Real Time Application Defense: Detection and Response with OWASP AppSensor

John Melton
jtmelton(@)gmail.com
http://www.jtmelton.com

2010 Fall Cyber Security Symposium
November 2, 2010
Agenda

- Understanding The Threat
- Application Defense Failures
- Real Application Protection
  - Attacker Detection & Response
  - Application Worm Detection & Containment
- AppSensor Project & Future
Quick Survey
The Threat - Advanced Attackers

- Skilled
- Financially Motivated
- Organized
- Patient and Persistent
- In Possession of Your Source Code
- Outside & Inside Your Company
Application Defense Failures

- “We Use SSL”
- “We Use Firewalls”
- “We Use Deep Packet Inspection”
- “We Installed A Web Application Firewall”
“We Use SSL”

- SSL Protects Transmitted Traffic
- No Guarantee or Inspection of Data
- Zero Impact to Attackers
- Provides Zero Protection to Site Against Attackers
“We Use Firewalls”

- Purpose of Firewall: Allow or Deny Access via Port
- Necessity of Working Web App: Allowed Access via 80 or 443
- Result: Firewall is an Open Door
“We Use Deep Packet Inspection”

- Performed by Generic Network Appliance
- No Knowledge of Application Attacks

- Example Attack: Access Control Attack via Direct Object References
- Not Detected by DPI

GET /updateProfile?id=52473&pass=newpass
Host: yourSite.com
“We Installed a Web Application Firewall”

- Custom application + Generic Solution != success
- Application context not available
- No concept of access violations

```
/viewAccount?id=1002
/viewAccount?id=1004
```

No attacks here, please proceed
Current Approach

Build secure & hope for the best
AppSensor Approach

Build as secure as possible

Detect & eliminate threats
Defend against the unknown

Add layer of attack detection & prevention
Enhancing App Security

Build Secure
- Integrate Security into SDLC
- Security Code Review & Penetration Testing

Actively Defend
- Attack Detection Points
- Application Trend Anomaly Detection
- Automated Response to Quarantine Attackers
Detecting Attacks the Right Way

- **Integration**
  - Detect INSIDE the application
  - Understand business logic

- **Effectiveness**
  - Minimal false positives
  - Immediate response

- **Effort**
  - Automatic detection
  - No manual work required
Inside the Application is Best

- Understand application & business context
- Integration with authentication & user store

![](image)

```
/viewAccount?id=1002
/viewAccount?id=1003
/viewAccount?id=1004
```

```
Authentication
Access Control
Functionality
AppSensor
```
Real Time Defenses Against

MALICIOUS ATTACKERS
Attack Detection: Real vs Cyber World

- Why do bank robbers get caught?
- Why don’t hackers get caught?
Robbing a Bank

Physical Controls
- Electronic Monitoring
- Human Monitoring
- Instant Detection and Response
- Controlled Access
- Multi Factor Auth
- Transaction Verification
Would You Bank Here?

- Ineffective Monitoring
- No Real Time Analysis
- Unnecessary Partner Trust
- Limited Security Training
- Single Factor Auth
- Alternate Admin Access
- Partial External Controls
- No Response Capability
Let’s Change Things - Applications Should...

- Detect attacks
- Understand normal use vs. suspicious use
- Instantly identify attackers
- Shutdown attackers in real time
- Modify application accessibility for defense
Detecting Malicious Users

Many malicious attacks are obvious and not “user error”
- POST when expecting GET
- Tampering with headers
- Submission of XSS attack
Examples of Malicious Actions

- Bypassing client side input validation
- Transaction using functionality not visible to user role
- Multiple access control violations
- Change of user agent midsession
- Double encoded data
Establishing Detection Points

Signature based events:
- Request
- Authentication
- Session
- Access control
- Input
- Exception
- Command injection
- File input/output
- Honey trap

