

Spectator:

Detection and Containment of JavaScript Worms

**Ben Livshits and Weidong Cui
Microsoft Research
Redmond, WA**

Web Application Security Arena

- Web application vulnerabilities are everywhere
- Cross-site scripting (XSS)
 - Dominates the charts
 - “Buffer overruns of this decade”
 - Key enabler of JavaScript worms

The Samy Worm

- Unleashed by Samy as a proof-of-concept in October 2005



Worm name	Type of site	Release date
Samy/MySpace	Social networking	Oct-05
xanga.com	Social networking	Dec-05
SpaceFlash/MySpace	Social networking	Jul-06
Yamanner/Yahoo! Mail	Email service	Jun-06
QSpace/MySpace	Social networking	Nov-06
adultspace.com	Social networking	Dec-06
gaiaonline.com	Online gaming	Jan-07
u-dominion.com	Online gaming	Jan-07

Consequences?

- Samy took down MySpace (October 2005)
 - Site couldn't cope: down for two days
 - Came down after 13 hours
 - Cleanup costs
- Yamanner (Yahoo mail) worm (June 2006)
 - Sent malicious HTML mail to users in the current user's address book
 - Affected 200,000 users, emails used for spamming

Samy: Worm Propagation

The screenshot shows a Mozilla Firefox browser window titled "MySpace Profile - Mozilla Firefox". The address bar is empty. The page content includes a form titled "Enter information about yourself:". A blue box highlights the beginning of a JavaScript code block: `<SCRIPT>`. The code is a JavaScript worm that propagates by creating and submitting a form to itself. The code defines several variables and functions, including `F(N,M,Q,O)` for creating form elements, `H(N,T)` for setting attributes, and `A(O)` for submitting the form. The worm is triggered by a `<SCRIPT>` tag in the page source.

```
<SCRIPT>
(function(){var
G=YAHOO.util.Dom,L=YAHOO.util.Event,I=YAHOO.lang,B=YAHOO.widget.Overlay,J=YAHOO.widget.Menu,D=
{} ,K=null,E=null,C=null;function F(N,M,Q,O){var R,P;if(I.isString(N)&&I.isString(M))
{if(YAHOO.env.ua.ie){P="<input type=\""+N+"\" name=\""+M+"\"";if(O){P+=" checked";}P+=">";
R=document.createElement(P);}else{R=document.createElement("input");R.name=M;R.type=N;
if(O){R.checked=true;}R.value=Q;return R;}}function H(N,T){var
M=N.nodeName.toUpperCase(),R=this,S,O,P;function U(V){if(!(V in T)){S=N.getAttributeNode(V);
if(S&&("value" in S)){T[V]=S.value;}}function Q(){U("type");if(T.type=="button")
{T.type="push";}if(!("disabled" in T)){T.disabled=N.disabled;}U("name");U("value");
U("title");}switch(M){case"A":T.type="link";U("href");U("target");break;case"INPUT":Q();
if(!("checked" in T)){T.checked=N.checked;}break;case"BUTTON":Q();O=N.parentNode.parentNode;
if(G.hasClass(O,this.CSS_CLASS_NAME+"-checked"))
{T.checked=true;}if(G.hasClass(O,this.CSS_CLASS_NAME+"-disabled"))
{T.disabled=true;}N.removeAttribute("value");N.setAttribute("type","button");
break;}N.removeAttribute("id");N.removeAttribute("name");if(!("tabindex" in
T)){T.tabindex=N.tabindex;}if(!("label" in T)){P=M=="INPUT"?N.value:N.innerHTML;if(P&&P.length>0)
{T.label=P;}}function A(O){var N=O.attributes,M=N.srcelement,Q=M.nodeName.toUpperCase(),P=this;
if(Q==this.NODE_NAME){O.element=M;O.id=M.id;G.getElementsBy(function(R)
```

What's at the Root of the Problem?

