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

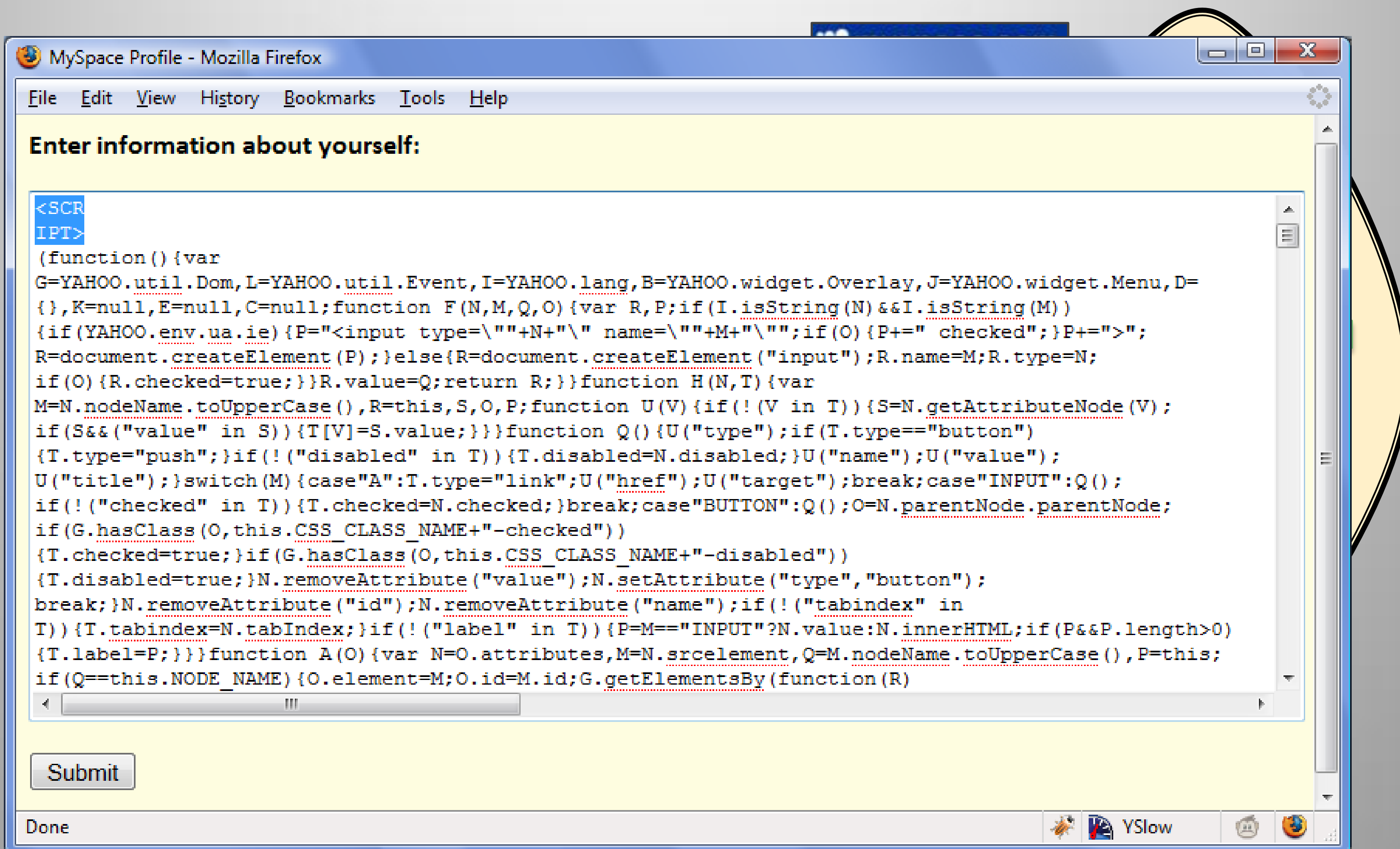
- Unleashed by Samy as a proof-of-concept in October 2005

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



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?

