From Yogi to Smokey

Are you smarter than your average programmer?
Only **YOU** can prevent the OWASP Top 10!

Who To Blame?

Bob Rudis
a.k.a. @hrbrmstr
bob@rudis.net
http://rud.is/

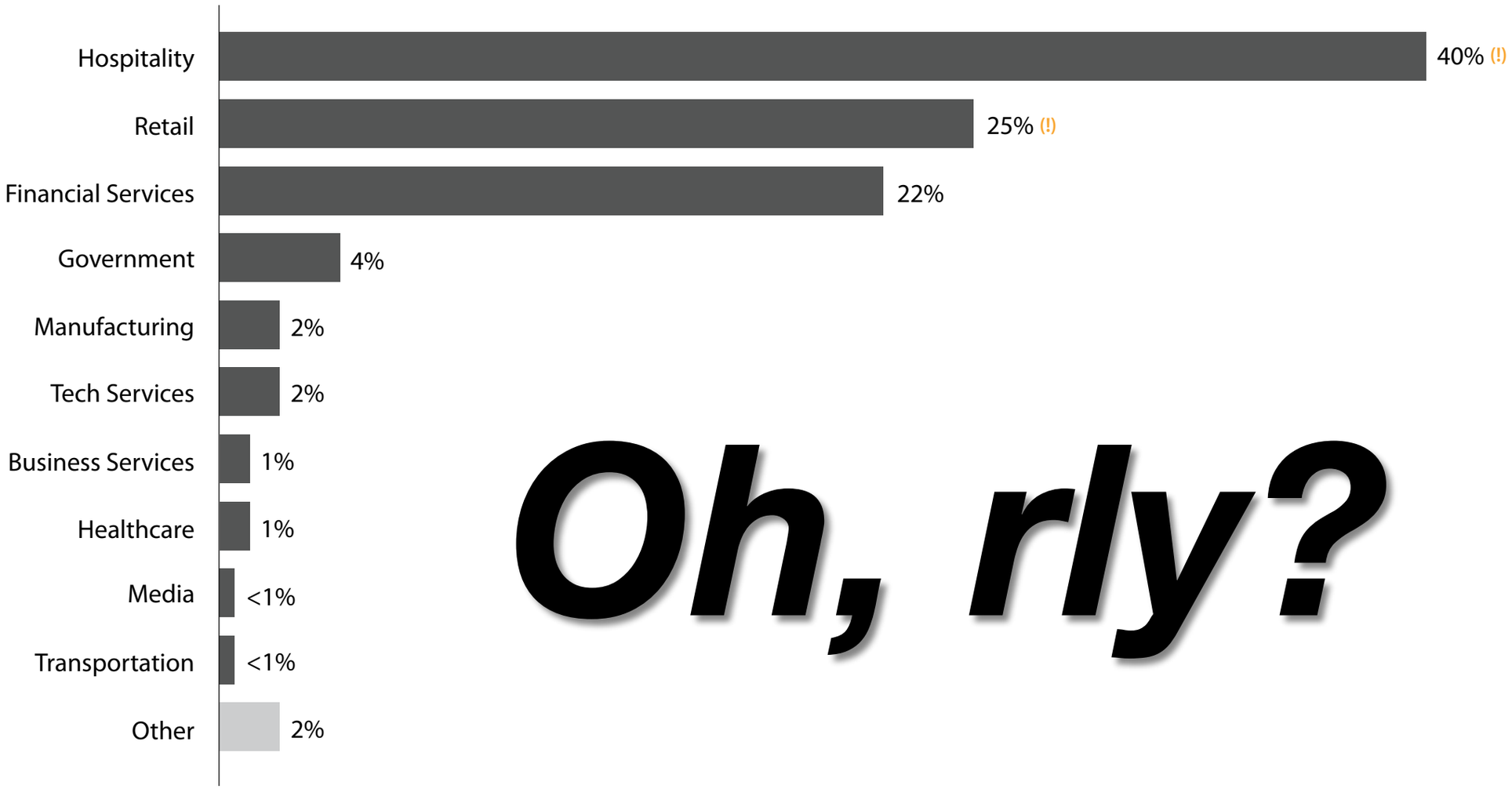
http://rud.is/nhjug

Why Care About Security?

Source: http://datalossdb.org/ | Sampling of 2011 web hack data

Not in **my** backyard!!!

Figure 3. Industry groups represented by percent of breaches



Oh, rly?

Source: Verizon 2011 Data Breach Report

But…they must be l33t new hax0rz!

Figure 22. Types of hacking by percent of breaches within Hacking and percent of records

SQL Injection	**10 Years Old & Still #1!**
Exploitation of insufficient authentication (e.g., no login required)	10% / 21%
Abuse of functionality	10% / 19%
Buffer overflow	9% / 15%
Exploitation of insufficient authorization (weak or misconfigured access control)	1% / 4%
Cross-site scripting	**Is is *that* hard to validate/escape?**
Cross-site request forgery	**Watch those headers/cookies!**
Cryptanalysis	<1% / <1%
DoS at the application layer (consumes system resources)	<1% / <1%

Source: Verizon 2011 Data Breach Report

OK! I'm convinced!

But, what can I do?

Step #1:

DON’T think SECURE

Think Rugged

http://www.ruggedsoftware.org/

http://www.owasp.org/index.php/Rugged_Software

The Rugged Software Manifesto

I am rugged... and more importantly, my code is rugged.
I recognize that software has become a foundation of our modern world.
I recognize the awesome responsibility that comes with this foundational role.
I recognize that my code will be used in ways I cannot anticipate, in ways it was not designed, and for longer than it was ever intended.
I recognize that my code will be attacked by talented and persistent adversaries who threaten our physical, economic, and national security.