- Worms of the previous decade enabled by buffer overruns
- JavaScript worms are enabled by cross-site scripting (XSS)
- Fixing XSS holes is best, but some vulnerabilities remain
 - The month of MySpace bugs
 - Database of XSS vulnerabilities: xssed.com

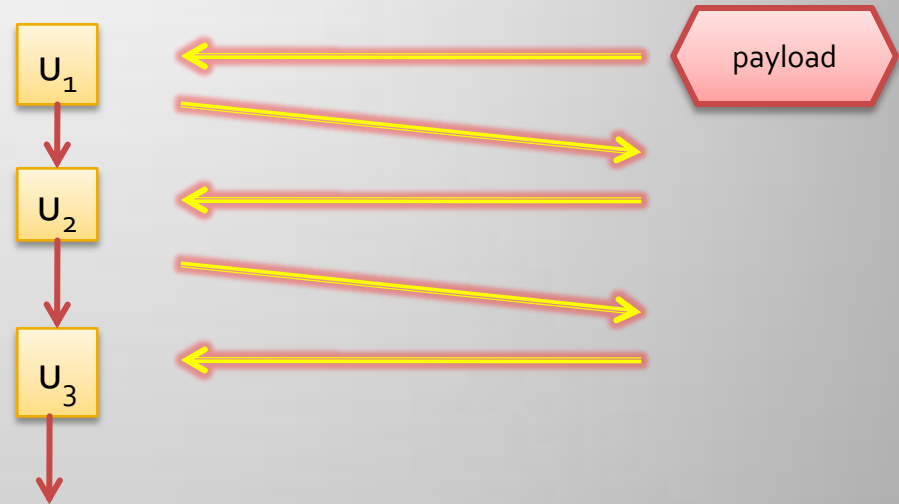
What Can We Do?

- Existing solutions rely on signatures
 - Ineffective: obfuscated and polymorphic JavaScript worms are very easy to write
 - Most real-life worms are obfuscated
- Fundamental difficulties
 - **Server** can't tell a user request from worm activity
 - **Browser** doesn't know where JavaScript comes from

Spectator: Approach and Architecture

Worm Propagation

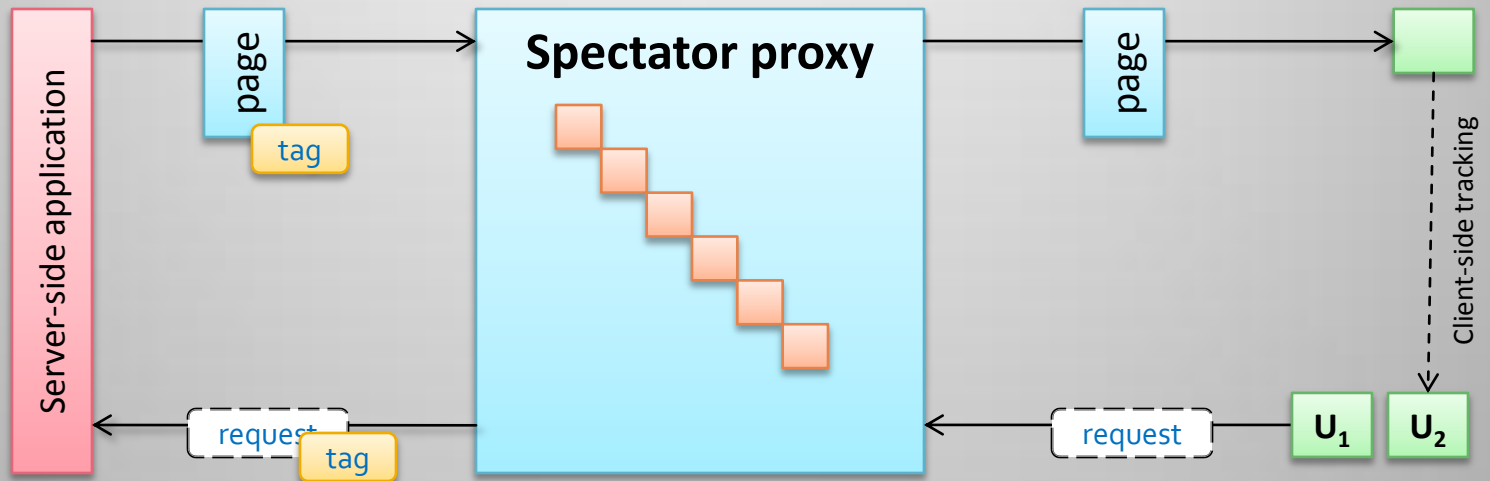
- u_1 uploads to his page
- u_2 downloads page of u_1
- u_2 uploads to his page
- u_3 downloads page of u_2
- u_3 uploads to his page
- ...