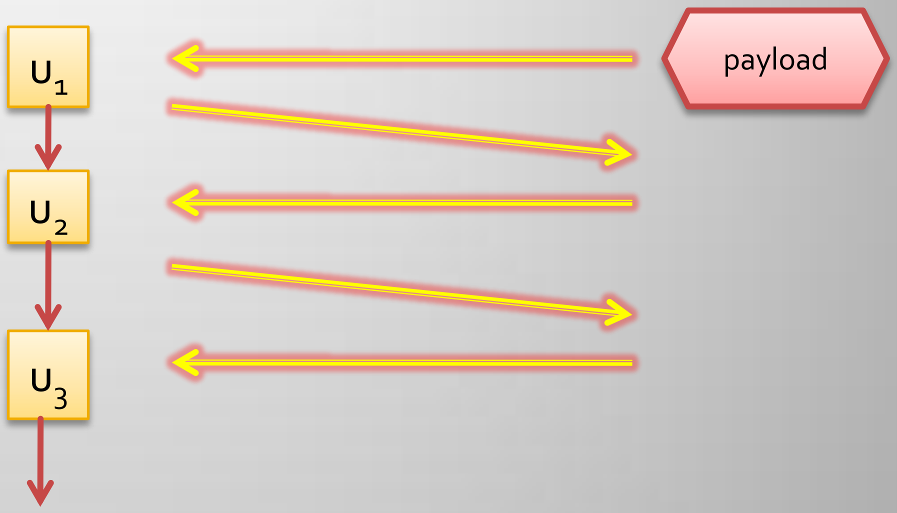
- Existing solutions rely on signatures
  - Ineffective: obfuscated and polymorphic JavaScript worms are very easy to write
  - Most real-life worms are obfuscated

- 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

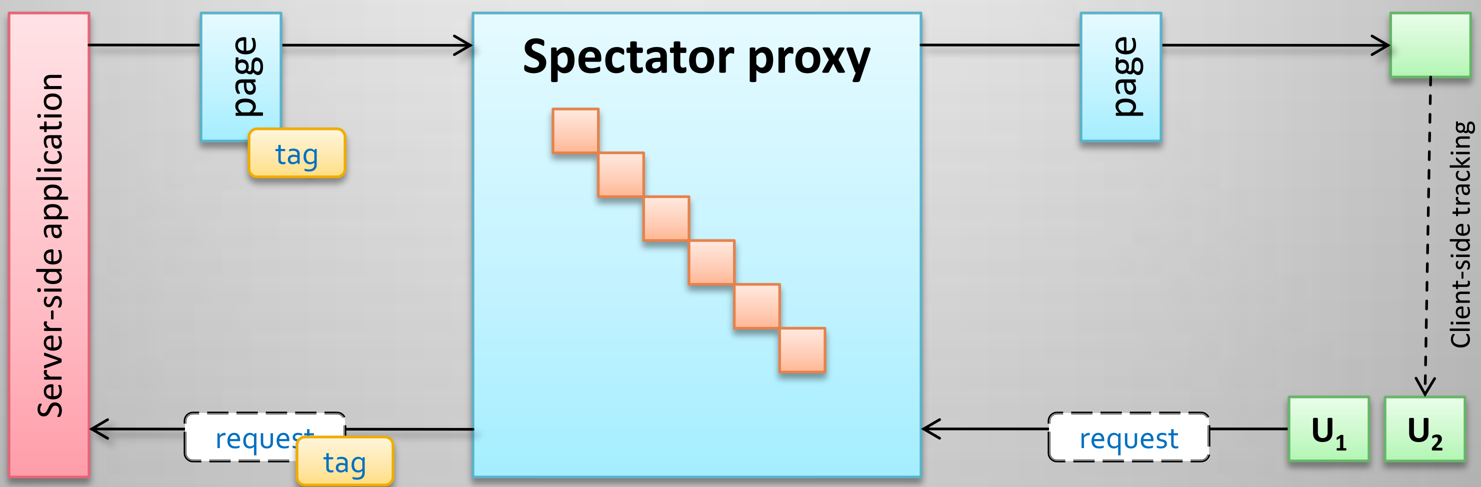
- **u_1 uploads to his page**
- u_2 downloads page of u_1
- **u_2 uploads to his page**
