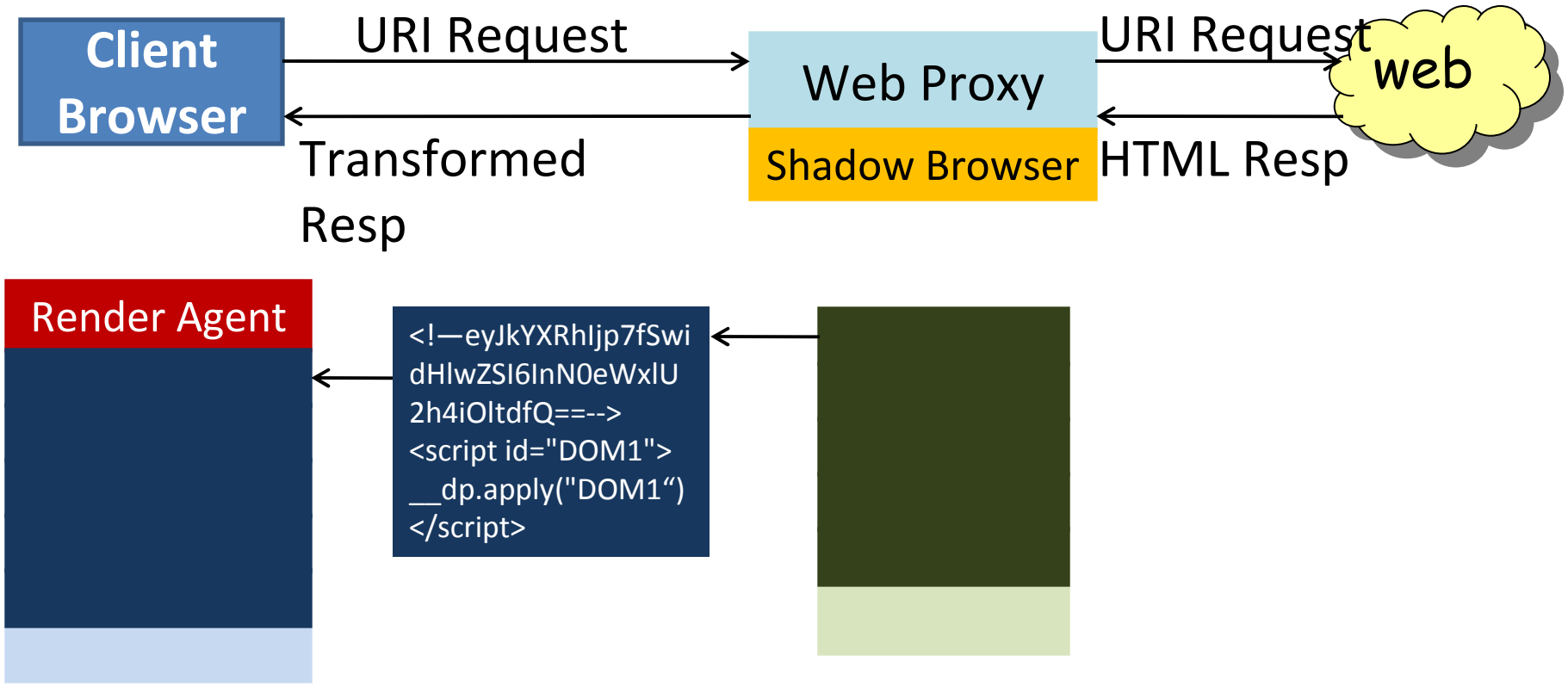
Our Design


HTML Parser
Java-Script Engine
CSS Parser
DOM
Render Engine
User Interface

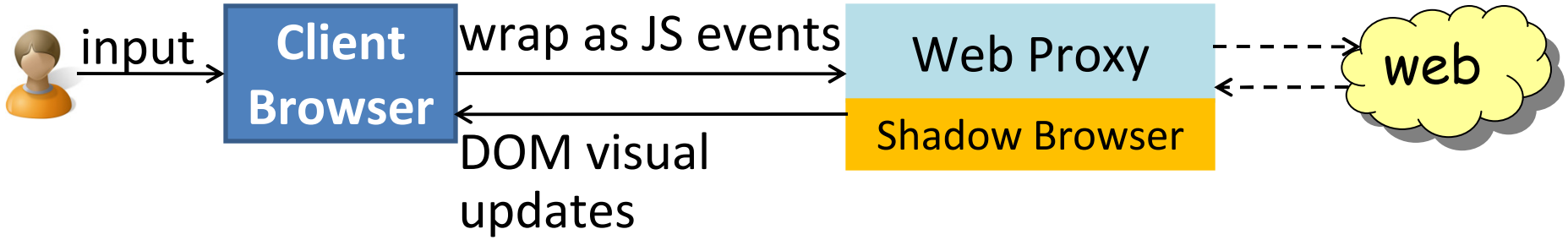
Our Design


JavaScript Render Agent
DOM
Render Engine
User Interface
Sync visual effects through encoded DOM updates
Proxy sandbox
HTML Parser
Java-Script Engine
