Auditing Web Applications

IT Audit strategies for web applications

August 2015
ISACA Geek Week
Robert Morella
MBA, CISA, CGEIT, CISSP
Rob.morella@gmail.com

About Me

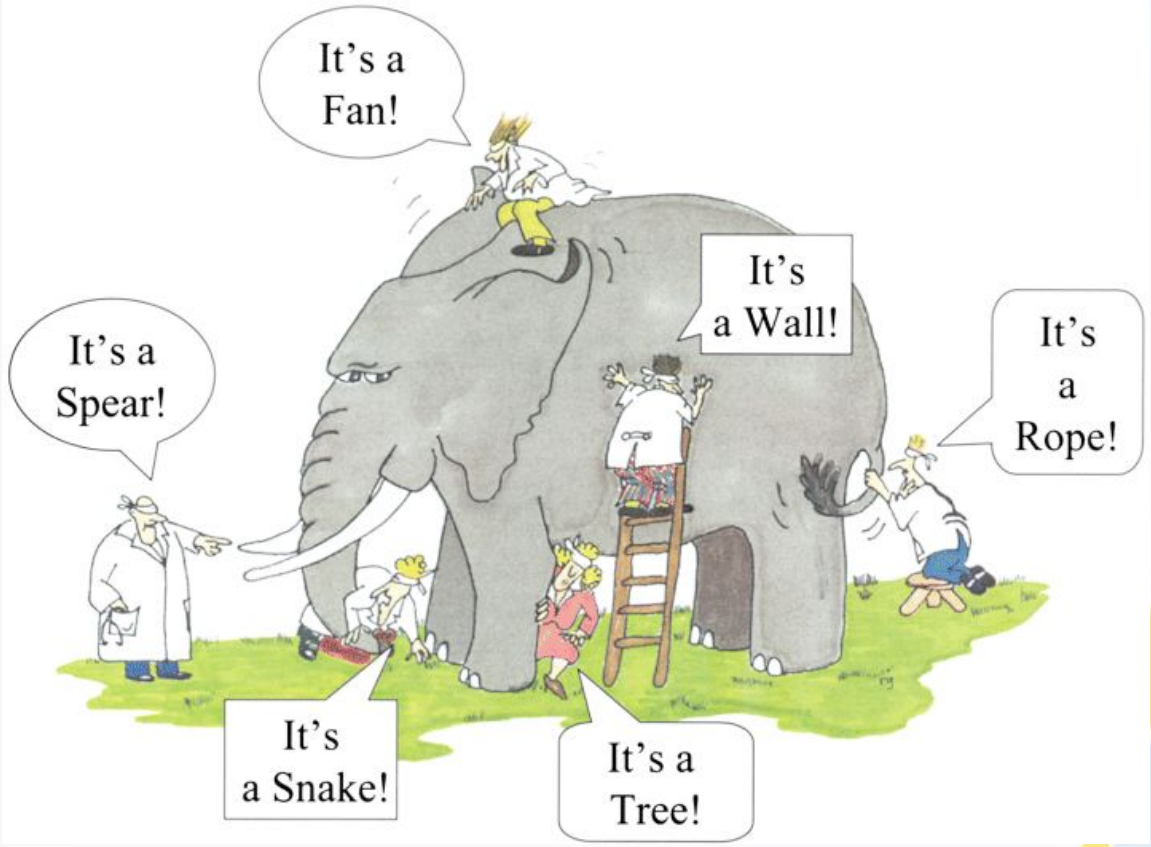
- ✓ Been there done that:
- ✓ IT Systems
- ✓ IT Architecture
- ✓ Governance/Security
- ✓ IT Audit
- ✓ Cybercrime Investigator
- ✓ ISACA QAT, CISA Boot Camp
- ✓ Geek
- ✓ In 2014 reported security breach on TV
- ✓ Helped to Pass SB386 Privacy Law

Agenda

- Definition
  - Web applications vs Web Sites
  - Risks of each type
  - Informational/Operational & Internal/External
- Risks
  - Security Risks of Web Apps
  - Audit Risks of Web Apps
  - OWASP Top Ten
- Techniques
  - Review Web Site/App Quickly
  - Demonstrate some useful tools

Definition

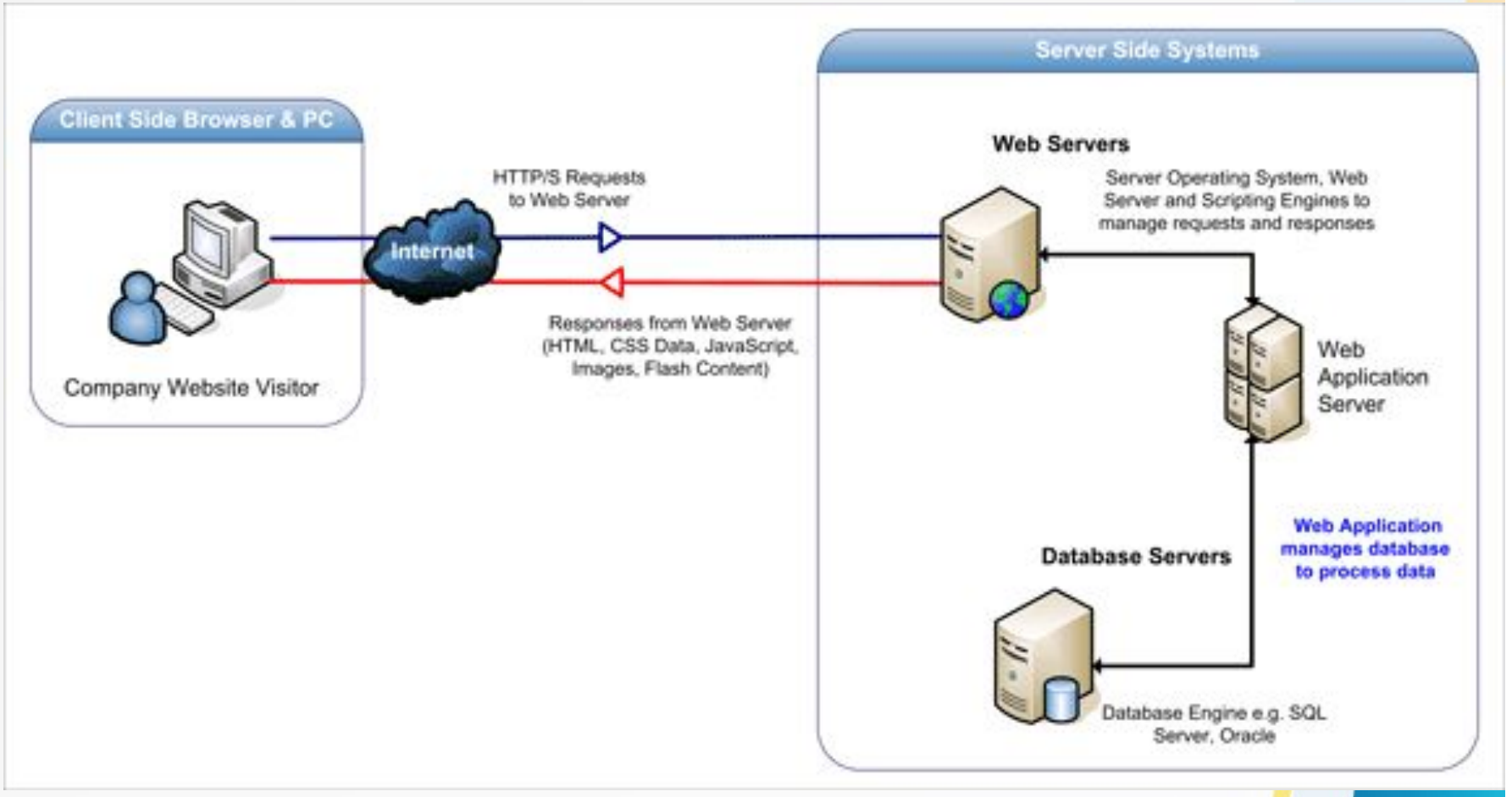
- What is a Web Application?
- Sometimes subject to interpretation



Definition of a Web Application

- Any program that runs in a web browser.
- Typically created in a browser-supported programming language (e.g. the combination of JavaScript, HTML and CSS).
- Relies on web browser to render the application.
- Web apps can be part of web page or web site
- Since there are 'web apps' that create web sites, and web sites that contain web apps, it can be tricky to tell the difference.
- Bottom line is **something with a web interface that does something.**

