GOTHAM
DIGITAL • SCIENCE

Tamper Proofing Web Applications at Runtime

Presentation Outline

- What is Tamper Proofing?
- Real-World Tamper Proofing Mechanisms
- OWASP Top 10 Coverage
- Tamper Proofing Method Analysis
  - URLS & Query Strings
  - Form Data
  - Cookies
  - JavaScript

Traditional Security Strategies

- Negative Security (BlackList)
  - Bypass
  - False Positives / Negatives
- Positive Security (WhiteList)
  - Difficult to Generate
  - Still might have false positives
- **Cant stop Authorization Attacks**
  - **Need context**

Tampering Attacks

- Goal: Prevent attacks targeting Embedded Input
  - Editable Data (Text, TextArea, Password)
  - Embedded Input (almost everything else)
- What are Embedded Inputs?
  - URLs (URI and Query String)
  - Cookies
  - HTML Form Inputs EXCEPT editable inputs
  - On average, >80% of all Inputs

Tamper Prevention

- Root Cause Analysis:
  - **Problem:** The application allowed the user to do something they shouldn't have been able to do
  - **Solution:** Only allow the user do what the application expects them to be able to do

- How?
  - **Look at what is presented to the user**
  - If an option is not presented, don't let them use it
  - If we don't <u>ask</u> the user for it, don't accept it!

Requirements

- Transparent to the application / developer
  - Web Server Module (ISAPI,NSAPI,MOD_*)
  - HttpModule (ASP.NET / IIS7)
  - Web Filters (Java)
- Tolerable performance hit
  - There will always be some
- Configurable
  - There is no silver bullet

Tamper Proofing Strategies

- Encryption
  - Prevents un-authorized viewing and tampering
  - Requires a secret key
- Abstraction
  - Prevents un-authorized viewing and tampering
  - Requires a storage location (client or server-side)
- Hashing (HMAC)
  - Prevents tampering
  - Requires a secret key or storage location

Tamper Proofing Strategies

- Define entry points
  - /default.aspx
  - /login.aspx

- Analyze entry point responses
  - URLs & Query Strings
  - HTML Form Inputs
  - HTTP Cookies
  - JavaScript Functions & Variables

Tamper Proofing Output

- Any effective mechanism must have two basic components
  - Input Validation & Transformation (easy)
    - Usually a Filter
    - May leverage framework for parsing
    - *Relatively* simple structure / format

Tamper Proofing Output

- Any effective mechanism must have two basic components
  - Output Transformation (hard)
    - Filter / Parser
      - HTML Forgiveness
      - JavaScript Complexity
    - Framework Tag / Control Extensions
      - Inconsistent Tag Use

Tamper Proofing Mechanisms

- Real World Tamper Proofing Mechanisms
  - Commercial Application Firewalls
    - Might offer some of the discussed protection mechanisms
  - Freeware
    - All are embedded mechanisms (software-only)
    - Deployable at the web server OR application level
    - May be applicable to only a specific application framework

Http Data Integrity Validator

- **Http Data Integrity Validator (HDIV)**
  - Java Web Filter & Custom TagLibs
  - Generates _HDIV_STATE_ token for each request
    - Server-side Reference to State Information
    - Client-side (encrypted /hashed)
  - Abstracts embedded QueryString and Form Data
    - Confidentiality
  - Editable Data Protection
    - Provides generic validators (configurable) for editable inputs

Http Data Integrity Validator

- **Http Data Integrity Validator (HDIV)**
  - Works with Struts 1.x, Struts 2.x, Spring MVC and JSTL (overrides framework HTML tags at runtime)
    - Does not parse HTML output, so data not rendered using a framework tag is not protected (JavaScript)
  - Version: 2.0.4 - Mar 11, 2008 (hdiv.org)

IIS Secure Parameter Filter (SPF)

- **IIS Secure Parameter Filter (SPF)**
  - ASP.NET HttpModule (C#)
  - Appends URLToken to every URL
    - Validates URI and any embedded Query String values
  - Encrypts Embedded Form Data and Cookies
  - Inserts a Form ID to capture state of each form
    - Only “enabled” inputs will be permitted
    - Only encrypted embedded inputs are accepted
    - Verifies Read-Only text attributes on form submission
  - Configurable JavaScript protection