CSS Parser
DOM
DOM Encoder
Browser Controller
Detection Engine
Web Proxy

Initial Page Render

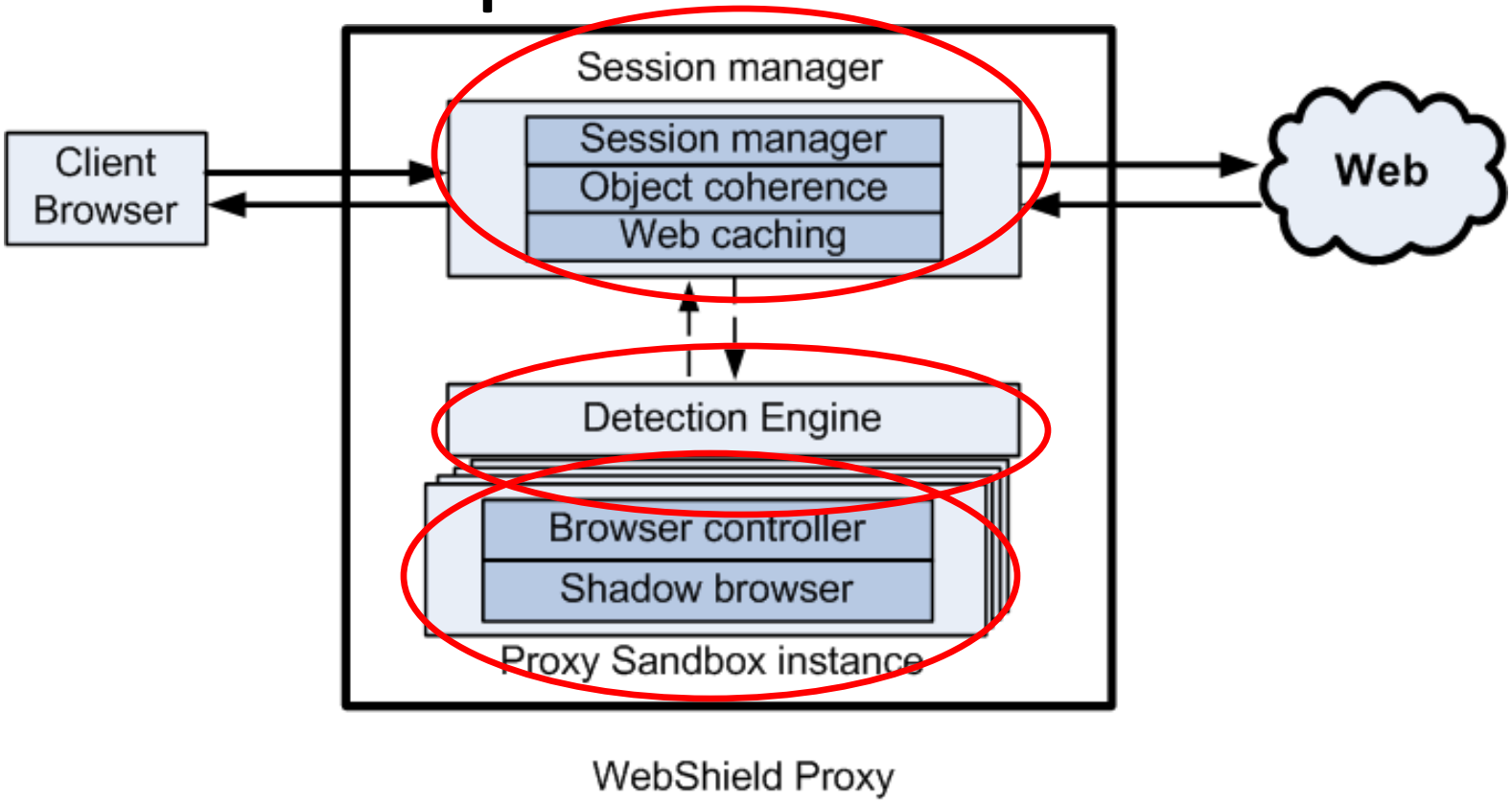


Dynamic HTML Interaction Support



- Latency added
  - Communication delay
  - DOM update delay
- DOM tree update location
  - Element ID
  - Location vector starting from the root of the tree