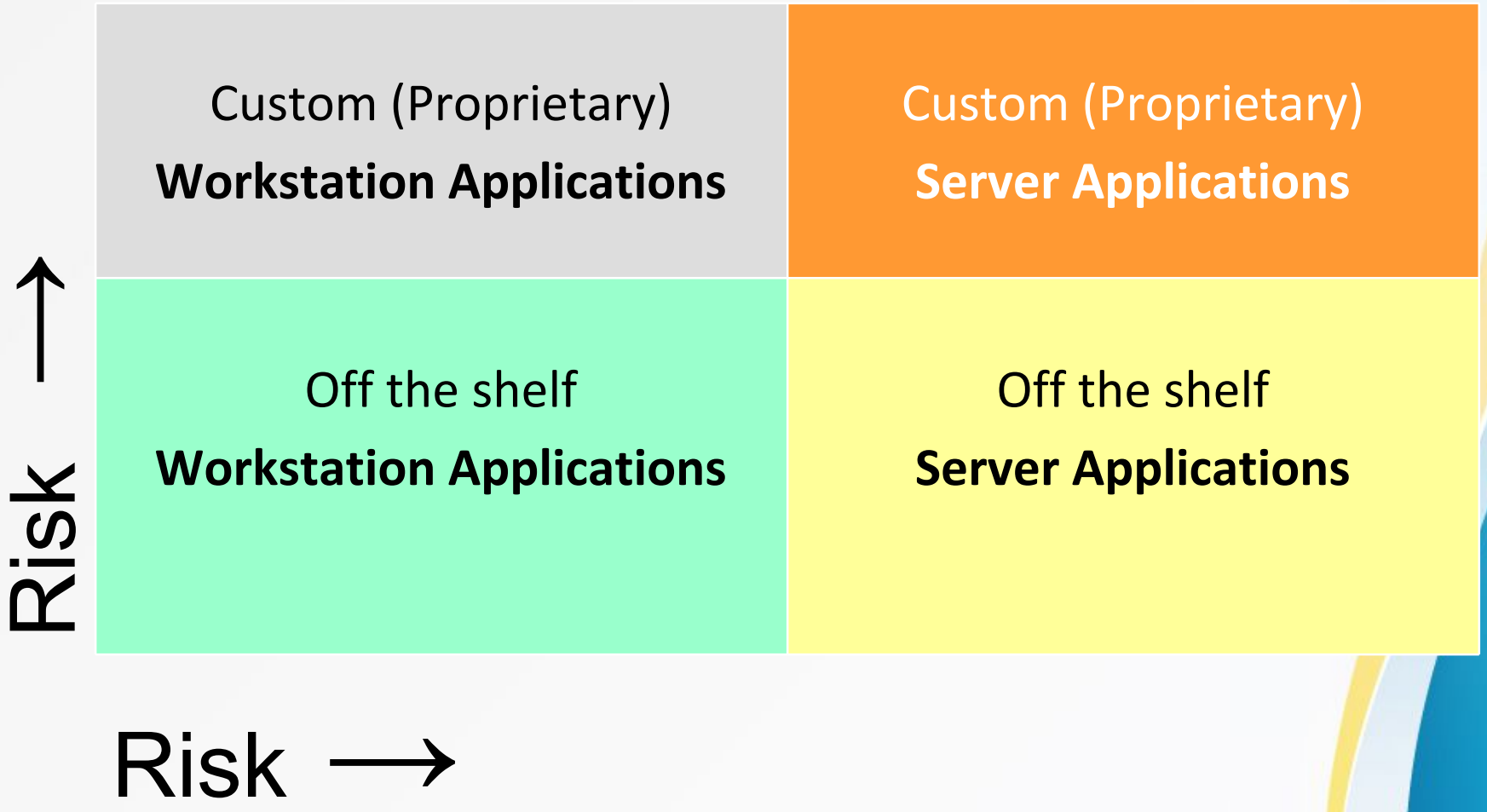
Typical Three-Tier Web Application



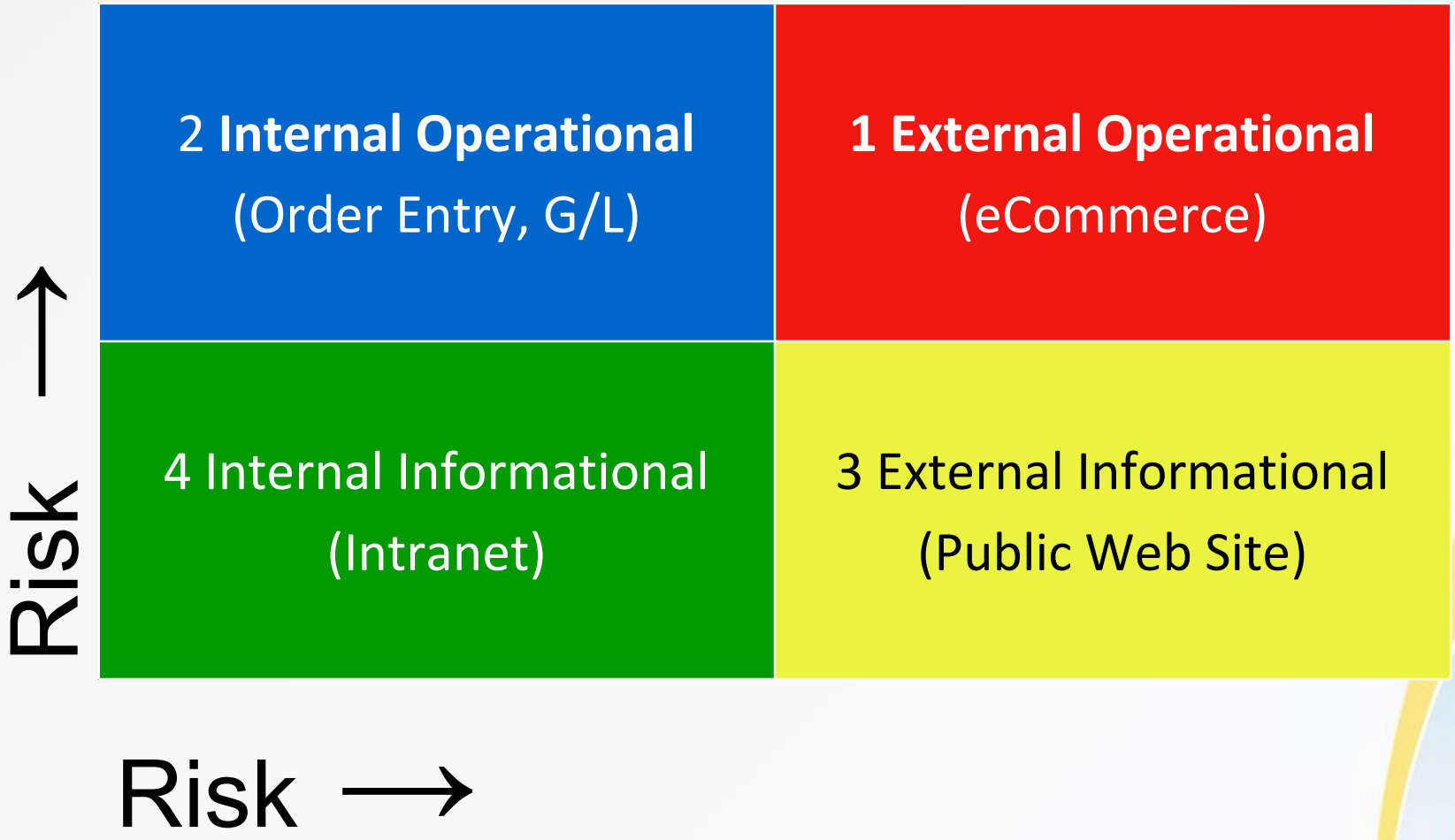
Got Risk?