I recognize these things - and I choose to be rugged.

I am rugged because I refuse to be a source of vulnerability or weakness.

I am rugged because I assure my code will support its mission.

I am rugged because my code can face these challenges and persist in spite of them.
I am rugged, not because it is easy, but because it is necessary... and I am up for the challenge.

Agile Manifesto

Customer satisfaction by rapid delivery of **useful software**

Welcome changing requirements, even late in development

Working software is delivered frequently (weeks rather than months)

Working software is the principal measure of progress

Sustainable development, able to maintain a constant pace

Close, daily co-operation between business people and developers

Face-to-face conversation is the best form of communication (co-location)

Projects are built around motivated individuals, who should be trusted

Continuous attention to technical excellence and good design

Simplicity

Self-organizing teams

Regular adaptation to changing circumstances

Step 2: Model & Implement

Threat Modeling

- Assets (what needs protecting)
- Entry Points (attack surface/vulnerabilities)
- Adversary/Threat (attack/exploit methods)
- Mitigation Strategies (countermeasure)

Threat Modeling Resources

- http://msdn.microsoft.com/en-us/library/ff648644.aspx
- http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx
- https://www.owasp.org/index.php/Threat_Risk_Modeling
- http://www.sans.org/reading_room/whitepapers/securecode/threat-modeling-process-ensure-application-security_1646
- http://video.google.com/videoplay?docid=-7341067668991160289#


Remember when writing clientside exploits was lame?
I robbed a bank through XSS the other week.
I hate myself.

Open Web Application Security Project (OWASP)

OWASP Top 10

- A1: Injection (SQL, LDAP, NoSQL, XPath)
- A2: Cross-Site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- A5: Cross-Site Request Forgery (CSRF)
- A6: Security Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards

OWASP Top 10

- **A1: Injection (SQL, LDAP, NoSQL, XPath)**
- **A2: Cross-Site Scripting (XSS)**
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- A5: Cross-Site Request Forgery (CSRF)
- A6: Security Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards

(Yes, it's **code** time now)

A1: Injection (SQL)

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
   response.setContentType("text/html;charset=UTF-8");
   PrintWriter out = response.getWriter();
   String email = request.getParameter("email");
   String password = request.getParameter("password");
   String checkbox = request.getParameter("remember_me");
   try {
      if(user.signinUser(email, password)){
         user u = new user();
         u.setEmail(email);
         u.setPassword(password);
         request.getSession().setAttribute("user", u);
         response.sendRedirect("shop.jsp");
      }
   } finally {
   }
}

public static boolean signinUser(String emailincome, String passwordincome){
   try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection c = DriverManager.getConnection("jdbc:mysql://localhost:9999/thedb", "dbuser", "dbpass");
      Statement st = c.createStatement();
      ResultSet rs =
         st.executeQuery("SELECT * FROM USERS WHERE email ='"+emailincome+"' AND password = '"+passwordincome
+"'");
      rs.next()
      return true;
   } catch (Exception e) {
   }
   return false;
}
```

A1: Injection (SQL)

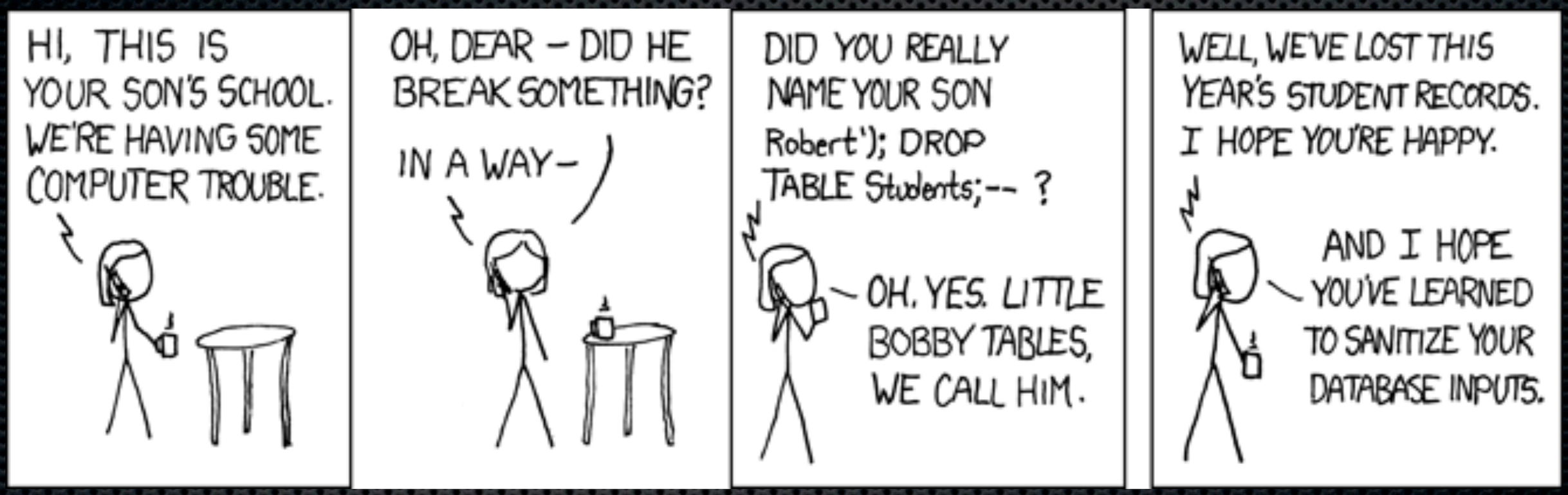
```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
   response.setContentType("text/html;charset=UTF-8");
   PrintWriter out = response.getWriter();
   String email = request.getParameter("email");
   String password = request.getParameter("password");
   String checkbox = request.getParameter("remember_me");
   try {
      if(user.signinUser(email, password)){
         user u = new user();
         u.setEmail(email);
         u.setPassword(password);
         request.getSession().setAttribute("user", u);
         response.sendRedirect("shop.jsp");
      }
   } finally {
   }
}

