PathCutter: Severing the Self-Propagation Path of XSS JavaScript Worms in Social Web Networks

Yinzhi Cao§, Vinod Yegneswaran†, Phillip Porras†, and Yan Chen§

§Northwestern Lab for Internet and Security Technology, Northwestern University, Evanston, IL
†SRI International, Menlo Park, CA
• Social networks – Platforms where people share their perspectives, opinions, thoughts and experiences – OSNs, Blogs, Social bookmarking etc.

• XSS worm threat is severe.
  – More and more prevalent: Renren, Yamanner, etc.
  – Akin to virus: human need to visit infected pages
  – Characteristic: Fast spreading

• In this paper,
  – Target: Prevent XSS worm propagation
  – Method: View separation & Request authentication

Number of infected clients after 20 hours (Social Networks’ XSS Worms, Faghani et al.)
Roadmap

• Introduction
• Background
  – Attack Steps
  – XSS Taxonomy
• Related Work
• Our Approach
• Implementation
• Evaluation
Background

- Step 1 – Enticement and Exploitation
- Step 2 – Privilege Escalation
- Step 3 – Replication
- Step 4 – Propagation

Other Users → Repeat Process

Get infected

Modify benign user’s account

Download

Samy’s page

Benign User

Download

Modify benign user’s account

Get infected

Benign User

Other Users → Repeat Process
XSS Taxonomy

XSS Attacks

Server-side XSS
- Stored XSS
  - MySpace Samy Worm
  - Yamanner Worm
- Reflected XSS

Client-side XSS
- Plugin XSS
  - Flash XSS
  - Renren Worm
  - SpaceFlash Worm
- Content Sniffing XSS
  - Java XSS
- DOM-based XSS
  - Our Experimental Worm
Related Work

• Group one: Prevent XSS vulnerabilities
  – Incomplete coverage (BluePrint, Plug-in Patches, Barth et al., and Saxena et al.)

• Group two: Prevent XSS worms
  – No early-stage prevention (Spectator and Xu et al.)
  – Not resistant to polymorphic worm (Sun et al.)

• Our goal: Prevent all the XSS worms with early-stage prevention and resistance to polymorphic worms
Our Approach

- Two key concepts: (1) request authentication and (2) view separation

We use request authentication. View separation is always enforced.
For example, blog A, blog B, blog C and so on.

Or more fine-grained, different pages in the same blog.

Isolating contents from the same origin – iframe tag with sandbox properties in HTML5 – Pseudodomain encapsulation (mentioned later)
Request Authentication

• For example, requests from blog A does not have permissions to modify blog B
• Identifying which view a client-side request is from.
  – Secret token
  – Referer header
• Check if the view has the permission
Our Approach

View one does not have the permission. Identify that it is from View One. If we cannot identify, deny. Isolating views at client side.

Benign User

View One

Modify benign user’s account

Access

Download

View Two
Roadmap

• Introduction
• Background
• Related Work
• Our Approach
• Implementation
  – Implementation One (Server Modification)
  – Implementation Two (Proxy)
• Evaluation
  – Case Study of Five Real-world Worms and Two Experimental Worms (only two covered in the talk)
  – Performance
Implementation One (Server Modification)

• Prototype examples: WordPress, Elgg
• Dividing views: by blogs
• Permissions for different views: can only modify its own blog.
View Isolation for Server Modification

- Isolating views at client side.
  - Pseudodomain encapsulation.

It cannot break isolate.x.com (different origin).

Secret token is required.
Request Authentication for Server Modification

• Identifying requests from client-side
  – Secret token
    • Insertion position: Each request that will modify server-side contents.

• Checking requests’ permission
  – Checking position: Database operation. (A narrow interface that each modifying request will go through.)
Implementation Two (Proxy)

- Dividing views: by different client-side URLs.
- Permissions for different views:
  - Possible outgoing post URL from those URLs
View Isolation for Proxy

• Isolating views at client side
  – The same as implementation one.

Request content.x.com/y.php
Redirect to isolate.x.com

<iframe src="content.x.com/y.php?token=***"

Proxy

Web Server

isolate.x.com
Request Authentication for Proxy

• Identifying requests from client-side
  – Referer header
    • Specified by the browser. Attackers cannot change it.

• Checking requests’ permissions
  – Checking position: Proxy.
  – Method: See if the view has the permission to send the request.
Case Study for Real World Worms

• XSS Worm in Renren (Facebook in China)
Request to share
- Yamanner Worm

Click to send emails to all your contacts
Compose email

Different views

Email body

Send email
• Memory Overhead – Normally, the number of frames is not high since comments can be hidden.

• Rendering Time Overhead – Less than 3.5% for Elgg Evaluation
Conclusions

• We cut off the propagation path of XSS worms through view separation by psuedodomain encapsulation and request authentication.

• We implement PathCutter by proxy assistance and server modification.

• We evaluate PathCutter on 5 real-world worms and 2 proof-of-concept worms.
Thanks!
Questions?
Backup
Comparison with Existing Works

<table>
<thead>
<tr>
<th></th>
<th>Spectator</th>
<th>Sun et al.</th>
<th>Xu et al.</th>
<th>BluePrint</th>
<th>Barth et al.</th>
<th>Saxena et al.</th>
<th>PathCutter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocking Step</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Polymorphic Worm</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Early-Stage Prevention</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Types of XSS that Can Be Defended</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Deployment</td>
<td>Server or Proxy</td>
<td>Client</td>
<td>Server</td>
<td>Server</td>
<td>Client</td>
<td>Client</td>
<td>Client</td>
</tr>
<tr>
<td>Passive/Active Monitoring</td>
<td>Active</td>
<td>Passive</td>
<td>Passive</td>
<td>Active</td>
<td>Active</td>
<td>Active</td>
<td>Active</td>
</tr>
</tbody>
</table>
Limitation

• Need to know the semantics of web application
• Only prevent worm behavior but not all the damages
Existing solutions

- Spectator

But it can only detect the worm when it spreads for a while!

... if it reaches a threshold, report it.
Existing solutions

• Esorics 09

But
(1) Payload may change.
(2) Pure client-side solution.
URL graph provided by the server or a third-party

- blogX/index.php
  - blogX/post-comment.php
  - blogX/options.php
  - blogX/update-options.php
  - blogX/x.php

- blogY/index.php