- Best approach from an audit perspective?
- Hint: Risk Based Approach

The 'Good Old Days' When Risk was Simple



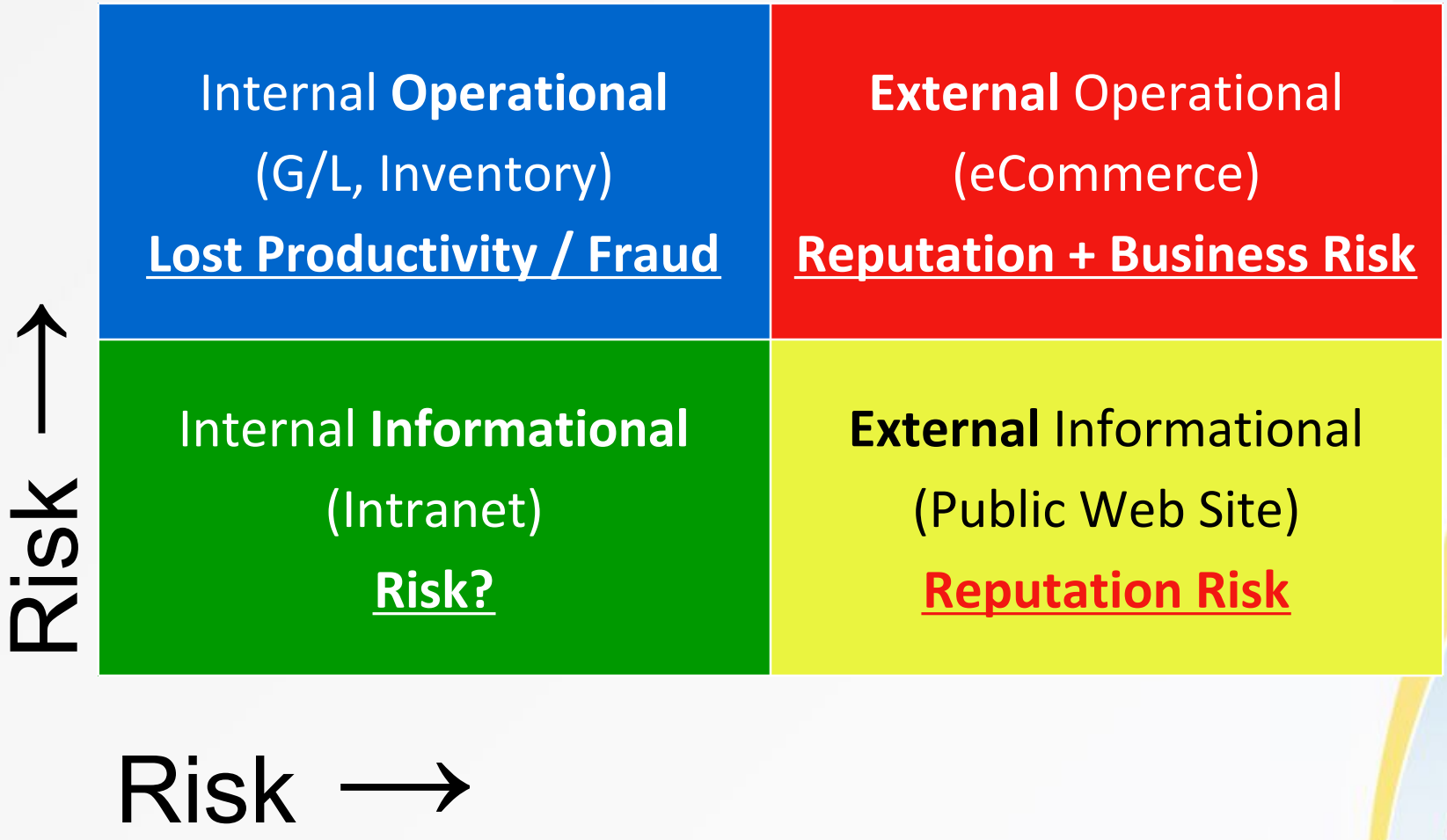
Web Applications > Function vs. Location
Informational or Operational?
Internal or External Facing?



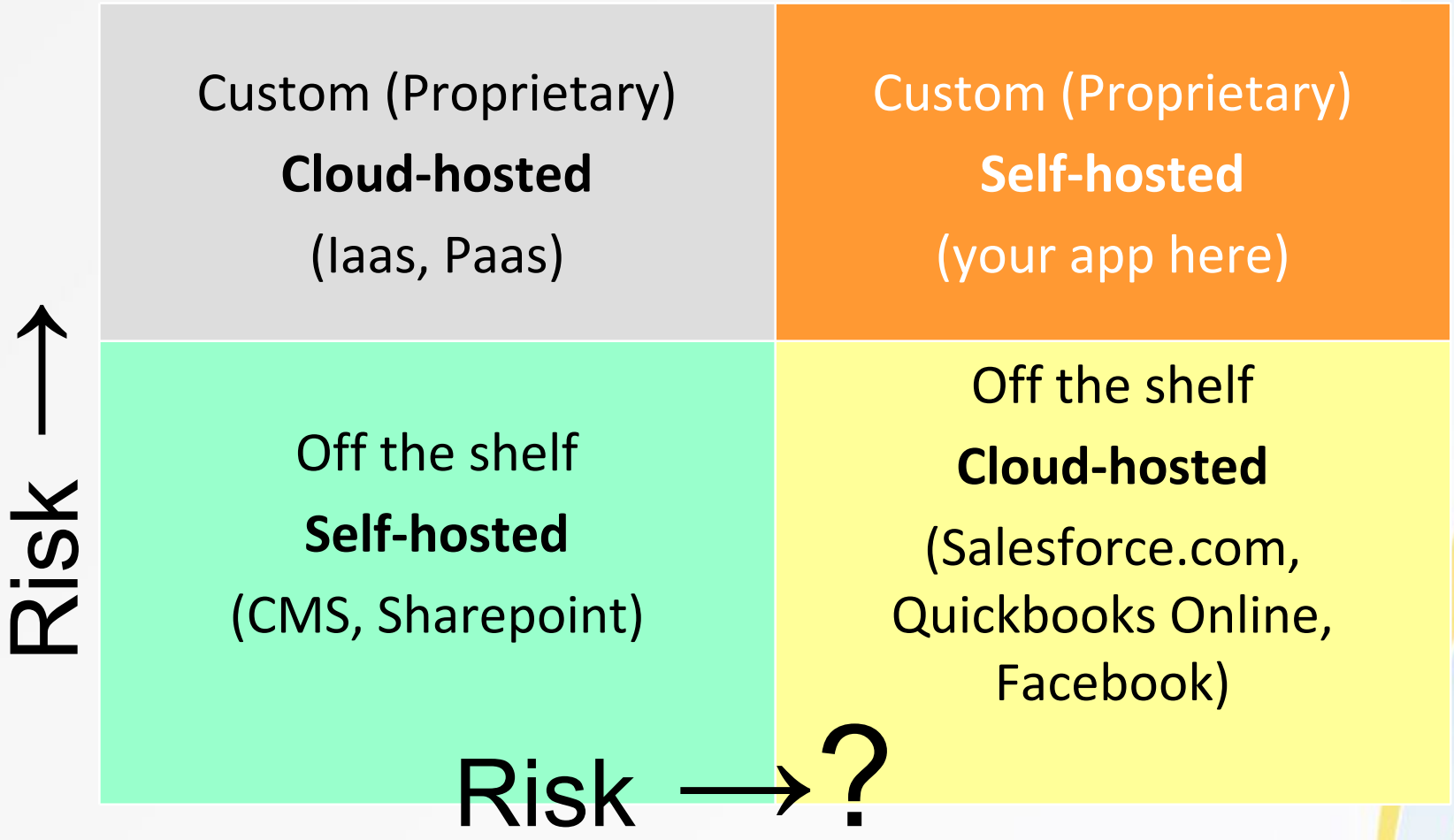
Web Application vs Site vs Web Interface

- Web page/site that 'does something useful'
- Audit risk: page/site that does something bad

Web Applications > Function vs. Location
Primary Risks



Hosting Location (Self hosted vs third party)
Proprietary vs. Off-the-Shelf (Code Development)



Web Application Risk Assessment: Must Carefully Consider Where Key Risks

Internal Operational	**External Operational**	Custom **Cloud-hosted**	Custom **Self-hosted**
Internal Informational	External Informational	Off the shelf **Self-hosted**	Off the shelf **Cloud-hosted**

- Bottom line: web applications are not inherently riskier, but require more thought.

Audit Risk of Web Apps

- Audit Risk = IR x CR x DR
- Inherent x Control x Detection
- Risk we got it wrong
- Operational Internet-Facing Apps most likely riskiest audit we do.