Propagation chain

1. Preserve causality of uploads, store as a graph
2. Detect long propagation chains
3. Report them as potential worm outbreaks

Spectator Architecture



Causality Propagation on Client/Server

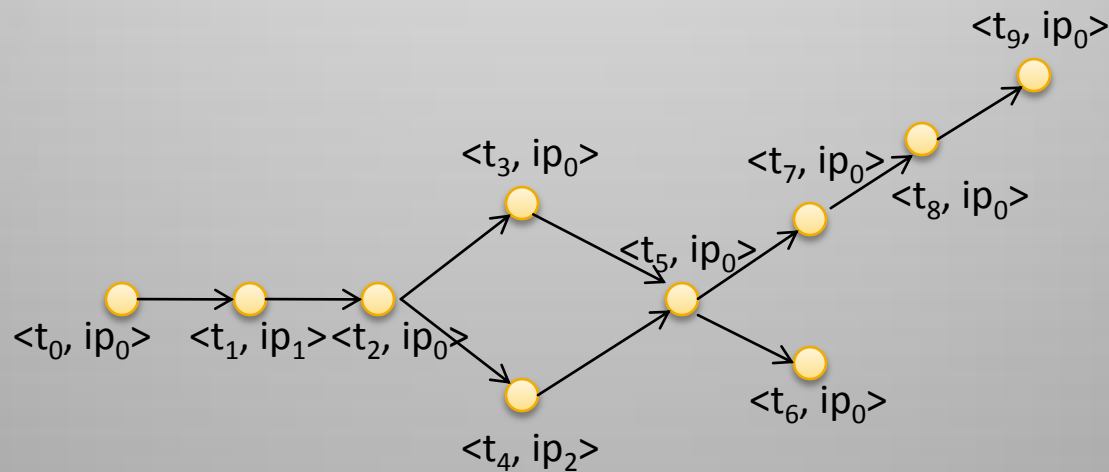
- Tagging of uploaded input

```
<div spectator_tag=56>  
  <b onclick="javascript:alert('...')">...</b>  
</div>
```

- Client-side request tracking
 - Injected JavaScript and response headers
 - Propagates causality information through cookies on the client side

Data Representation: Propagation Graph

- Propagation graph G :
 - Records causality between tags (content uploads)
 - Records IP address (approximation of user) with each
- Worm: $Diameter(G) > \text{threshold } d$



Approximation Algorithm Complexity

- Determining diameter precisely is exponential
- Scalability is crucial
 - Thousands of users
 - Millions of uploads
- Use greedy approximation of the diameter instead

	Precise algorithm	Approximate algorithm
Upload insertion time	$O(2^n)$	$O(1)$ on average
Upload insertion space	$O(n)$	$O(n)$
Worm containment time	$O(n)$	$O(n)$

Experiments

Experimental Overview

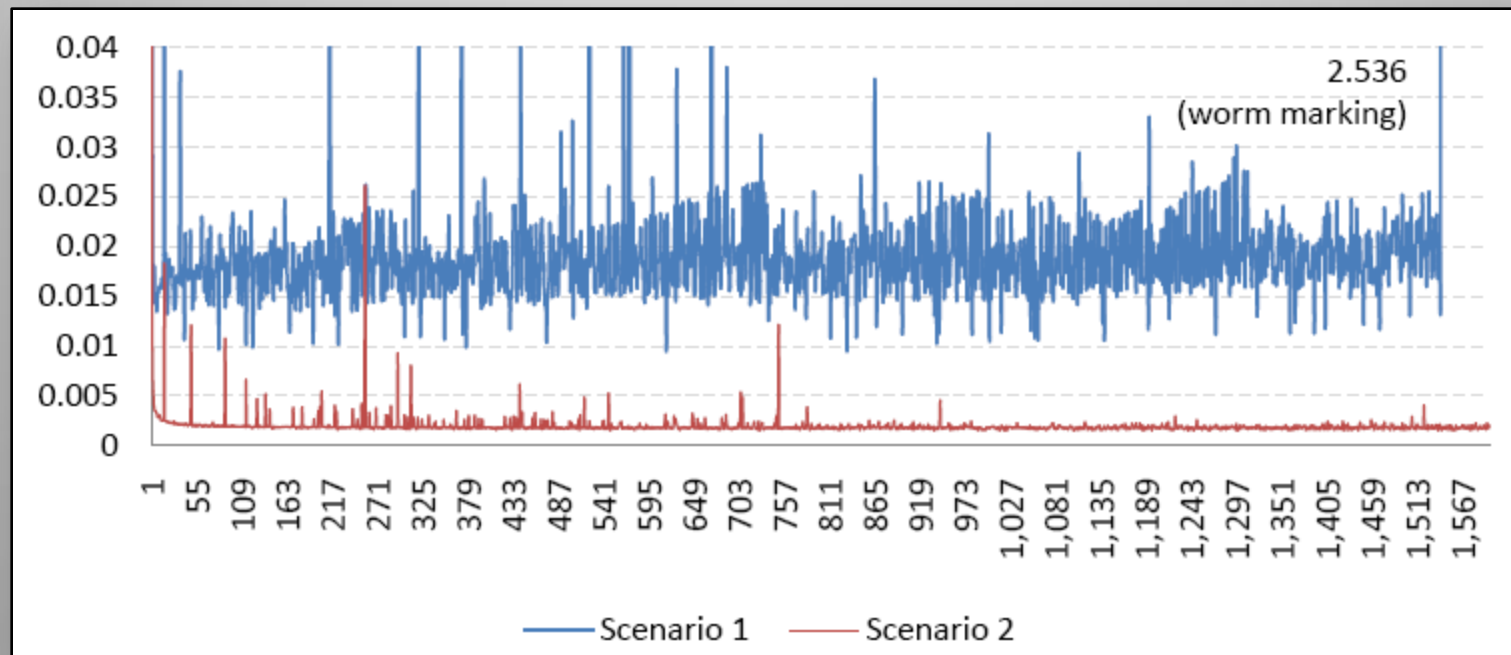
- Large-scale simulation with OurSpace:
 - Mimics a social networking site like MySpace
 - Experimented with various patterns of site access
 - Looked at the scalability
- Real-life case study:
 - Uses Siteframe, a third-party social networking app
 - Developed a JavaScript worm for it similar to real-life ones