Implementation



- Use Webkit to implement Shadow browser
- Current sandbox based on SELinux
- Session manager in Python

Outline

- Our Design
- Implementation
- Evaluation
- Conclusion

Evaluation

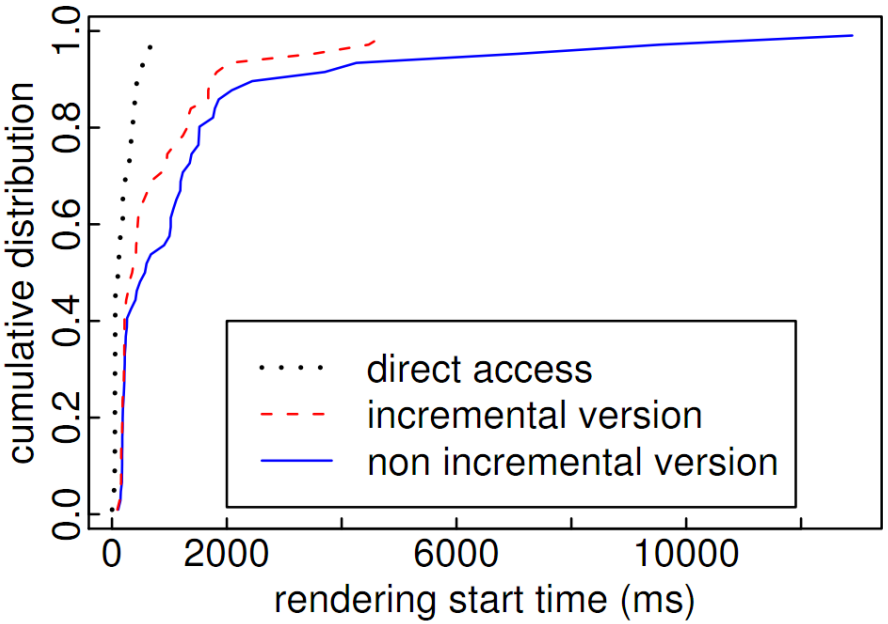
- Environment Setup
  - Web Proxy: 2.5GHz Intel Xeon server
  - Web Browser    on Core2 2.66GHz
- Evaluation Metrics
  - Compatibility
  - Performance (user transparency)
    - Latency
    - Memory
    - Communication overhead
  - Drive-by-download detect demonstration

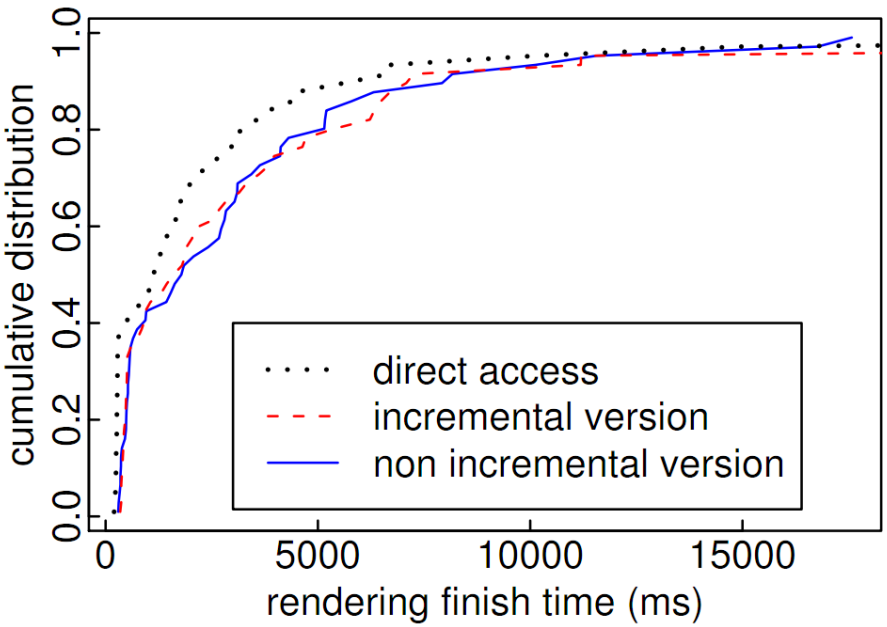
Evaluation

- Compatibility
  - 91 out of Alexa top 100 web sites
  - 19 out of Alexa top 20 web sites
  - Reasons for not compatible websites
    - Not supported features
    - Stability of the prototype