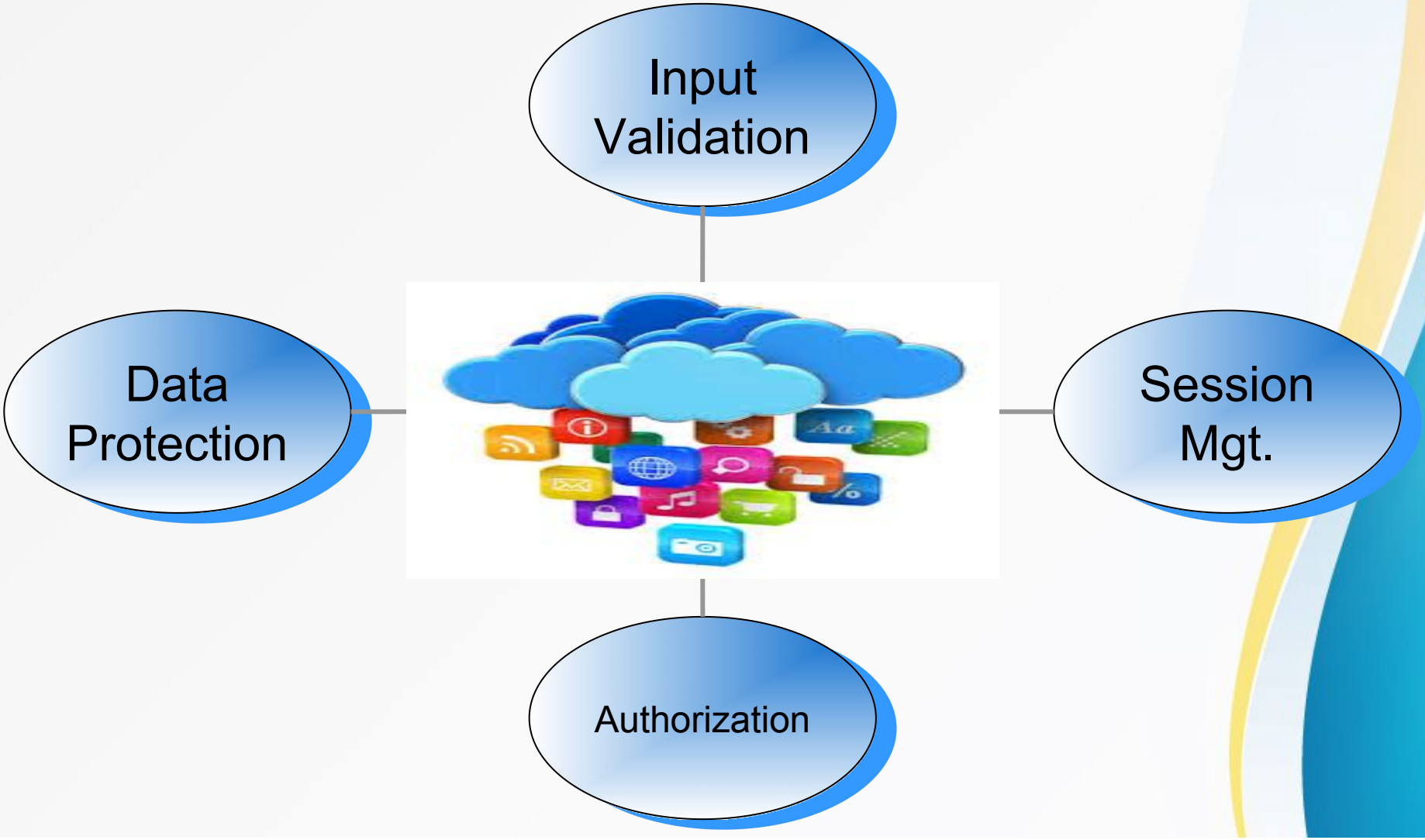
For Highest Risk Items:

- Third-party vulnerability assessment
- Third-party application penetration test

Risks of Web Applications

- **The <u>Strength and Power</u> of Web Applications is that the Interface Can be Adapted and essentially 'created on the fly'.**
- **The <u>Weakness and Risk</u> of Web Applications is that the Interface Can be Adapted and essentially 'created on the fly'.**

Risks of Web Applications



Risks of Web Apps: Input Validation

- **Input Data Validation:** interface untrusted
- **ANY** input can be sent to application

Controls:

- Initial values
- Error/exception handling
- Logging
- Fail safe vs fail open
- Resource exhaustion controls

Risks of Web Apps: Authentication

Authorization & Authentication since application is exposed to untrusted users, privilege escalation is dangerous.

Controls:

- Tokens & Two-Factor Authentication (2FA)
- Authorize transaction, not user
- Use of Certificates for non-repudiation
- Adaptive Authorization, Site keys
- Out of band (SMS/voice)

Risks of Web Apps: Data Protection

- **Network is untrusted** so any communcation can be intercepted and/or modified

Controls:

- Valid SSL certs
- Encrypted connection strings
- Encrypted cookies
- Encrypted database

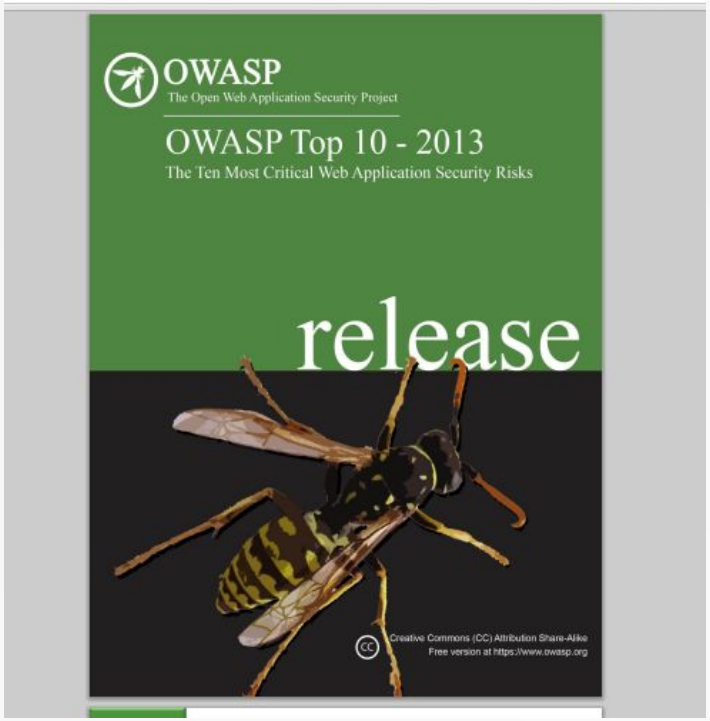
Risks of Web Apps: Session Management

Session Management since **nothing** is trusted! Risk of hijack of user session.

Controls:

- Session time limits
- Geolocation, IP, known PC
- Hard to guess session varibles/IDs

OWASP Top 10 Application Security Risks



https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

A1 – Injection

Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query.

The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Example: http://example.com/app/accountView?id=' or '1'='1

Real world example:

"In July 2012 a hacker group stole 450,000 login credentials from Yahoo!. The logins were stored in plain text and were allegedly taken from a Yahoo subdomain, Yahoo! Voices. The group breached Yahoo's security by using a "union-based SQL injection technique"

See: OWASP SQL Injection Cheat Sheet

Web application used as a SQL query tool

A2 – Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.

Example: http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?car_model=BMW_335i

Real world example:

Intruders accessed Target's network on Nov. 15, 2013 using network credentials stolen from a provider of refrigeration and HVAC systems.

A3 – Cross-Site Scripting (XSS)

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

See: OWASP XSS Cheat Sheet

Example: <script> name='alert(document.domain)'; location.href='http://tw.adspecs.yahoo.com/tc/index.php'; </script>

Real world example:

(January 2013) "Hackers exploit a cross-site scripting (XSS) vulnerability in a Yahoo website to hijack the email accounts of Yahoo users and use them for spam. In the background, a piece of JavaScript code exploits a cross-site scripting (XSS) vulnerability in the Yahoo Developer Network (YDN) Blog site in order to steal the visitor's Yahoo session cookie."

A4 – Insecure Direct Object References

An direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

Example: http://www.victim.com/global.asa+.htr

Real world example:

" WordPress Site Hacks Continue: 70% of WordPress sites are running outdated software..recent examples hit MIT, NEA and Penn State servers."
-Information Week 10/1/2013

The .htaccess file of WordPress is often not properly protected which makes the site vulnerable.