public static boolean signinUser(String emailincome, String passwordincome){
   try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection c = DriverManager.getConnection("jdbc:mysql://localhost:9999/thedb", "dbuser", "dbpass");
      Statement st = c.createStatement();
      ResultSet rs =
         st.executeQuery("SELECT * FROM USERS WHERE email ='"+emailincome+"' AND password = '"+passwordincome
+"'");
      rs.next()
      return true;
   } catch (Exception e) {
   }
   return false;
}
```

A1: Injection (SQL)

```
 ResultSet rs =
  st.executeQuery("SELECT *
FROM USERS WHERE email
='"+emailincome+"' AND password
= '"+passwordincome+"'");
```

SELECT * FROM USERS WHERE email=‘jon@example.com’AND password=‘quorble44’

SELECT * FROM USERS WHERE email=‘jon@example.com’ OR 1=1 --AND password=‘’


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR - DID HE BREAK SOMETHING?
IN A WAY-
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

You ***could*** attempt to sanitize/validate input parameters & use **`PreparedStatement`**s in structured queries from scratch...

That's great for **shiny**, **new** code

I suspect you work with a ton of **crufty**, **old** code

OWASP Java ESAPI

OWASP Enterprise Security API (Java Edition)

OWASP ESAPI (Java)

- Understands database nuances
- Input validation
- Output escaping
- Built-in Web Application Firewall (WAF)
- Crypto routines
- (and more)

http://code.google.com/p/owasp-esapi-java/

A1: Injection (SQL)

```
Encoder oe = new OracleEncoder();
ResultSet rs =
   st.executeQuery("SELECT * FROM
USERS WHERE email ='"+oe.encode
(emailincome)+"' AND password =
'"+oe.encode(passwordincome)
+"'");
```

You just made your code more R[illegible]ng only

55 [illegible]racters*

*(& a few `CR/LFs` & `imports`)

Actually, two

Least Privilege

- Minimize the privileges assigned to every database account in your environment
- Do not assign DBA or admin type access rights to your application account
- Make sure that accounts that only need read access are only granted read access to the tables they need access to
- Use views that limit access to portions of data and assign account access to those views instead

Least Privilege

```
GRANT SELECT ON database.* TO
normaluser@localhost IDENTIFIED BY 'yyy';

GRANT SELECT, UPDATE, INSERT, DELETE ON
database.* TO adminuser@localhost
IDENTIFIED BY 'xxx';
```


RCUM
U.S. DEPARTMEN
HOME ND SEC

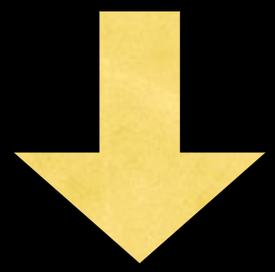
Whitelist Input Validation

- Defining exactly what IS authorized
- Easiest for structured data (zip codes, dates, e-mail)
- ESAPI can help:

```
getValidDate()          getValidNumber()
getValidCreditCard()    getValidFileName()
getValidSafeHTML()      getValidRedirectLocation()
getValidInput()         (and more)
```

A1: Injection (SQL) – Whitelist Input Validation

```
1 public void doPost( HttpServletRequest request, HttpServletResponse response) {
2     try {
3         String zipCode = request.getParameter( "zip" );
4         .. do what you want with ‘zipCode’ param here ..
5     } catch( ValidationException e ) {
6         response.sendError( response.SC_BAD_REQUEST, e.getMessage() );
7     }
8 }
```