Latency Overhead

- Initial page rendering
  - Evaluate Alexa top 100 sites
  - Render start: median +134ms, 90th percentile +1.08 sec
  - Render end: median +382 ms, 90th percentile +2.46 sec

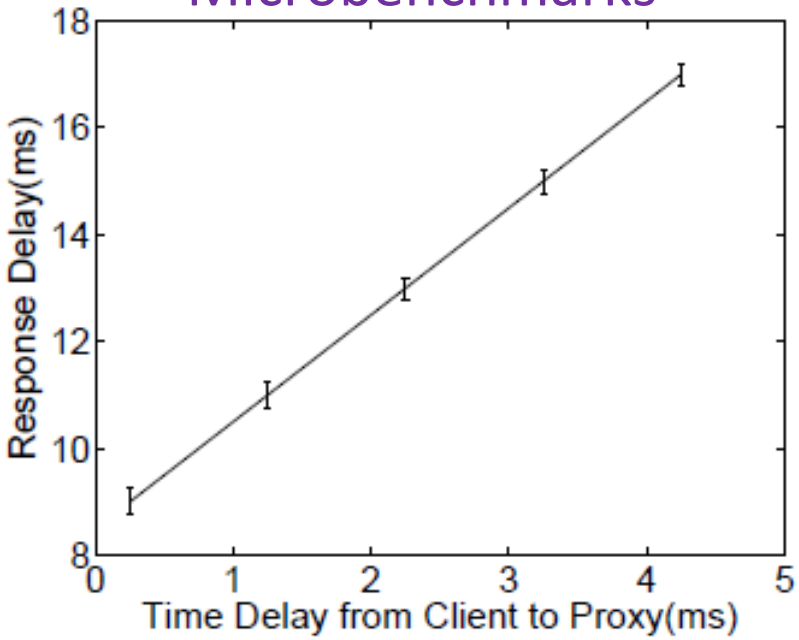


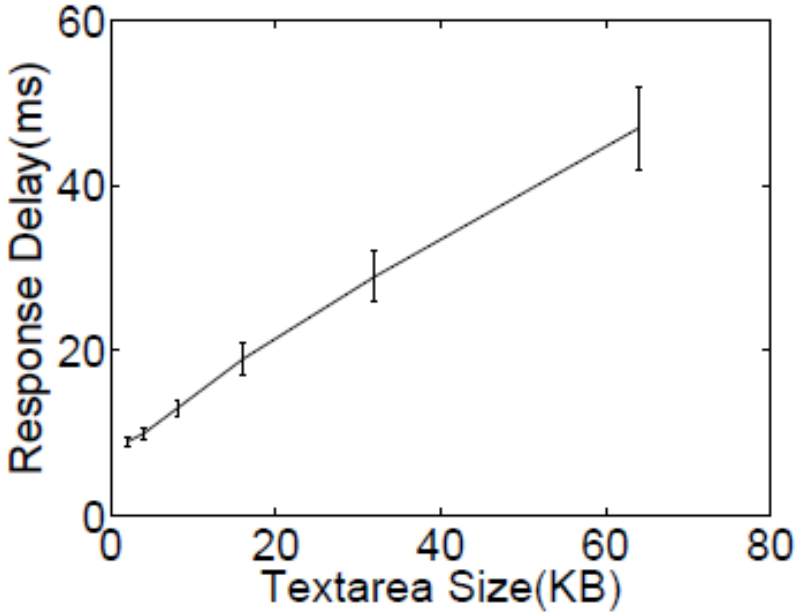


Chrome render start and end time

Latency Overhead

- Interactive Performance for Dynamic HTML
  - Microbenchmarks



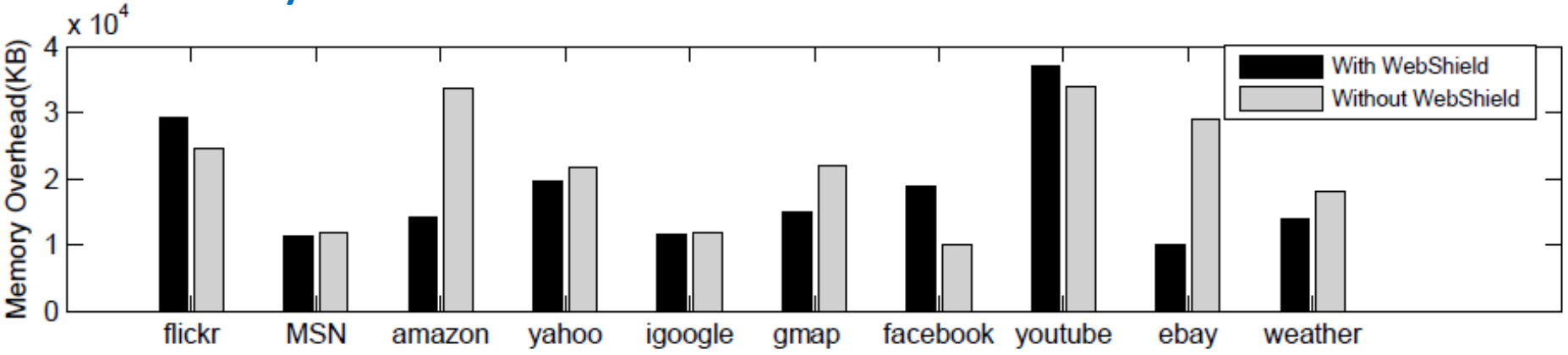


  - Test on a real JavaScript game: JavaScript Game – connect 4

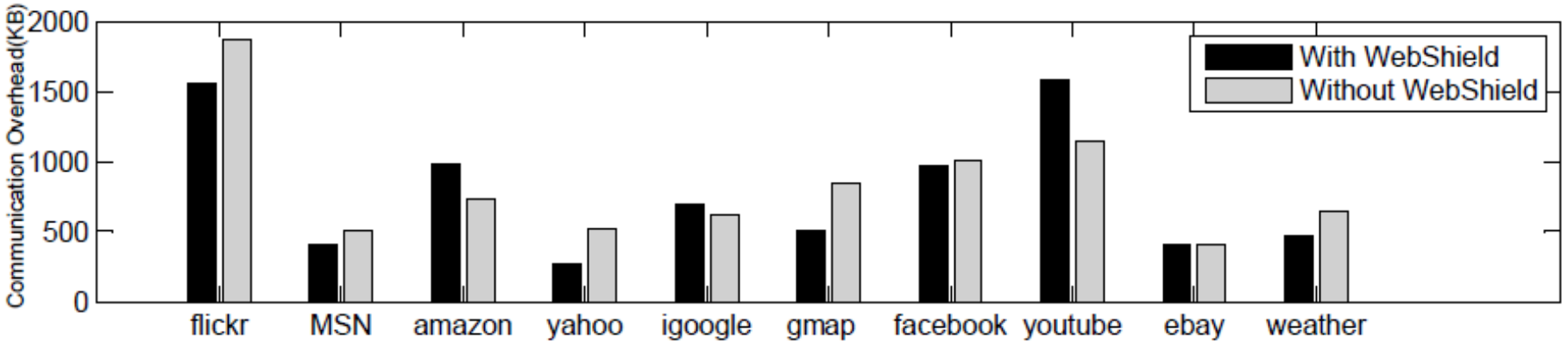
	Start Game	Move Mouse	Drop a Piece	Game Over
Additional Delay	41ms	7ms	10ms	7ms

Memory and Communication Overhead

- Memory overhead



- Communication overhead



Usefulness Demonstration

- Drive-by-download detection
  - Implement both policy-based and behavior-based detection
    - Policy-based: check the parameters of JavaScript API calls and the parsing process
    - Behavior-based: check a list of abnormal behaviors similar to SpyProxy
  - Evaluate eight vulnerabilities with Alexa top 500 web sites.

Detection plug-ins	False Negative	False Positive
Policy Engine	0	1/500
Behavior Engine	0	0/500

Conclusion

- We design, implement and evaluate WebShield
  - A general middlebox that enables various web defense mechanisms
  - Run JavaScript inside the middlebox, and thus reduce the attack surface
  - No client modification
  - Small overhead for latency, communication and memory → remain good user experience

Advertisement

- Positions available for system people (OS, Network, and Security) in NEC Research Labs
  - Full-time
  - Interns

Q & A