A5 – Security Misconfiguration

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

Example: Webserver admin password set to *password*

Real world example:

Hardened eCommerce server started sending spam email for one day, then suddenly stopped. Firewall administrator had accidentally made a rule change. (personal experience)

A6 – Sensitive Data Exposure

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

Example: Passwords stored unencrypted in database

Real world example:

The Sony PlayStation Network was compromised in April 2011 and the personal details from approximately 77 million accounts were stolen and prevented users of PlayStation 3 and PlayStation Portable consoles from playing online through the service.

A7 – Missing Function Level Access Control

Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

Example: https://vmware1/folder?dcPath=ha-datacenter

Need to prohibit ability to execute functions on web page not just hide page from navigation.

Real world example:

"Server hack prompts call for cPanel customers to take “immediate action”Change root and account passwords and rotate SSH keys, company advises."-Arstechnica February 2013

A8 - Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) is an attack that tricks the victim into loading a page that contains a malicious request. It is malicious in the sense that it inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf, like change the victim's e-mail address, home address, or password, or purchase something. CSRF attacks generally target functions that cause a state change on the server but can also be used to access sensitive data.

Example: Malware infected web browser at hotel steals cookies and allows airline site login

Real world example:

"Security researcher Ronen Zilberman publicly disclosed a new Cross-Site Request Forgery (CSRF) attack vector that uses an HTML image tag to steal a Facebook user's information. According to Zilberman's disclosure, the user needs only to load an infected page to launch the attack." -Internet News 8/20/2009

A9 - Using Components with Known Vulnerabilities

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

Example: Using outdated version of Apache web server

Real world example:

Heartbleed. Technically vulnerability was not 'known', however this illustrates how single component vulnerability can have widespread impact.

A10 – Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

- Example: Link within a site to different server to accept payments.

Real world example: **Super Bowl-Related Web Sites Hacked – PC World 2/2/2007** “The Dolphins' sites were serving up malicious JavaScript code that exploits two known Windows vulnerabilities, then attempted to connect with a second Web server that installs a Trojan horse downloader and a password stealing program on the victim's computer”

Vulnerability Assessment of Web Apps

Web Application Vulnerability Assessment
AKA Application Penetration Test

Explore Application

Identify Vulnerabilties

Test for Exploitability

Explore: What Platform, What Data?

- ✓**What:** what do you have?
- ✓What data is in the system?
- ✓What is the value of the data?

OS	Database	WebServer	Framework	Application	User Interface
Ubuntu 12.04	MySQL 14.5	Apache 7.0.30	PHP 5.4.0	Payroll 1.0	Local HTTPS IE 8.0

Who: Developer or service provider

Who:

- ✓Trusted third party developer or ASP
- ✓Unknown third party developer or ASP
- ✓Trained and seasoned development team
- ✓Eager and inexperienced development team
- ✓Business unit management purchase

Where: Logical & Physical Location

Where:

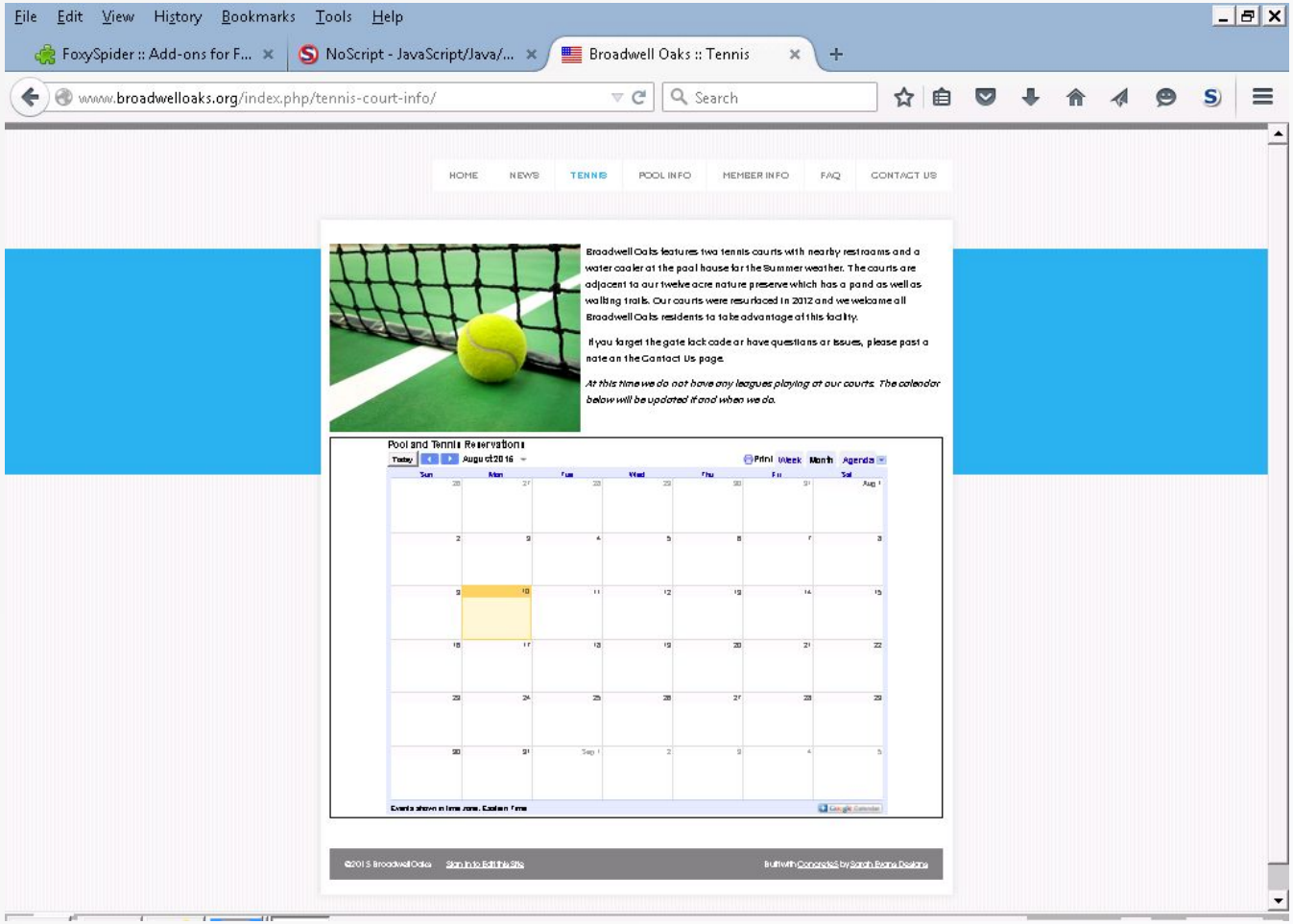
- ✓Local Intranet Web application
- ✓Public website hosted internally
- ✓Customer portal hosted internally
- ✓Remotely hosted web application
- ✓Connectivity, firewall, transport, etc.

One Minute Web Application (or site) Audit

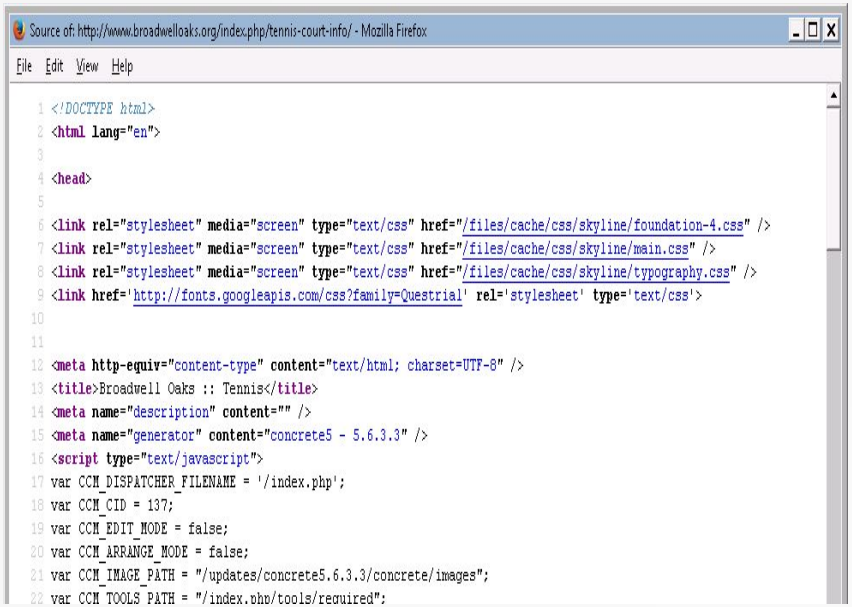
Tools Required:

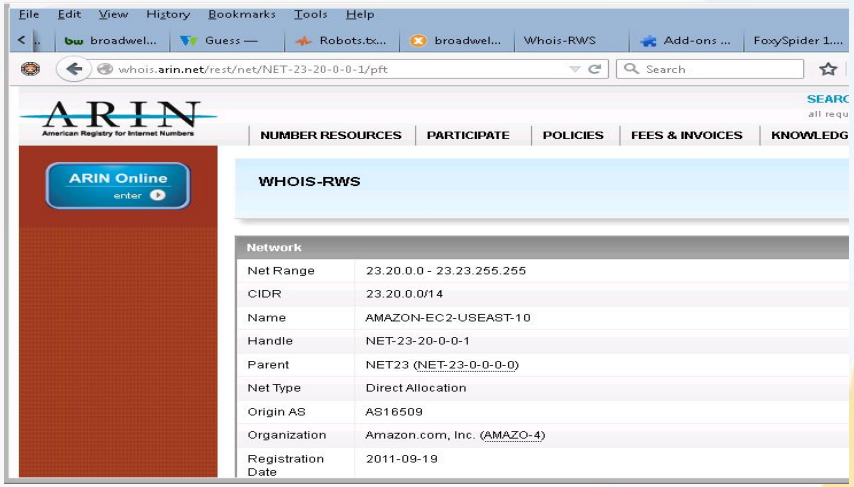
- Web browser
- For this example, am using a site I manage and looking at a simple web application.