Behaviour based events:
- User trend
- System trend
- Reputation
## Examples

<table>
<thead>
<tr>
<th><strong>ACE1</strong></th>
<th>Modifying URL Arguments Within a GET For Direct Object Access Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exception type</strong></td>
<td>AccessControlException</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The application is designed to use an identifier for a particular object, such as using categoryId=4 or user=guest within the URL. A user modifies this value in an attempt to access unauthorized information. This exception should be thrown anytime the identifier received from the user is not authorized due to the identifier being nonexistent or the identifier not authorized for that user.</td>
</tr>
<tr>
<td><strong>Example(s)</strong></td>
<td>The user modifies the following URL from</td>
</tr>
<tr>
<td></td>
<td>example.com/viewpage?page=1&amp;user=guest</td>
</tr>
<tr>
<td></td>
<td>to</td>
</tr>
<tr>
<td></td>
<td>example.com/viewpage?page=22&amp;user=admin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>UT2</strong></th>
<th>Speed Of Application Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exception type</strong></td>
<td>UserTrendException</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The speed of requests from a user indicates that an automated tool is being used to access the site. The use of a tool may indicate reconnaissance for an attack or attempts to identify vulnerabilities in the site.</td>
</tr>
<tr>
<td><strong>Example(s)</strong></td>
<td>The user utilizes an automated tool to request hundreds of pages per minute.</td>
</tr>
</tbody>
</table>
How Does AppSensor Protect the App?

Requests Needed for Attacker vs. AppSensor

Attacker: find vulnerability

AppSensor: determine user is malicious

# of malicious requests
AppSensor is Faster than an Attacker

- User identified as malicious & blocked before vulnerability is found

Attacker: searching for vulnerability

0 1 2 3 4 5 6 7 8
# of malicious requests

AppSensor Detecting Attacks
AppSensor Blocks Attacker
Response Capabilities

Current Capabilities
- Log
- Logout
- Disable
- Email Admin
- SMS Admin
- Disable Component
- Disable Component for a specific user

Table 1: AppSensor Responses

<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>TYPE</th>
<th>ID</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silent</td>
<td>User unaware of application's response</td>
<td>ASR-A</td>
<td>Logging Change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-B</td>
<td>Administrator Notification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-C</td>
<td>Other Notification</td>
</tr>
<tr>
<td>Passive</td>
<td>Changes to user experience but nothing denied</td>
<td>ASR-D</td>
<td>User Status Change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-E</td>
<td>User Notification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-F</td>
<td>Timing Change</td>
</tr>
<tr>
<td>Active</td>
<td>Application functionality reduced for user(s)</td>
<td>ASR-G</td>
<td>Process Terminated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-H</td>
<td>Function Amended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-I</td>
<td>Function Disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-J</td>
<td>Account Logout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-K</td>
<td>Account Lockout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASR-L</td>
<td>Application Disabled</td>
</tr>
<tr>
<td>Intrusive</td>
<td>User's environment altered</td>
<td>ASR-M</td>
<td>Collect Data from User</td>
</tr>
</tbody>
</table>

http://www.owasp.org/index.php/
File:Owasp-appsensor-responses.pdf
From Theory to Reality

- Demo Social Networking Application
- Leverages AppSensor Principles
Detection Points

- home.jsp
  - RE4
- updateProfile.jsp
  - RE4, IE1, IE4
- friends.jsp
  - RE4
- friendRequests.jsp
  - RE4, 3 x ACE1
- addFriend.jsp
  - RE4
- search.jsp
  - RE4, IE1, RE3
AppSensor vs Scanners

- Tools attempt 10,000s of generic attacks
- AppSensor stops automated scans nearly instantly
AppSensor vs Advanced Attackers

- Very difficult for attacker
- Requires advanced obfuscation for each attack
- Multiple probes == detection

Test 1
Test 2
Test 3
Detecting/Preventing an Application Worm

- Can you find / fix all XSS?
- Pattern matching easily foiled
- Block the common factor!
  - Worms use XSS and CSRF for propagation
  - 1000% usage increase → problem
- Our example:
  (updateProfile, updateStatus, updateName)
Case Study: Samy