- u_3 downloads page of u_2
- **u_3 uploads to his page**
- …



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

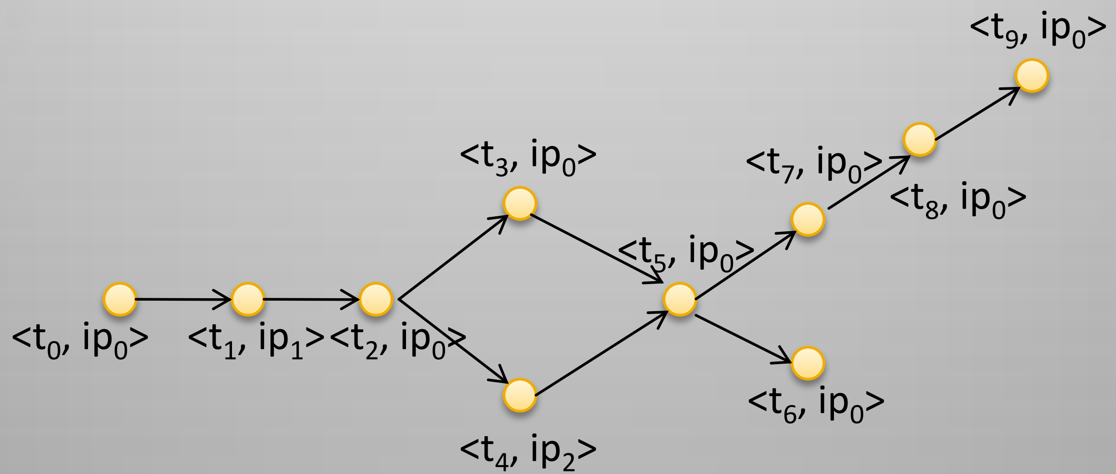
- Tagging of uploaded input

```
<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)* > 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

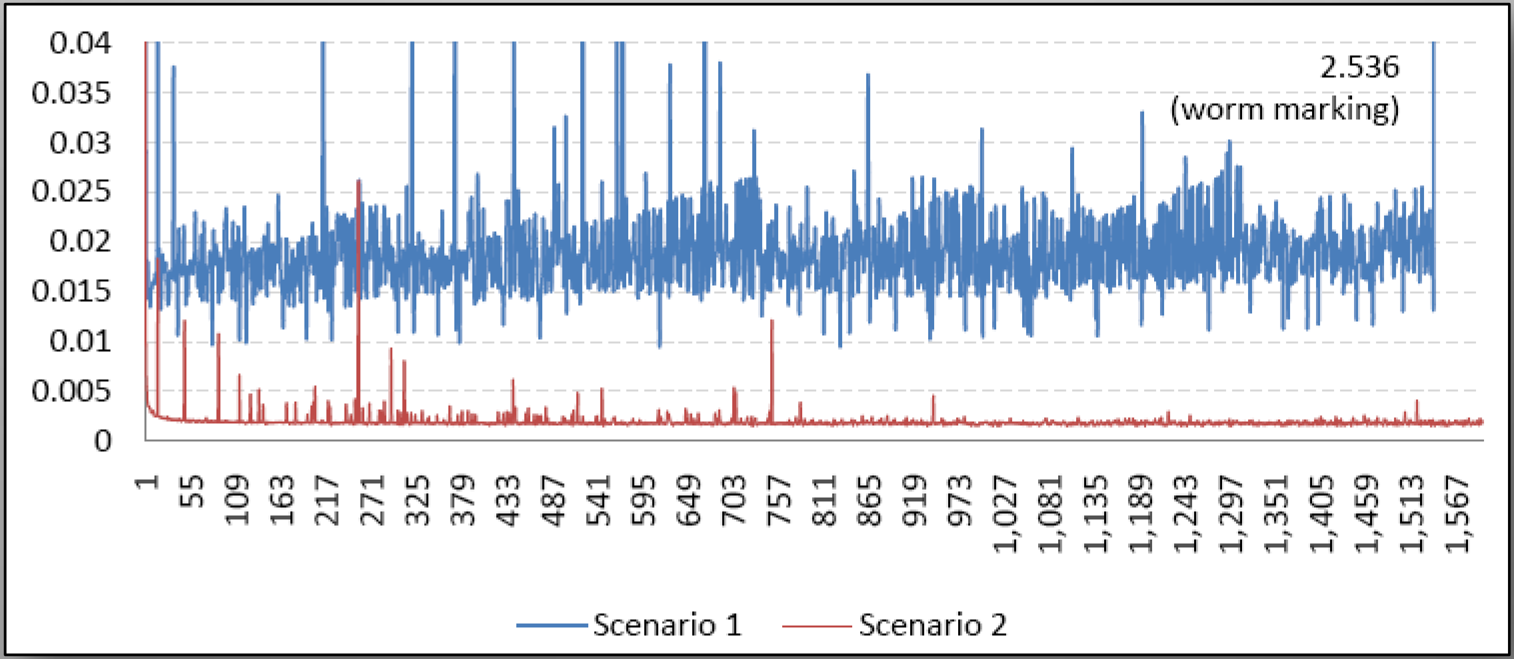
- Large-scale simulation with OurSpace:
  - Mimics a social networking site like MySpace
  - Experimented with various patterns of site access
  - Looked at the scalability
- Real-life case study:
  - Uses Siteframe, a third-party social networking app
  - Developed a JavaScript worm for it similar to 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(U_1, "hello"); Write(U_1, Read(U_2)); Write(U_3, Read(U_1));**
- 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

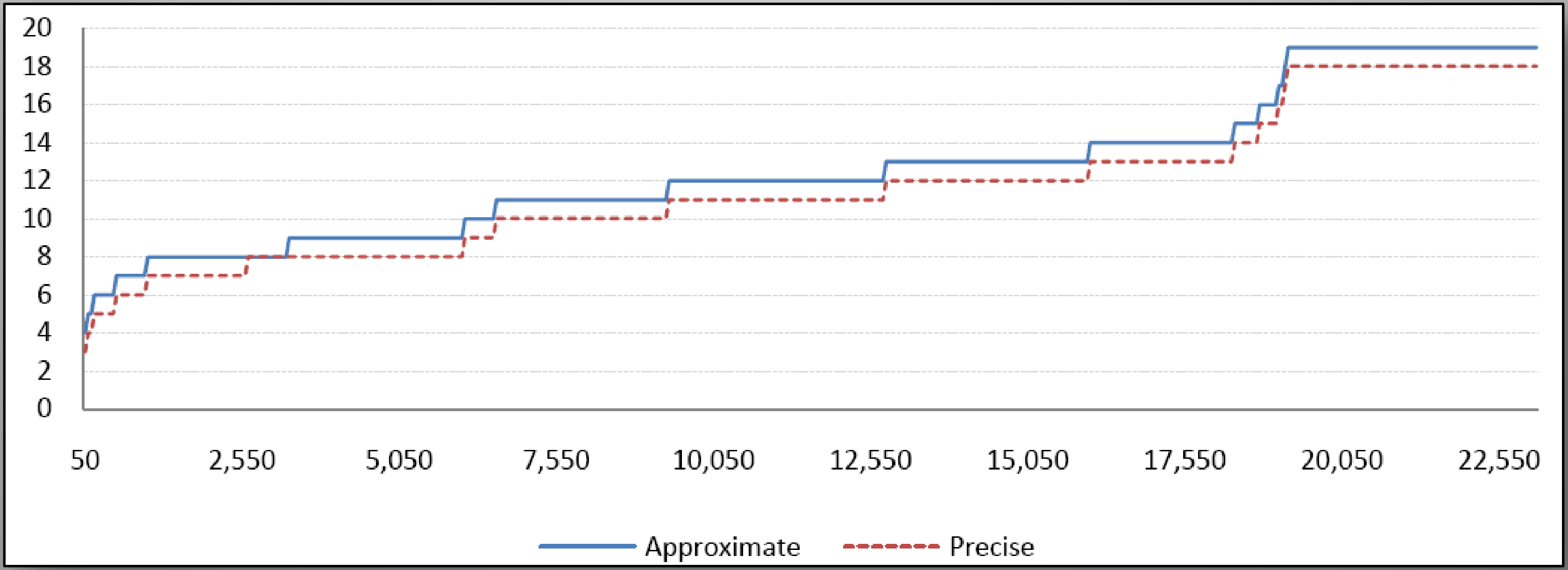
Latency of Maintaining Propagation Graph

- Tag addition overhead pretty much constant



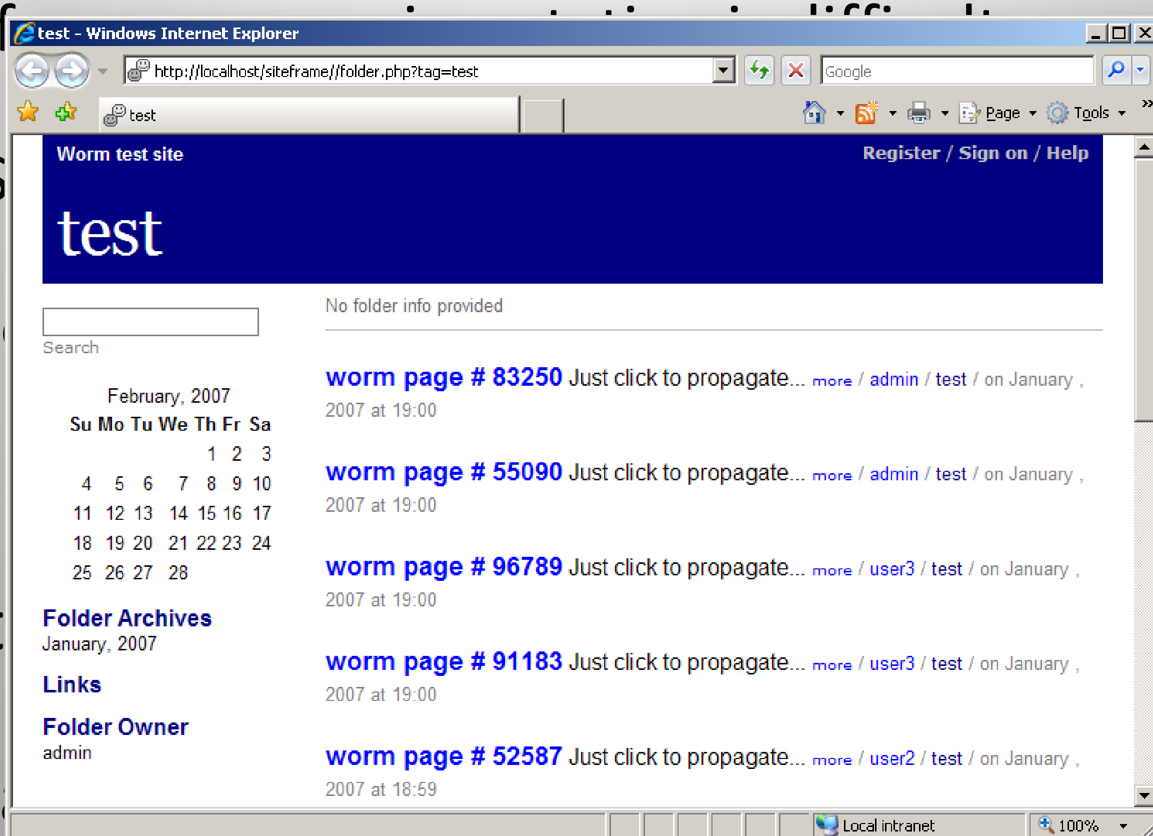
Approximation of Graph Diameter

- Approximate worm detection works well



Siteframe Experiment

- Real-lif
- Used S
  - Foun
  - 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