IIS Secure Parameter Filter (SPF)

- **IIS Secure Parameter Filter (SPF)**
  - Parses Response HTML (not tied to Framework)
    - Uses HTML Agility Pack to parse HTML responses
    - Non-ASP.NET application can also be protected on IIS7
  - Optional “BlackList” RegEx protection capability
  - Future Enhancements
    - Input Abstraction
    - AJAX Support
    - ASP.NET Control Override
  - Version: 1.0.1 – Dec 1, 2008 (gdssecurity.com)

Mod Anti-Tamper

- **Mod Anti-Tamper**
  - Apache Module (written in C)
  - Parses outbound web server responses (Regex) for embedded links
  - Appends encrypted token (HMAC) to embedded query strings and cookie values
    - Does not cover FORM data
  - Rumors of integration into mod_security
  - Version 0.1 - 2005 (wisec.it)

OWASP Top 10 Coverage

A1 - Cross Site Scripting (XSS)

- URL tokens should thwart reflected XSS exploits (if tied to a session cookie)

A2 - Injection Flaws

A3 - Malicious File Execution

A4 - Insecure Direct Object Reference

A5 - Cross Site Request Forgery (CSRF)

- URL tokens should provide added benefit of CSRF protection (if tied to a session cookie)

OWASP Top 10 Coverage

A6 - Information Leakage and Improper Error Handling

- Encryption might mitigate information leakage within application inputs (hidden fields, cookies, etc)

A7 - Broken Authentication and Session Management

- Cookie protection will mitigate weak/predictable session IDs

A8 - Insecure Cryptographic Storage

A9 - Insecure Communications

A10 - Failure to Restrict URL Access

- URL tokens should thwart forced browsing

Tamper Proofing Considerations

- Tamper Proofing Considerations
  - URIs
  - Form Data
  - Cookies
  - JavaScript

Tamper Proofing URIs

- URIs
  - Every URI that is not an entry point is generated by the application
    - A HREF
    - FORM ACTION
    - SCRIPT/IMG SRC
  - Not normally considered “input”

Tamper Proofing URIs

- Encrypt the URI
  - Decrypt and replace URI on every request

    http://foo.test/UserProfile.aspx

    ↓

    http://foo.test/5a47670634430784a6db394.aspx

- URL Tokens
  - Token is an HMAC of URL or Server-Side Reference
  - Validate token on every request

    http://foo.test/UserProfile.aspx

    ↓

    http://foo.test/UserProfile.aspx?token=5a476b394d535a7063443

Tamper Proofing URIs

- Other Considerations
  - URL Length Limitations
    - Vary by Browser & Web Server
  - If authenticated, must be tied to the user / session
  - Can’t be tied to user/session if link-able
  - Token must be tied to URI
  - Token must NOT be alterable by user

Tamper Proofing Query Strings

- Query Strings
  - Embedded query strings within HTML URLs
    - A HREF, FORM ACTION, SCRIPT/IMG SRC
  - FORMS using GET method will also generate query string data
    - We will address FORM inputs separately

GOTHAM DIGITAL • SCIENCE

Tamper Proofing Query Strings

- Encrypted Query String
  - Decrypt on each request

    http://foo.test/UserProfile.aspx?id=392

    ↓

    http://foo.test/UserProfile.aspx?qs=394d535a7063443078
- Query String Token
  - HMAC or Server-Side Reference
  - Can be combined with URI to cover entire URL
  - Validated on every request

    http://foo.test/UserProfile.aspx?id=392

    ↓

    http://foo.test/UserProfile.aspx?id=392&token=394d535a706344

GOTHAM DIGITAL • SCIENCE

Tamper Proofing Query Strings

- Query String Abstraction
  - Requires a storage location for data (protected)
  - Re-populate real values on each request (key lookup)

```
http://foo.test/UserProfile.aspx?id=392
                ↓
http://foo.test/UserProfile.aspx?id=0&key=0-1-526189
```

- Other Considerations
  - Same as with URIs (see previous list)
  - URI and QueryString "token" can be the same
  - Query String Values should not be interchangeable