- MySpace Application Worm
- XSS worm embedded in User Profile
  - Added Samy as friend
  - Infected viewer’s profile with XSS
- Exponential Growth of Samy’s friends
  - 10 hours – 560 friends,
  - 13 hours – 6400 friends,
  - 18 hours – 1,000,000 friends,
  - 19 hours – site down for repair
Samy vs AppSensor

- AppSensor detects uptick in addFriend usage
- Compares against trended info
- Automatic response initiated
  - Alerts Admin +%200 Add Friend Usage
  - Alerts Admin 2nd time +%500 Add Friend Usage
  - Automatically shuts down Add Friend Feature

Result:
- Worm Contained,
- Add Friend Temporarily Disabled,
- Site Stays Up
The Exploit

- XSS infects victim’s “Status” with worm
- CSRF adds victim as friend of Charlie

```javascript

document.write('I am a worm ' + img);
if(document.URL != 'https://localhost:8443/AppSensorDemo/updateProfile.jsp'){
    xmlHttp = new XMLHttpRequest();
    xmlHttp.open("POST", "https://localhost:8443/AppSensorDemo/UpdateProfile", true);
    xmlHttp.setRequestHeader('Content-Type','application/x-www-form-urlencoded; charset=UTF-8' );
    var attackstr = '<script src=https://localhost:8443/AppSensorDemo/badsite/worm.js></script>';
    sdata = "status=" + attackstr + "&profile=wormed";
    xmlHttp.send(sdata);
    xmlDoc = xmlHttp.responseText;
}
document.close();
```
The Target

Update Your Info

Status:

Profile:

AppSensor Demo Application

Logged in
Login | Home | UpdateProfile | Friends | Search | Logout |

Friends
Add a Friend

<table>
<thead>
<tr>
<th>Friend</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>sue</td>
<td>Gone Fishing</td>
</tr>
<tr>
<td>Fred Parker 6555</td>
<td>Swimming</td>
</tr>
<tr>
<td>Paul Adams 8196</td>
<td>Totally lost</td>
</tr>
<tr>
<td>Angie Thomas 5340</td>
<td>Running</td>
</tr>
<tr>
<td>Peter Chen 7428</td>
<td>Sleeping</td>
</tr>
<tr>
<td>Peter Lee 8910</td>
<td>Looking at bears</td>
</tr>
<tr>
<td>Peter Adams 4110</td>
<td>At work</td>
</tr>
<tr>
<td>George Cook 6293</td>
<td>Reading a book</td>
</tr>
</tbody>
</table>
Setting the Attack

AppSensor Demo Application

Logged in
Login | Home | UpdateProfile | Friends | Search | Logout |

---

**UserName**
charlie

**Status**
I am a worm

**Profile**

Charlie is "patient zero"

XSS to propagate

CSRF to add friend
First Victim - “Molly”

Worm Fires
Molly Infected

AppSensor Demo Application

Logged in
Login | Home | UpdateProfile | Friends | Send Message

Welcome Home
Your Current Status
I am a worm

Pending Friend Requests
Sent Requests awaiting response
Awaiting response from: charlie

Worm Spreads

CSRF Success
Friends Accumulate for Charlie

AppSensor Demo Application

Not Logged in
Login | Home | UpdateProfile | Friends | Search | Logout |

Welcome Home
Your Current Status
I am a worm

Pending Friend Requests

charlie wants to be your friend! [Accept?] [Reject!]
I am a worm

molly wants to be your friend! [Accept?] [Reject!]
I am a worm

sue wants to be your friend! [Accept?] [Reject!]
I am a worm

foo wants to be your friend! [Accept?] [Reject!]
I am a worm

bob wants to be your friend! [Accept?] [Reject!]
I am a worm

Sent Requests awaiting response

Done
Defend with AppSensor

- AppSensor Policy
  - Notify Admin if events > 5
  - Disable Service if events > 10

- AppSensor notices anomaly – alerts admin

**Trend Alert**: Trend greater than 5 - utilization=7
/AppSensorDemo/UpdateProfile
ResponseAction: **Sending Email Alert** to:admin@site.com re: Service
/AppSensorDemo/UpdateProfile