```
1 public void doPost( HttpServletRequest request, HttpServletResponse response) {
2     try {
3         String zipCode = Validator.getValidInput("ChangeAddressPage_ZipCodeField",
4           request.getParameter( "zip" ), "zipCodePattern", 10, false));
5         .. do what you want with validated ‘zipCode’ param here ..
6     } catch( ValidationException e ) {
7         response.sendError( response.SC_BAD_REQUEST, e.getMessage() );
8     }
9 }
```

Validator.getValidInput ("ChangeAddressPage_ZipCodeField", request.getParameter("zip"), **"zipCodePattern",** 10, false));

A1: Injection (SQL) – Whitelist Input Validation

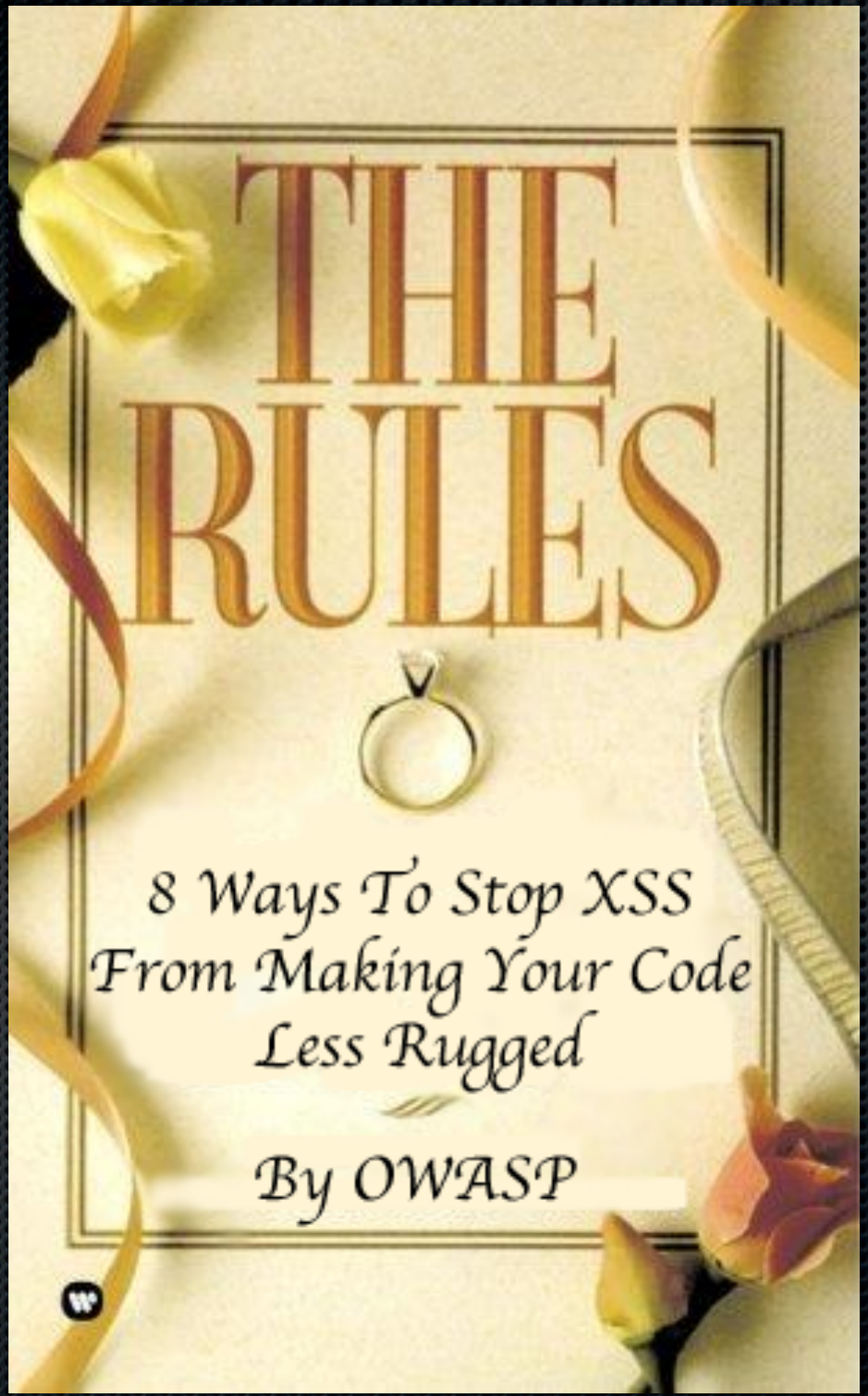
```
Validator.SafeString=^[\p{L}\p{N}.]{0,1024}$
Validator.Email=^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\\.[a-zA-Z]{2,4}$
Validator.IPAddress=^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2
[0-4][0-9]|[01]?[0-9][0-9]?)$
Validator.URL=^(ht|f)tp(s?)\\:\\/\\/[0-9a-zA-Z]([-.\\w]*[0-9a-zA-Z])*(:(0-9)*)*(\\/?)
([a-zA-Z0-9\\-\\.\\?\\,\\:\\'\\/\\\\\\+=&amp;%\\$#_]*)?$
Validator.CreditCard=^(\\d{4}[- ]?){3}\\d{4}$
Validator.SSN=^(?!000)([0-6]\\d{2}|7([0-6]\\d|7[012]))([ -]?)(?!00)\\d\\d\\3(?!0000)\\d
{4}$

Validator.HTTPScheme=^(http|https)$
Validator.HTTPServerName=^[a-zA-Z0-9_.\\-]$
Validator.HTTPParameterName=^[a-zA-Z0-9_]{0,32}$
Validator.HTTPParameterValue=^[a-zA-Z0-9.\\-\\/+=_ ]*$
Validator.HTTPCookieName=^[a-zA-Z0-9\\-_]{0,32}$
Validator.HTTPCookieValue=^[a-zA-Z0-9\\-\\/+=_ ]*$
Validator.HTTPHeaderName=^[a-zA-Z0-9\\-_]{0,32}$
Validator.HTTPHeaderValue=^[a-zA-Z0-9()\\-=\\*\\.\\?;,+\\/:&_ ]*$
Validator.HTTPContextPath=^[a-zA-Z0-9.\\-_]$
Validator.HTTPPath=^[a-zA-Z0-9.\\-_]$
Validator.HTTPQueryString=^[a-zA-Z0-9()\\-=\\*\\.\\?;,+\\/:&_ ]$
Validator.HTTPURI=^[a-zA-Z0-9()\\-=\\*\\.\\?;,+\\/:&_ ]$
Validator.HTTPURL=^.*$
Validator.HTTPJSESSIONID=^[A-Z0-9]{10,30}$
```

Validation helps with

A2: Cross-Site Scriptinig

(XSS)

too, **IF** you also follow


THE RULES
8 Ways To Stop XSS From Making Your Code Less Rugged
By OWASP

XSS In A Nutshell

- **Stored** (think, whacked blog comment)
- **Reflected** (think, whacked URL in e-mail)
- **DOM** (client-side, think jQuery whacking DOM elements before AJAX calls)

Since you saw input validation already *(it's important for XSS, too)*, let's focus on cleaning **output**

(See, even kids can do it)

XSS Prevention Rules

- RULE #0 - Never Insert Untrusted Data Except in Allowed Locations