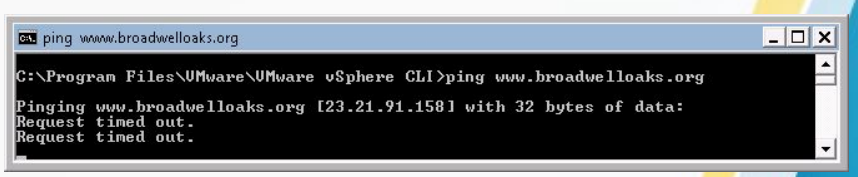
Step One: Look at the page



View Page Source, Ping, Visit BuiltWith



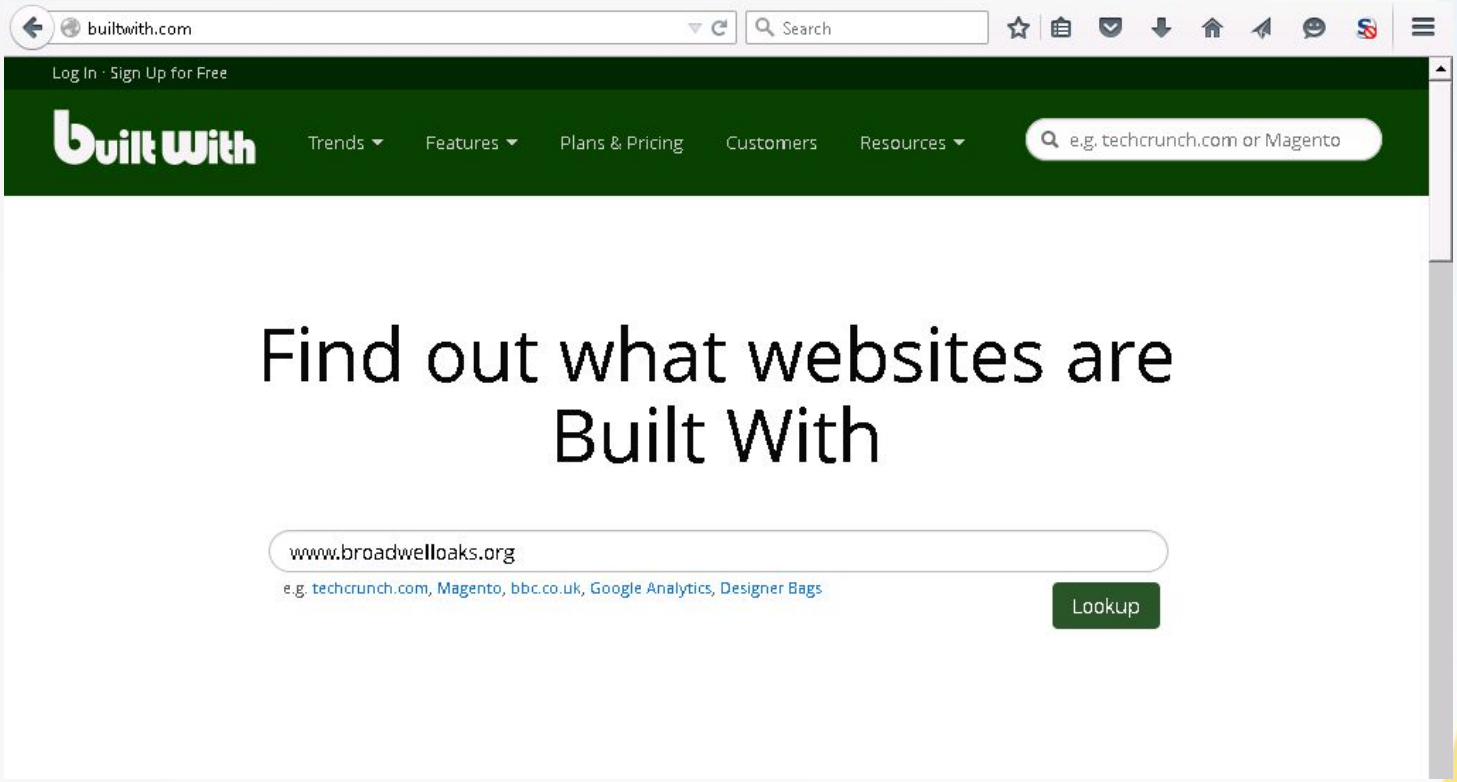




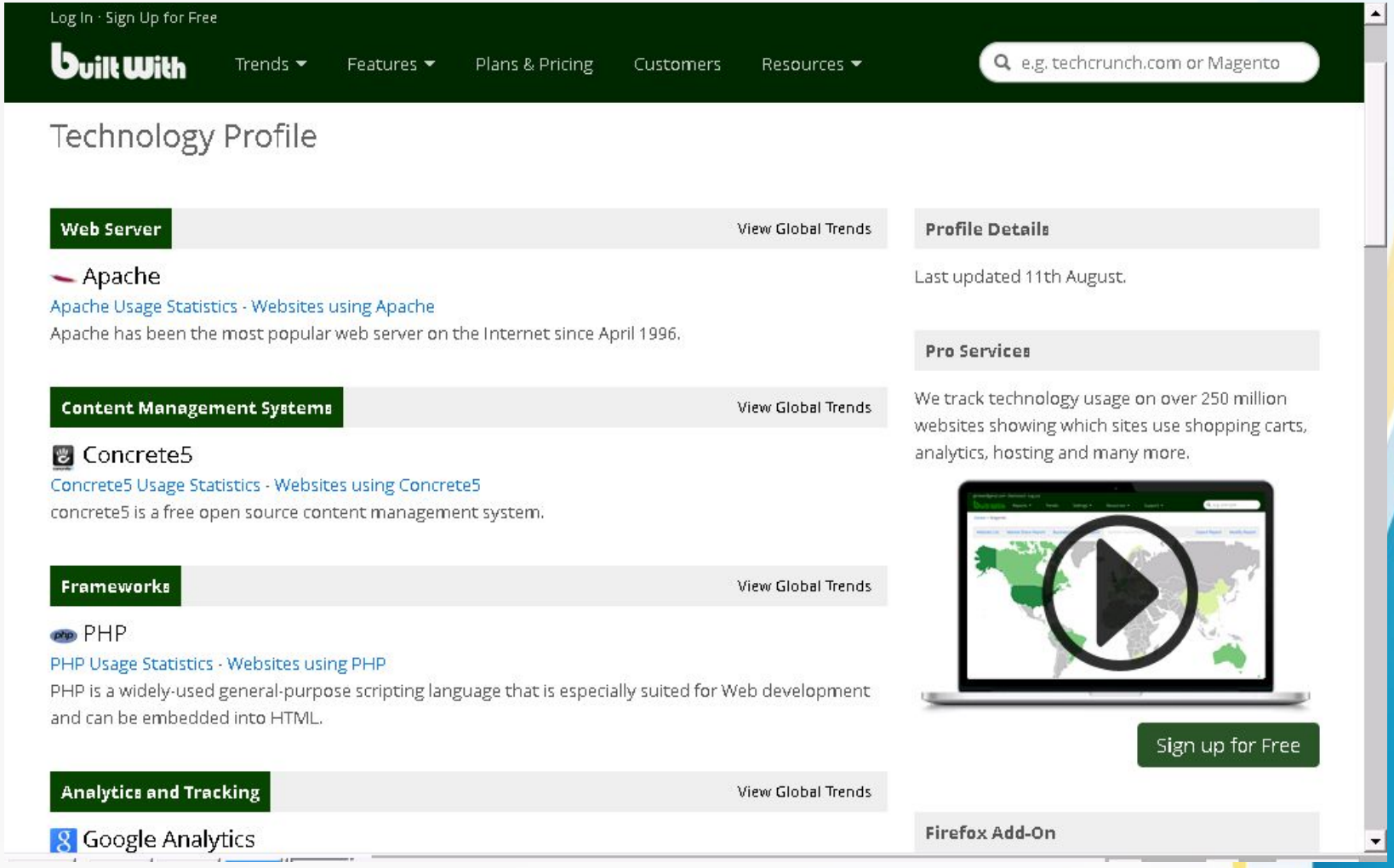
View Page Source

- Concrete5 version 5.6.3.3
- Google Calendar App, obfuscated string

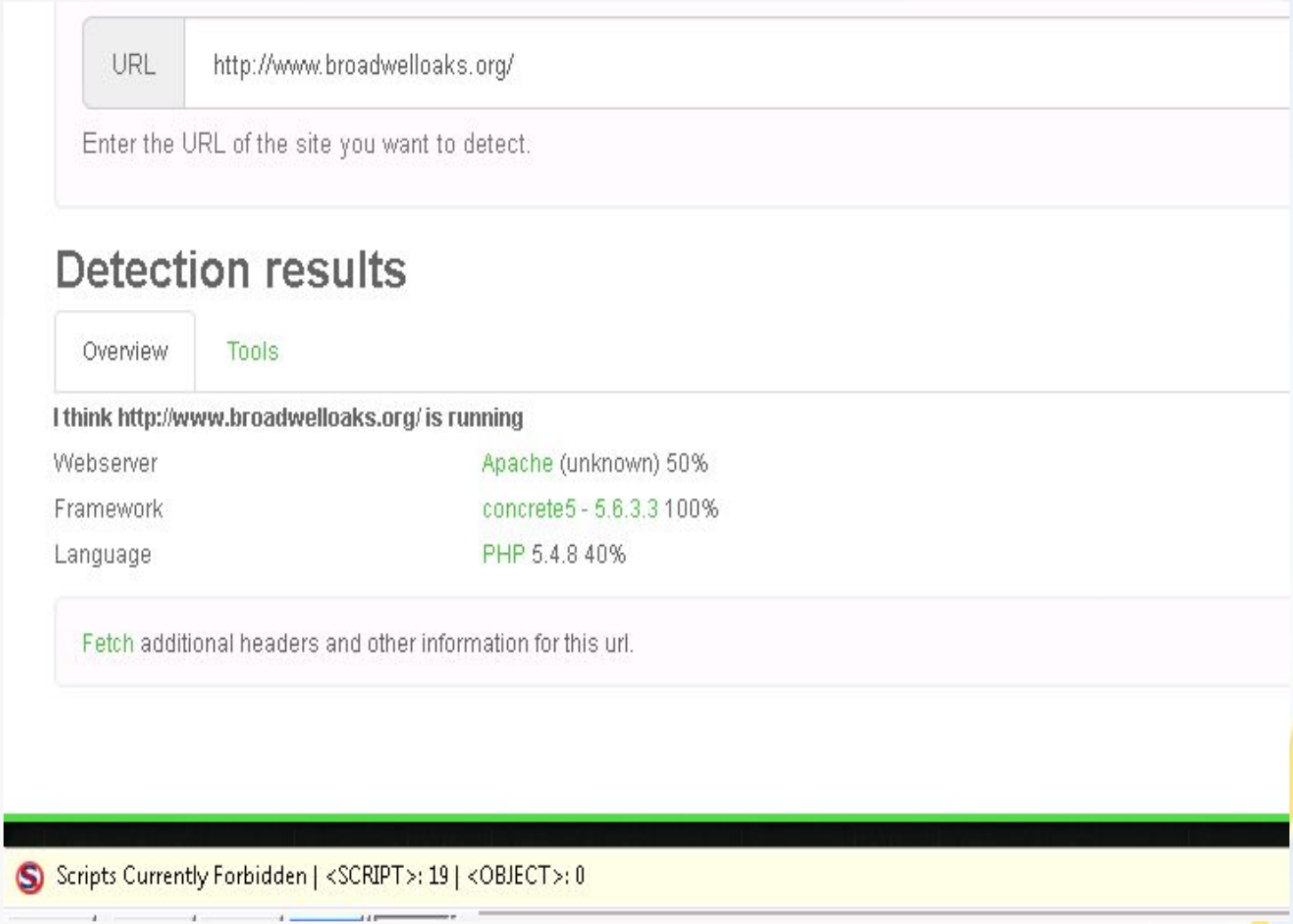
Visit BuiltWith Site



BuiltWith shows Apache, PHP, and dozens of other elements in use.