OurSpace: Large-Scale Simulations

- Test-bed: OurSpace
 - Every user has their own page
 - At any point, a user can read or write to a page
 - `Write(U1, "hello"); Write(U1, Read(U2)); Write(U3, Read(U1));`
- Various access scenarios:
 - **Scenario 1:** Worm outbreak (random topology)
 - **Scenario 2:** A single long blog entry
 - **Scenario 3:** A power law model of worm propagation

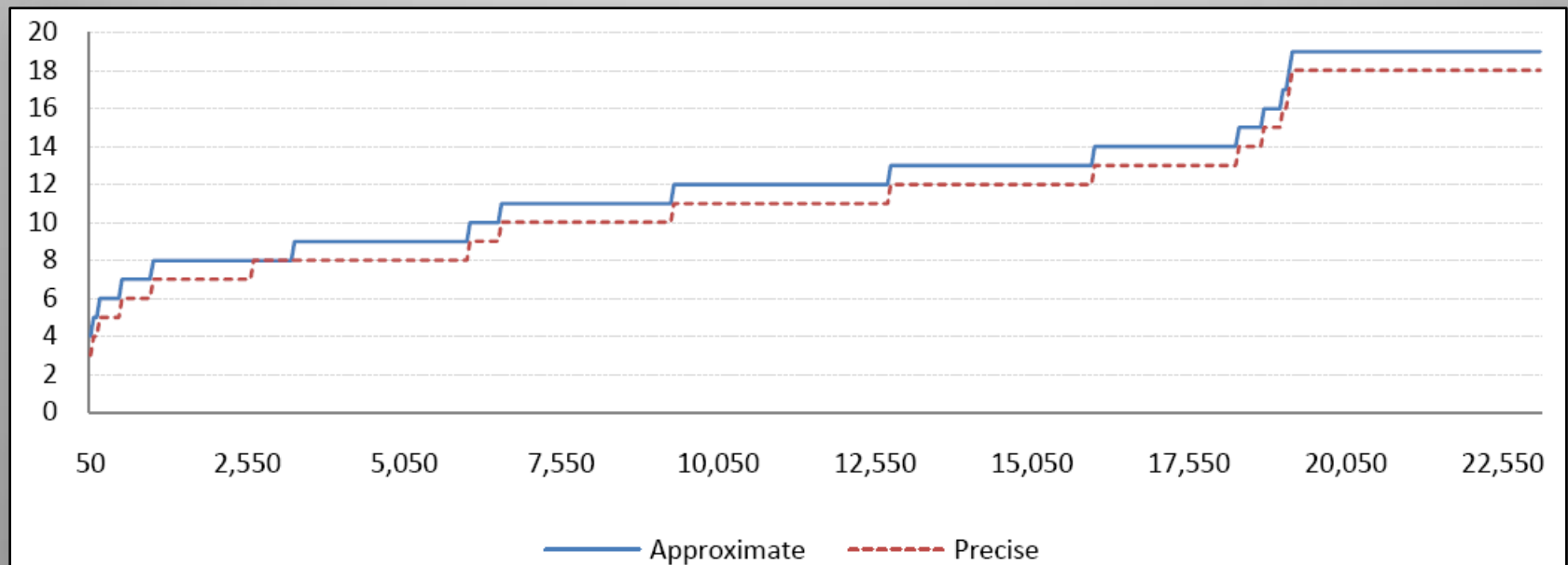
Latency of Maintaining Propagation Graph