```
<script>NEVER PUT UNTRUSTED DATA HERE</script>
directly in a script

<!--NEVER PUT UNTRUSTED DATA HERE-->
inside an HTML comment

<div NEVER PUT UNTRUSTED DATA HERE=test />
in an attribute name

<NEVER PUT UNTRUSTED DATA HERE href="/test" />
in a tag name
```

XSS Prevention Rules

- RULE #1 - HTML Escape Before Inserting Untrusted Data into HTML Element Content

```
<body>ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE</body>

<div>ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE</div>
```

```
String safe = ESAPI.encoder().encodeForHTML
    ( request.getParameter( "input" ) );
```

XSS Prevention Rules

- RULE #2 - Attribute Escape Before Inserting Untrusted Data into HTML Common Attributes

```
<div attr=ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE>content</div>
inside UNquoted attribute

<div attr='ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE'>content</div>
inside single quoted attribute

<div attr="ESCAPE UNTRUSTED DATA BEFORE PUTTING
HERE">content</div>
inside double quoted attribute
```

```
String safe = ESAPI.encoder
().encodeForHTMLAttribute( request.getParameter
( "input" ) );
```

XSS Prevention Rules

- RULE #3 - JavaScript Escape Before Inserting Untrusted Data into HTML JavaScript Data Values

```
<script>alert('ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE')
                                    </script>
                        inside a quoted string

<script>x='ESCAPE UNTRUSTED DATA BEFORE PUTTING HERE'</script>
                        one side of a quoted expression

<div onmouseover="x='ESCAPE UNTRUSTED DATA BEFORE PUTTING
                                    HERE'"</div>
                        inside quoted event handler
```

```
String safe = ESAPI.encoder().encodeForJavaScript
        ( request.getParameter( "input" ) );
```

XSS Prevention Rules

- RULE #4 - CSS Escape Before Inserting Untrusted Data into HTML Style Property Values

```
<style>selector { property : ESCAPE UNTRUSTED DATA BEFORE
                  PUTTING HERE; } </style>
                        property value

<style>selector { property : "ESCAPE UNTRUSTED DATA BEFORE
                  PUTTING HERE"; } </style>
                        property value

<span style="property : ESCAPE UNTRUSTED DATA BEFORE PUTTING
                  HERE">text</style>
                        property value
```

```
String safe = ESAPI.encoder().encodeForCSS
    ( request.getParameter( "input" ) );
```

XSS Prevention Rules

- RULE #5 - URL Escape Before Inserting Untrusted Data into HTML URL Parameter Values

```
<a href="http://www.example.com?test=ESCAPE UNTRUSTED DATA
BEFORE PUTTING HERE">link</a>
```

```
String safe = ESAPI.encoder().encodeForURL
    ( request.getParameter( "input" ) );
```