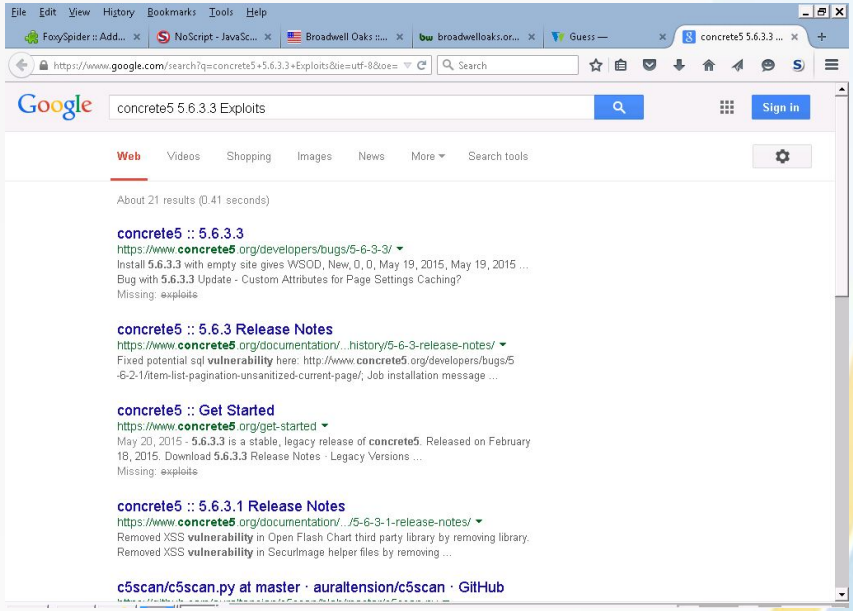


CMS Detector shows more: http://guess.scritch.org

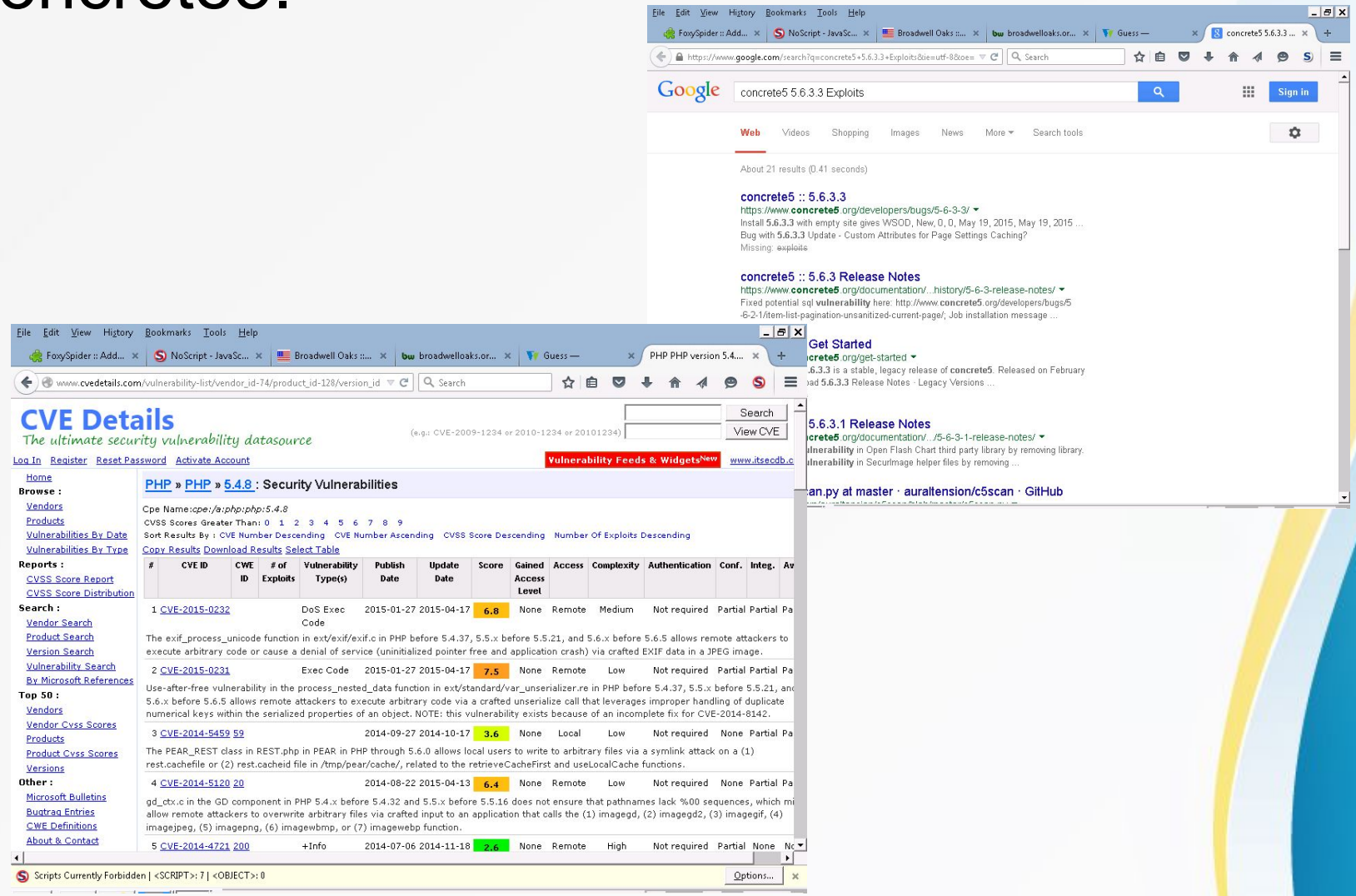


What Do We Know So Far?

- Concrete5 version 5.6.3.3
- Google Calendar App, obfuscated string
- PHP version 5.4.8
- IP of 23.21.91.158
- Hosted at AWS EC2 East



Google: Published Vulnerabilites for PHP and Concrete5, default passwords for Concrete5.



Drive-by Audit Tips

1) Look closely at versions of everything. Is everything patched?
2) Having done what/why/where/when/who part, what **data** are you protecting and how you are protecting it?
3) Look for OWASP Top Ten issues.
4) Do some non-invasive pen tests.

Web Application Security: OWASP

- OWASP Testing Guide Version 4.0

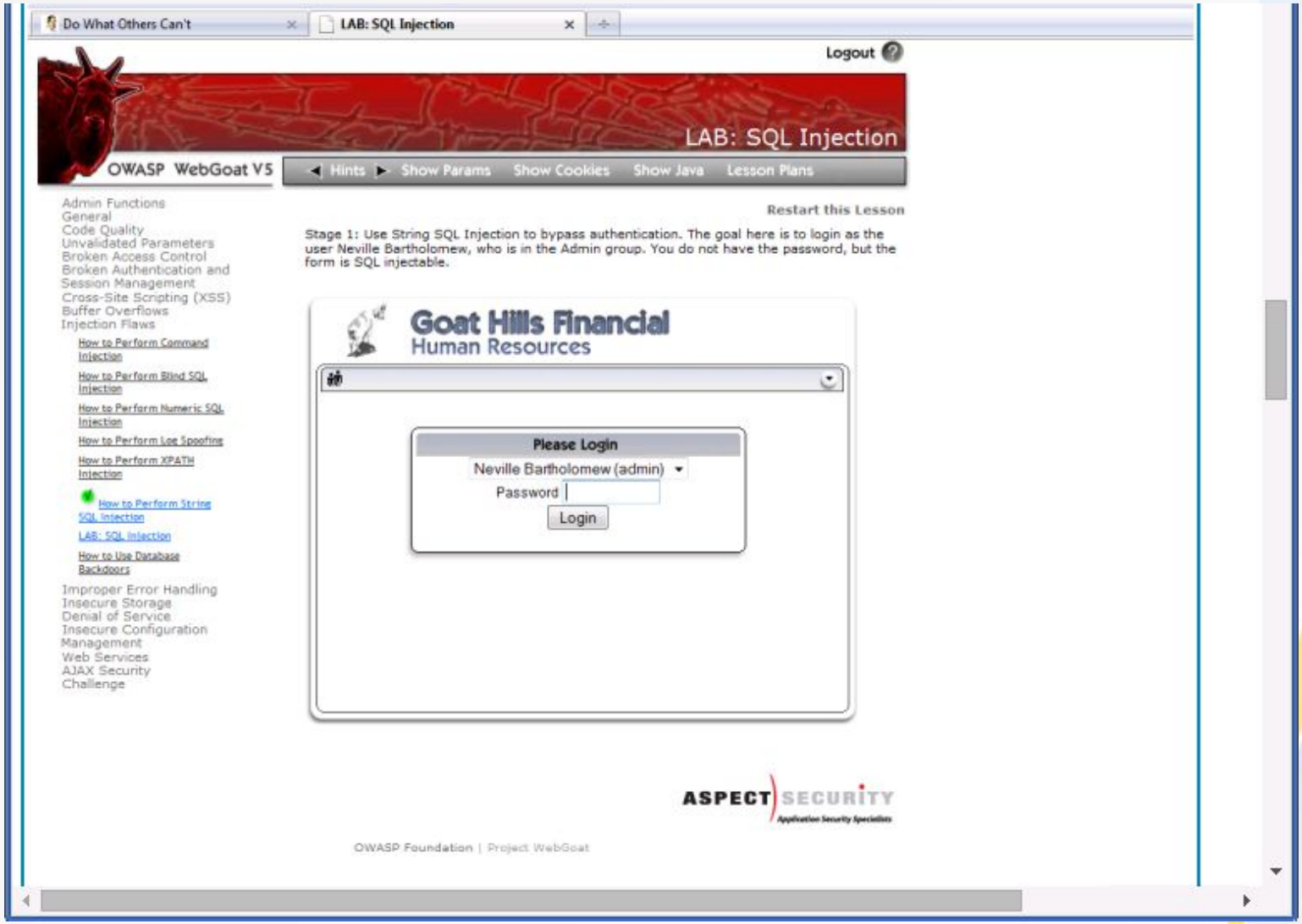
Excellent Free Teaching Tool: WebGoat

- WebGoat is a deliberately insecure web application maintained by OWASP designed to teach web application security.
- Once deployed, the user can go through the lessons and track their progress with the scorecard. There are currently over 30 lessons,

WebGoat Teaches You All About:

- Cross-site Scripting (XSS)
- Access Control
- Thread Safety
- Hidden Form Field Manipulation
- Parameter Manipulation
- Weak Session Cookies
- Blind SQL Injection
- Numeric SQL Injection
- String SQL Injection
- Web Services
- Fail Open Authentication
- Dangers of HTML Comments
- ... and many more!

WebGoat ScreenShot



WebGoat Demo

Takeaways:

- ✓Define Web Application
- ✓Risks Unique to Web Applications
- ✓Explore & Learn about Web Apps
- ✓OWASP
- ✓Web Goat
- ✓Helpful Tips

Questions?

Thanks!!