

CS 152: *Programming Language Paradigms*



Dynamic Code Evaluation & Taint Analysis

Prof. Tom Austin
San José State University

Dynamic code evaluation



eval

- Executes dynamically
- Typically, eval takes a string:
`eval "puts 2+3"`
- Popular feature
 - especially in JavaScript
 - Richards et al. *The Eval that Men Do*, 2011
- Source of security problems

Parsing JSON (in-class)

Review: additional Ruby eval methods

- `instance_eval` evaluates code within the body of an object.
- `class_eval` evaluates code within the body of a class.
- These methods can take a string or (more safely) a block of code.

class_eval example
(in class)

Securing Code



The mind of a developer



Web Security in the News

6.46 million leaked online

Summary: More than 6.4 million user login details were leaked online over the weekend.

A user on LinkedIn posted a screenshot of a LinkedIn page showing a reflected cross-site scripting (XSS) attack. The post includes a link to a blog post by Eduard Kovacs detailing the exploit.

XSS vulnerabilities are being exploited by malicious actors. The expert demonstrated how a反射型跨站脚本攻击 (reflected XSS attack) can be used to compromise e-commerce giant eBay.

The expert has demonstrated how to use eBay login page to gain access to the local system.

ADVERTISING

Times Web Ads Show Security Flaws

By ASHLEE VANCE

Published: September 14, 2009

OVER the weekend, some visitors to the Web site of The New York Times received a nasty surprise. An unknown person sneaked a rogue advertisement onto the site's pages.

Home > Vulnerability

XSS Attacks

By Eduard Kovacs

in Share 38

A reflected cross-site scripting attack was exploited by malicious actors.

The flaw was identified by the expert, who compromised the e-commerce giant eBay.

XSS vulnerabilities are being exploited by malicious actors. The expert demonstrated how a反射型跨站脚本攻击 (reflected XSS attack) can be used to compromise eBay.

The expert has demonstrated how to use eBay login page to gain access to the local system.

May 21st, 2009, 08:16 GMT · By Lucian Constantin

U.S. Bank and Bank of America Websites Vulnerable

Mortgage Rates -- Select A Loan Program

<input type="checkbox"/> 40 Year Fixed	<input type="checkbox"/> 7 Year ARM	<input type="checkbox"/> 30 Year Interest Only
<input type="checkbox"/> 30 Year Fixed	<input type="checkbox"/> 5 Year ARM	<input type="checkbox"/> 5 Year Interest Only
<input type="checkbox"/> 15 Year Fixed	<input type="checkbox"/> 3 Year ARM	<input type="checkbox"/> 3 Year Interest Only
<input type="checkbox"/> 10 Year Fixed	<input type="checkbox"/> 1 Year ARM	<input type="checkbox"/> 1 Year Interest Only



SHARE: +1 0

Like Send

Tweet

Adjust text size: - +

Ads by Google

Bank Internet

Bank Owned Homes

Online Bank

US Bank

Bank of America

Cross-site scripting weaknesses have been discovered in two websites belonging to the Bank of America and U.S. Bank. The flaws facilitate potential phishing attacks, because they allow attackers to inject iFrames, hijack sessions, or prompt arbitrary alerts.

December 28th, 2011, 15:27 GMT · By Eduard Kovacs

CIA and NASA Websites Vulnerable to XSS Attacks, Hacker Proves

CIA - Intelligence Degree

Earn an intelligence degree online at American Military University.

www.AMU.APUS.edu/Intelligence

◀ ▶

SHARE: +1 0

Like Send

Tweet

AdChoices ▾

Adjust text size: - +

Ads by Google

Computer Hacker

Attack

CIA Agent

Ethical Hacker

Search Results: Hacked by D35...

Central Intelligence Agency (US) https://www.cia.gov/search?q=Hacked+by+D35M0ND142" style="background-color: green; color: white; text-align: center; font-size: 24px;">Hacked by D35M0ND142

myip

Search

Search Results

Hacked by D35M0ND142

All documents

Search

Searched

Results

Search

How do companies/developers cope?

- Train/shame developers to follow best practices.
- Hire security experts
- Use analysis tools
- Hush up mistakes
- Budget to handle emergencies
- Bury their heads in the sand.

Secure By Architecture

Developers make mistakes.



Can we design tools to
create secure systems,
despite developer mistakes?

Success story: memory-safe languages

- Buffer overflows were once ubiquitous
- Memory-safe languages manage memory automatically
 - Developer focus on functionality
 - Security-critical bugs are eliminated
- Buffer overflows have virtually disappeared
 - Except in your OS, web browser, etc.

Two Security Mechanisms

- **Taint analysis:**
 - protect critical fields from "dirty" data
- **Information flow analysis:**
 - Prevent secrets from leaking.

Taint Analysis: Protecting against dirty data



Taint analysis

- Taint analysis focuses on *integrity*:
 - does "dirty" data corrupt trusted data?
- Integrated into Perl and Ruby
- Handles *explicit flows* only
 - direct assignment
 - passing parameters

Attacks preventable by taint analysis

- Data under the control of the user may pose a security risk
 - SQL injection
 - cross-site scripting (XSS)
 - cross-site request forgery (CSRF)
- Taint tracking tracks untrusted variables and prevents them from being used in unsafe operations

Taint Tracking History

- 1989 – Perl 3 support for a taint mode
- 1996 – Netscape included support for a taint mode in server-side JavaScript
 - Later abandoned
- Ruby later implemented a taint mode; we'll review in more depth.

Taint Mode in Ruby

- Protect against integrity attacks.
 - E.g. Data pulled from an HTML form cannot be passed to eval.
- Cannot taint booleans or ints.
- Multiple ways to run in safe mode:
 - Use -T command line flag.
 - Include \$SAFE variable in code.

\$SAFE levels in Ruby

- 0 – No checking (default)
- 1
 - Tainted data cannot be passed to eval
 - Cannot load/require new files
- 2 – Can't change, make, or remove directories
- 3
 - New strings/objects are automatically tainted
 - Cannot untaint tainted values
- 4 – Safe objects become immutable

```
s = "puts 4-3".taint
$SAFE = 1 # Can't eval tainted data
s.untaint # Removes taint from data
puts s.tainted?
eval s
```

```
$SAFE = 3
s2 = "puts 2 * 7" # Tainted
s2.untaint # Won't work now
eval s2
eval s # this is OK
```

```
# Data from web
s = "Robert'); DROP TABLE " +
    "STUDENTS; --"
s.taint
exec_query("SELECT *" +
            " FROM STUDENTS" +
            " WHERE NAME=''" +
            s + "'';"
```

```
class Record
  def exec_query(query_str)
    if query_str.tainted?
      puts "Err: tainted string"
    else
      # Perform the query
      ...
    end
  end
end
```

Lab: Taint tracking

Today's lab explores taint tracking in Ruby.

Starter code is available on the course website.

Details in Canvas.