```
String userURL = request.getParameter( "userURL" );
boolean isValidURL = ESAPI.validator().isValidInput
("URLContext", userURL, "URL", 255, false);
if (isValidURL) {
    <a href="<%=encoder.encodeForHTMLAttribute(userURL)
%>">link</a>
}
```

encodeForBase64(byte[] input, boolean wrap)
Encode for Base64.

encodeForCSS(java.lang.String input)
Encode data for use in Cascading Style Sheets (CSS) content.

encodeForDN(java.lang.String input)
Encode data for use in an LDAP distinguished name.

encodeForHTML(java.lang.String input)
Encode data for use in HTML using HTML entity encoding

encodeForHTMLAttribute(java.lang.String input)
Encode data for use in HTML attributes.

encodeForJavaScript(java.lang.String input)
Encode data for insertion inside a data value or function argument in JavaScript.

encodeForLDAP(java.lang.String input)
Encode data for use in LDAP queries.

encodeForOS(Codec codec, java.lang.String input)
Encode for an operating system command shell according to the selected codec (appropriate codecs include the WindowsCodec and UnixCodec).

encodeForSQL(Codec codec, java.lang.String input)
Encode input for use in a SQL query, according to the selected codec (appropriate codecs include the MySQLCodec and OracleCodec).

encodeForURL(java.lang.String input)
Encode for use in a URL.

encodeForVBScript(java.lang.String input)
Encode data for insertion inside a data value in a Visual Basic script.

encodeForXML(java.lang.String input)
Encode data for use in an XML element.

encodeForXMLAttribute(java.lang.String input)
Encode data for use in an XML attribute.

encodeForXPath(java.lang.String input)
Encode data for use in an XPath query.

XSS Prevention Rules

- RULE #6 - Use an HTML Policy engine to validate or clean user-driven HTML in an outbound way
- RULE #7 - Prevent DOM-based XSS

http://www.owasp.org/images/e/e9/OWASP-WASCAppSec2007SanJose_AntiSamy.ppt

HTTPOnly

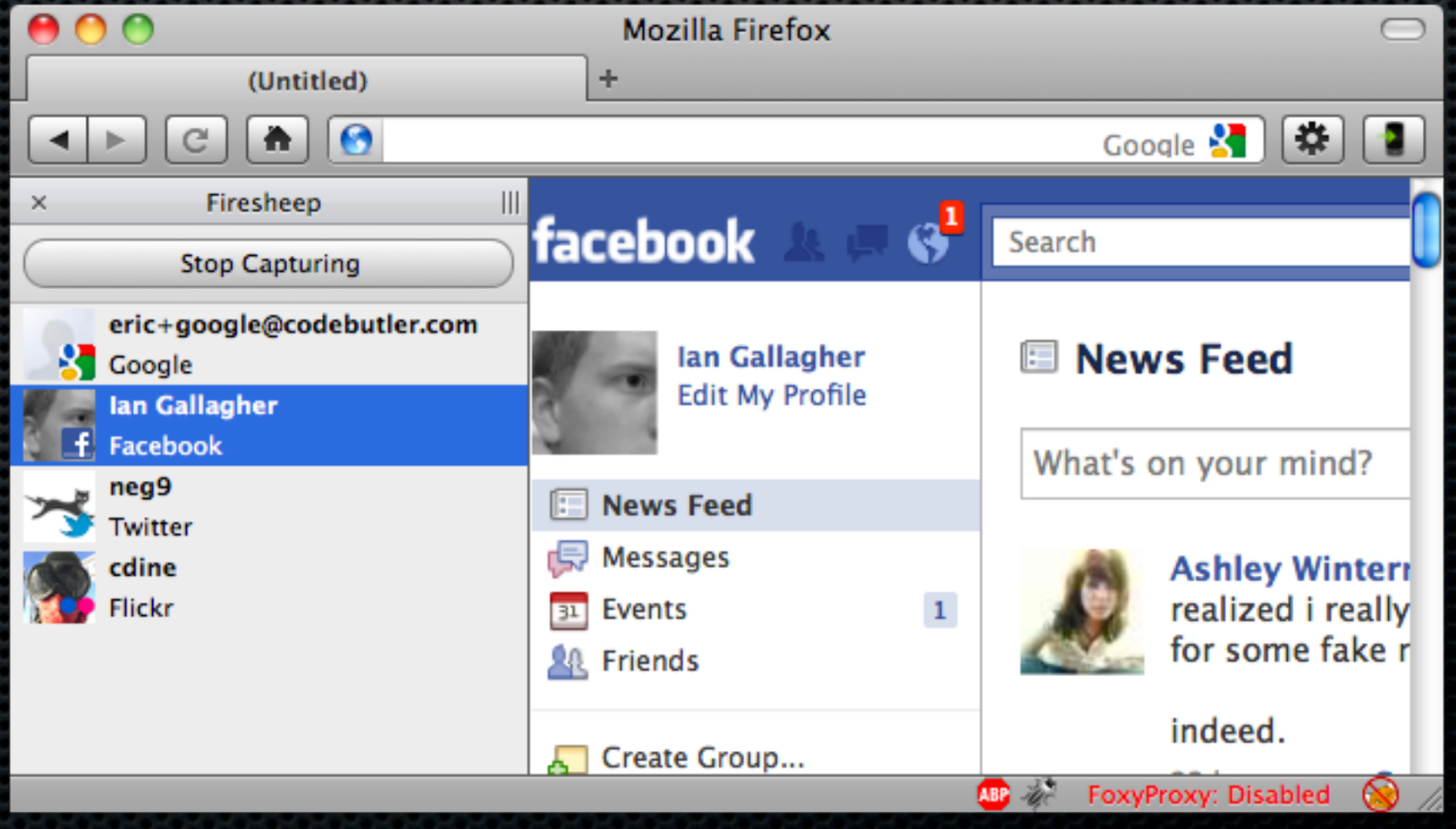
Use if a Cookie should not be accessible on the client

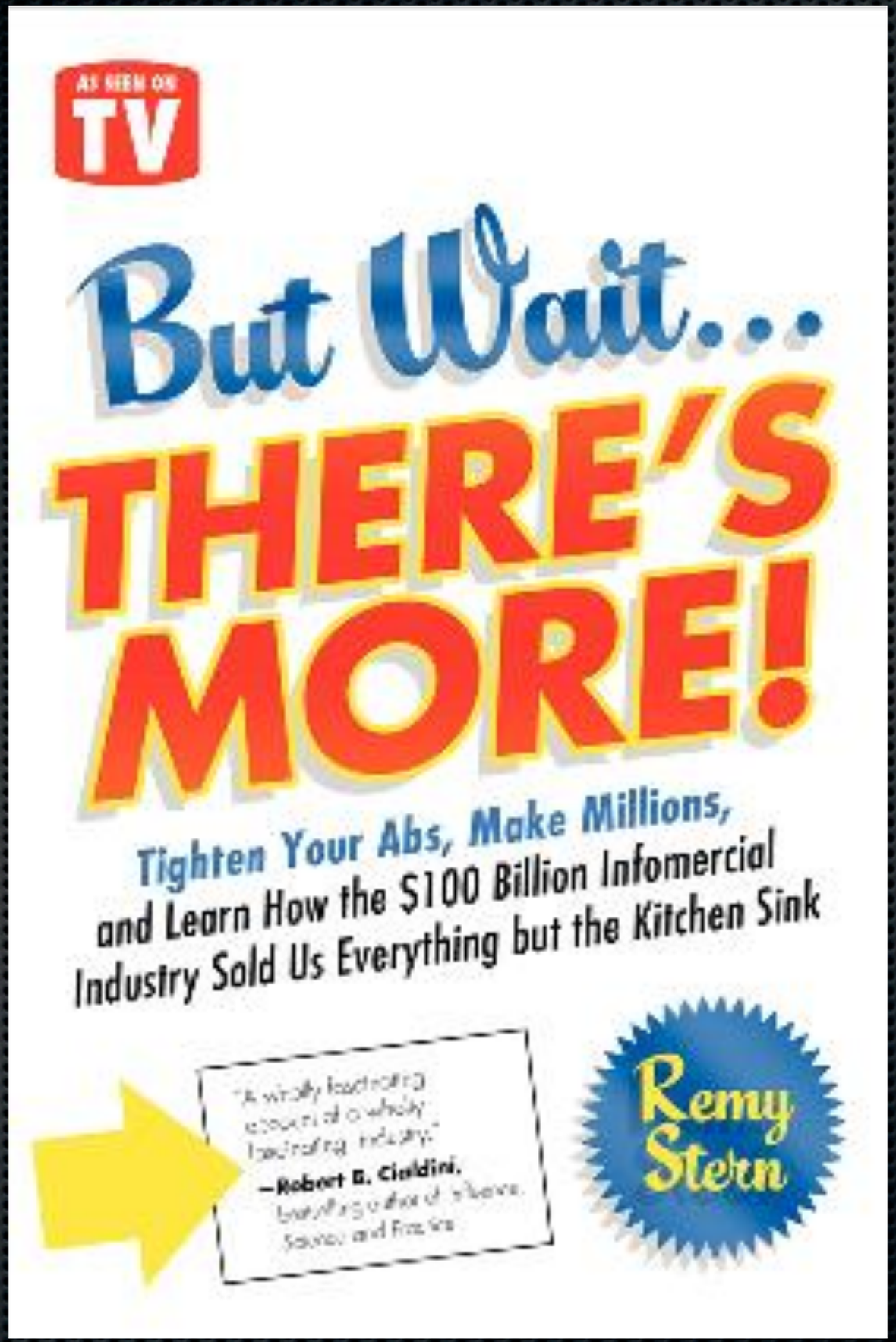
If a browser detects a cookie containing the HttpOnly flag, and client side script code attempts to read the cookie, the browser returns an empty string as the result.