Tamper Proofing FORM Data

- HTML Form Data
  - Embedded HTML FORM data
    - TYPE=HIDDEN | RADIO | CHECKBOX, SELECT
    - Read-Only Text Boxes
  - Forms can use either GET or POST

Tamper Proofing FORM Data

- Encrypt Embedded Inputs
  - Decrypt parameters on each request

```
<INPUT TYPE="hidden" NAME="acct" VALUE="149">
                 ↓
<INPUT TYPE="hidden" NAME="acct" VALUE="4d535a7067">
```

- Hashing or Abstraction
  - Can use Hashing or Lookup Table
  - Requires a storage location
    - Server-side list (with a unique lookup key)
    - Client-side via hidden field (must be tamper proof)

Tamper Proofing FORM Data

- Example HTML Form (Original)

```
<form method="post" action="/UserPreferences.aspx">
 <select name="Color">
  <option value="RD">Red</option>
  <option value="GR">Green</option>
  <option value="BL">Blue</option>
 </select>
 <input type="radio" name="theme" value="T323" />Classic<br/>
 <input type="radio" name="theme" value="T301" />Modern<br />
 <input type="radio" name="theme" value="T100" />Text-Only<br />
 <input type="submit" value="Submit" />
</form>
```

Tamper Proofing FORM Data

- Example HTML Form (Abstracted)

```
<form method="post" action="/UserPreferences.aspx">
 <input type="hidden" name="formId" value="0-2-526189" />
 <select name="Color">
  <option value="0">Red</option>
  <option value="1">Green</option>
  <option value="2">Blue</option>
 </select>
 <input type="radio" name="theme" value="0" />Classic <br />
 <input type="radio" name="theme" value="1" />Modern <br />
 <input type="radio" name="theme" value="2" />Text-Only <br />
 <input type="submit" value="Submit" />
</form>
```

Tamper Proofing FORM Data

- Other Considerations
  - Inputs cannot be tied to session for 3rd party Forms
    - Same as a linkable URL
  - Storage location must be protected
    - Key-based lookup (server-side)
    - Encryptred or HMAC (client-side)
  - Interchanging Protected Inputs
    - Form ID should be tied to Action URI
    - Tie input value to NAME & ACTION
  - Javascript (covered later)

Tamper Proofing FORM Data

- Disabled or Read-Only Form Inputs
  - Disabled inputs DO NOT submit, so should be ignored
  - Read-Only inputs should be treated as embedded but are still rendered within the UI
    - Cannot be visibly altered
    - Must verify integrity
- Input NAMES must be tracked on each form
  - INPUT TYPE=IMG Example
  - Avoid altering due to client-side references

Tamper Proofing Cookies

- Encrypting Cookies
  - Decrypt cookies on each request

```
Set-Cookie: user-id=123;
        ↓
Set-Cookie: user-id=4d535a7067;
```

- Hashing or Abstraction
  - Can use Hashing or Lookup Table
  - Requires a storage location
    - Server-side list (with a unique lookup key)
    - Client-side (must be tamper proof)

Tamper Proofing JavaScript

- JavaScript used to perform request-related tasks
  - Populate data (URLs and Inputs)
- Too complex to parse for tamper proofing
- Common constructs for request related tasks:
  - Function Calls
  - Variable / Property Assignments

Tamper Proofing JavaScript

- Protecting Function Calls
  - Define the function name and arguments
  - Determine which need to be protected
  - Determine the data type of each
    - URL or FORM INPUT
- Protecting Variables or Properties
  - Several common properties that are URLs
    - location.href, window.location
  - Custom variables require name and data type to be specified

Tamper Proofing JavaScript

- Protected Function Call:

```
__doPostBack('ctl00$CustomerHeader$btnSearch','')
                    ↓
__doPostBack('4d535a706738b1ca827e90fc284ba628c3ef231','27e90fc284ba')
```

- Caveats:
  – Limitations since the target form may not be accessible
    - Abstraction may not be possible
    - Input value may not be tie-able to the target URI
  – Likely to require manual configuration

Conclusion

- Tamper proofing allows us to only worry about what we **ask** the user for
- Real world solutions do exist and work
- As long as tested, either works or doesn’t

Questions?