**Trend Alert**: Trend greater than 5 - utilization=6
/AppSensorDemo/addFriend.jsp
ResponseAction: **Sending Email Alert** to:admin@site.com re: Service
/AppSensorDemo/addFriend.jsp
Defend with AppSensor

- Anomaly continues – disable service

**Trend Alert:** Trend greater than 10 - utilization=11
/AppSensorDemo/addFriend.jsp
ResponseAction: **Disabling Service**

**Trend Alert:** Trend greater than 10 - utilization=11
/AppSensorDemo/UpdateProfile
ResponseAction: **Disabling Service**
Worm Contained, Site Stays Up

AppSensor Demo Application

Logged in
Login | Home | UpdateProfile | Friends | Search | Logout |

Search Page
Search:  
Submit

AppSensor Demo Application

Logged in
Login | Home | UpdateProfile | Friends | Search | Logout |

Friends
Add a Friend

<table>
<thead>
<tr>
<th>Friend</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>charlie</td>
<td>I am a worm</td>
</tr>
<tr>
<td>Tom Adams 5047</td>
<td>Going Fishing</td>
</tr>
<tr>
<td>Britney Adams 8031</td>
<td>Running</td>
</tr>
<tr>
<td>Peter Chen 8729</td>
<td>At work</td>
</tr>
</tbody>
</table>
Trend Monitoring Benefits

- Auto detection of attacks
- Automatic worm containment
- Maintain overall site availability
- Insight to scripted traffic / attack probing
Bring AppSensor Into Your Application

A. Build it into Requirements
B. Roll Your Own
   - Detection Points:
   - AppSensor Methodology:
C. ESAPI
   - AppSensor Integration into Java ESAPI
D. Standalone
E. Security Information/Event Management?
   - Add Detection Points into App
   - Integrate Logging into Real Time Monitor
ESAPI / AppSensor Integration Status

- appsensor.jar ready to use w/ESAPI
- AppSensor + ESAPI bundle planned for ESAPI 2.0 rc8
ESAPI / AppSensor Adoption

- **AppSensor**
  - ModSecurity
  - Major Insurance Company - AppSensor standard for all new web apps
  - Mozilla - AppSensor detection integrated into web apps

AppSensor.jar

- Drop-in support for ESAPI
- 3 Line configuration in ESAPI.properties
- Define policies in appsensor.properties
- Add detection points in code (2-3 lines each)
- Done!
How Easy To Setup?

```java
if (AttackDetected){
  new AppSensorException( "X1", "User Error Message",
    "Logged Error Message" + "("+ request.getRequestURI()+ ")"
    + " user (" + ESAPI.authenticator().getCurrentUser().getAccountName() + ")");
}
```
Detecting Attacks

- 50+ attack detection points and growing
- Grouped into logical areas
  - Request, Auth, Input, Access etc
- Most have nearly zero false positive rate
  - POST When Expecting GET
  - Evading Presentation Access Control Through Custom POST
  - Attempt to Invoke Unsupported HTTP Method

AppSensor Project Status

- Core Team:
  - Michael Coates
  - John Melton
  - Colin Watson

- Source in Google Code
- Demo WAR for download
- Tutorial WAR for download


- Application Based Intrusion Detection highlighted in OWASP Top 10 “What’s Coming”
AppSensor Team

AppSensor Core Team
Michael Coates
John Melton
Colin Watson

Contributors
Ryan Barnett
Simon Bennetts
August Detlefsen
Randy Janida
Jim Manico
Giri Nambari
Eric Sheridan
John Stevens
Kevin Wall
Questions?

- Now
- Catch me after
- jtmelton@gmail.com
- http://www.jtmelton.com

OWASP AppSensor Project mailing lists
https://lists.owasp.org/mailman/listinfo/owasp-appsensor-project
https://lists.owasp.org/mailman/listinfo/owasp-appsensor-dev