```
String sessionid = request.getSession().getId();
response.setHeader("SET-COOKIE", "JSESSIONID=" +
                   sessionid + "; HttpOnly");
```

And, if you use **session cookies** to let users switch between security levels within your web app

STOP IT!




AS SEEN ON TV
But Wait...
THERE'S MORE!
Tighten Your Abs, Make Millions,
and Learn How the $100 Billion Infomercial
Industry Sold Us Everything but the Kitchen Sink
—Robert B. Cialdini,
Remy Stern

Filters For Free

```
org.owasp.esapi.filters.ClickjackFilter
org.owasp.esapi.filters.RequestRateThrottleFilter
org.owasp.esapi.filters.SecurityWrapper
org.owasp.esapi.waf.ESAPIWebApplicationFirewallFilter
```

Rugged Tools :: Static Code Analysis

- FindBugs — http://findbugs.sourceforge.net/
- $ Klocwork — http://www.klocwork.com/products/solo/?source=feature
- $ AppScan — http://www-01.ibm.com/software/awdtools/appscan/
- $ Fortify — https://www.fortify.com/
- PMD — http://pmd.sourceforge.net/
- $ Coverity — http://www.coverity.com/products/index.html
- **Free & $** Veracode — http://www.veracode.com/

Rugged Tools :: Vulnerability Scanners

- Nessus – http://www.tenable.com/products/nessus
- Wapiti – http://wapiti.sourceforge.net/
- Nikto 2 — http://www.cirt.net/nikto2
- Burp Suite – http://portswigger.net/burp/
- $ Retina — http://www.eeye.com/Products/Retina/Web-Security-Scanner.aspx
- $ QualysGuard – http://www.qualys.com/products/qg_suite/was/

Rugged Tools :: WAFs

- mod_security — http://www.modsecurity.org/
- IronBee – https://www.ironbee.com/
- $ Hyperguard – http://www.artofdefence.com/en/products/hyperguard.html
- $ F5 ASM – http://www.f5.com/products/big-ip/application-security-manager.html.html
- $ Imperva – http://www.imperva.com/index.html
- WebKnight – http://www.aqtronix.com/?PageID=99

Rugged Tools :: SQLi

- sqlmap – http://sqlmap.sourceforge.net/
- sqlninja – http://sqlninja.sourceforge.net/
- **Free & $** Havij — http://itsecteam.com/en/projects/project1.htm

Rugged Tools :: General

- nmap – http://nmap.org/
- netstat – Win: http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/netstat.mspx?mfr=true
- netstat – Other: http://linux.die.net/man/8/netstat

Rugged Tools :: Frameworks

- JAAS – http://download.oracle.com/javase/6/docs/technotes/guides/security/jaas/tutorials/index.html (Watch out under load, tho)
- Shiro – http://shiro.apache.org/
- ESAPI – http://code.google.com/p/owasp-esapi-java/
- Spring Security – http://static.springsource.org/spring-security/site/
- Hibernate – http://www.hibernate.org/quick-start

Rugged Tools :: Framework ?s

- Is it under active development?
- Are there a good number of contributors?
- When was the last commit?
- What does the community (e.g. Stack Exchange) think?
- Full OWASP Top 10 Coverage?
- Any major vulnerabilities in the framework in last year?

About Those Frameworks