- Tag addition overhead pretty much constant



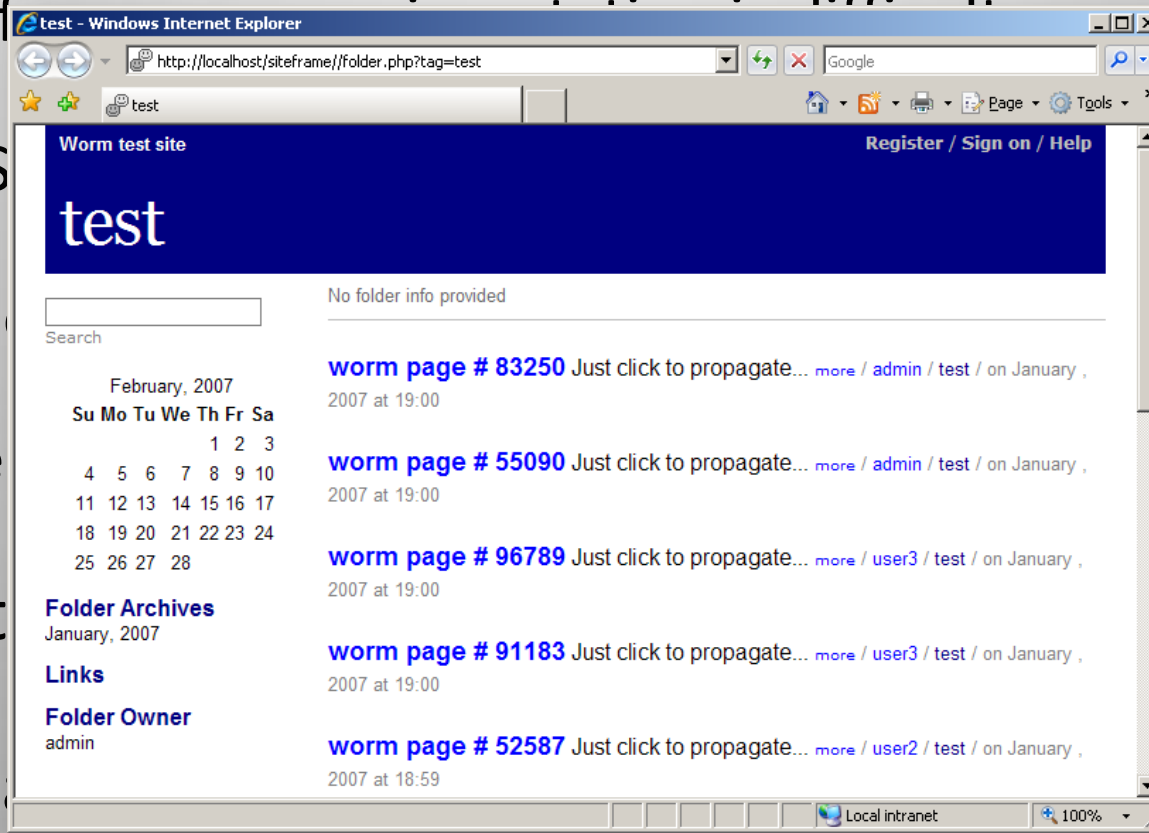
Approximation of Graph Diameter

- Approximate worm detection works well



Siteframe Experiment

- Real-life
- Used S
- Found
- Deve
- Script
- Spect



Conclusions

- First defense against JavaScript worms
 - Fast and slow, mono- and polymorphic worms
 - Scales well with low overhead
- Essence of the approach
 - Perform distributed data tainting
 - Look for long propagation chains
- Demonstrated scalability and effectiveness