```
sha1sum source.tgz
md5sum source.tgz
```

Maven can do this for you!

About Those Frameworks

Attacking the Build Through Cross-Build Injection

https://www.fortify.com/downloads2/public/fortify_attacking_the_build.pdf

About Those Frameworks

- Use trusted sources (validate DNS, SSL, IP)
- Create internal repository from known, good start
- Monitor for security vulnerabilities & patch/integrate
- Maintain a complete inventory of all your external dependencies
- Perform static code analysis (internal or third party)

You Are Not Alone

- Static Code Analysis
- Code Reviews
- Vulnerability Scanning
- Web Application Firewall
- Secure Frameworks
- Traditional Firewall
- Logging & Alerting
- Configuration Security
- Server Hardening
- Server, Java & Framework Patching

Training & Certification


CSSLP
Certified Secure Software Lifecycle Professional

SANS Certifications

- GIAC Secure Software Programmer — Java (GSSP)
- GIAC Web Application Defender (GWEB)
- GIAC Web Application Penetration Tester (GWAPT)

http://software-security.sans.org/certification
http://www.giac.org/certification/gssp-java

Free Online Training

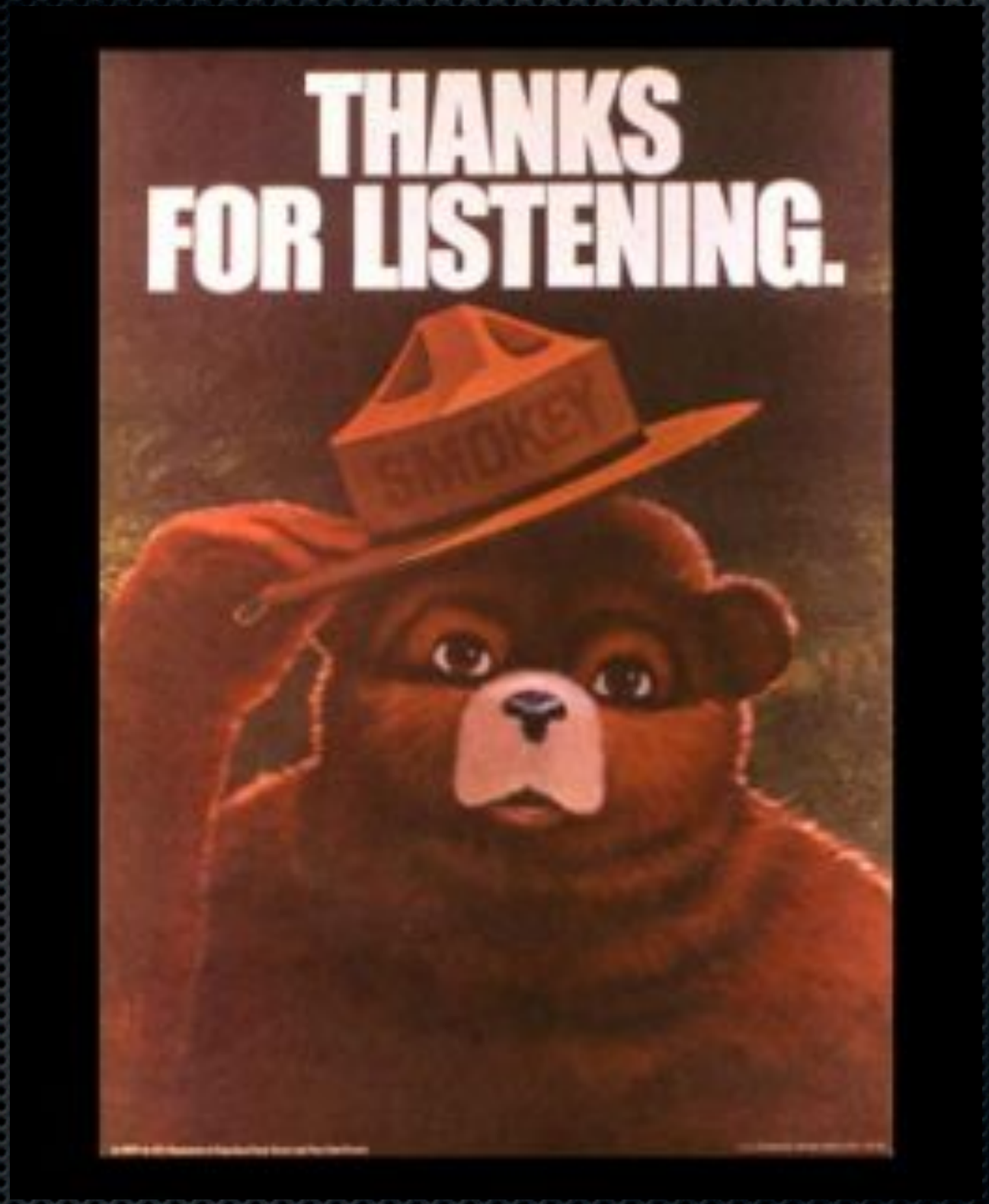
- http://l.rud.is/owasptraining (Links to Security Innovation courses):
  - OWASP Top Ten - Threats & Mitigations
  - Introduction to the Microsoft SDL
  - Introduction to Cross-Site Scripting

Practice

- Gruyere
  http://gruyere.appspot.com/
- DVWA
  http://sourceforge.net/projects/dvwa/
- WackoPicko
  https://github.com/adamdoupe/WackoPicko

Cool Hacker Tools

You
are
Rugged


THANKS
FOR LISTENING.
SMOKEY