Understanding ESAPI

Lavakumar K
Penetration Tester, Royal Bank of Scotland
lavakumark@gmail.com

OWASP
29th Nov, 2008

The OWASP Foundation
http://www.owasp.org

About me:

- Have been doing security auditing for 3 years
- Performed more than 100 penetration tests
- Perl and C# programmer

I write code for pleasure

And break code at work ☺

Why does my application have the same vulnerabilities every time?

Developers are trusted with reinventing the security wheel each time.

- Developer type 1: Is ignorant of the common vulnerabilities
- Developer type 2: Understands the vulnerabilities but does not build security controls because of….
- Developer type 3: Builds the security controls but does not get it completely right each time
- Most teams have all the three types

What can be done about this?

- Train developers about common security threats – Shouldn't be a problem
- Identify your most common security needs and make functions for each of them. Now the developers can easily call these functions from their code – Hmm…good idea…this could take up sometime
- Get your best developers to code these functions just right, a mistake here could make the entire application vulnerable – Now this is the tricky part…HELP!!!

Behold the ESAPI – Enterprise Security API

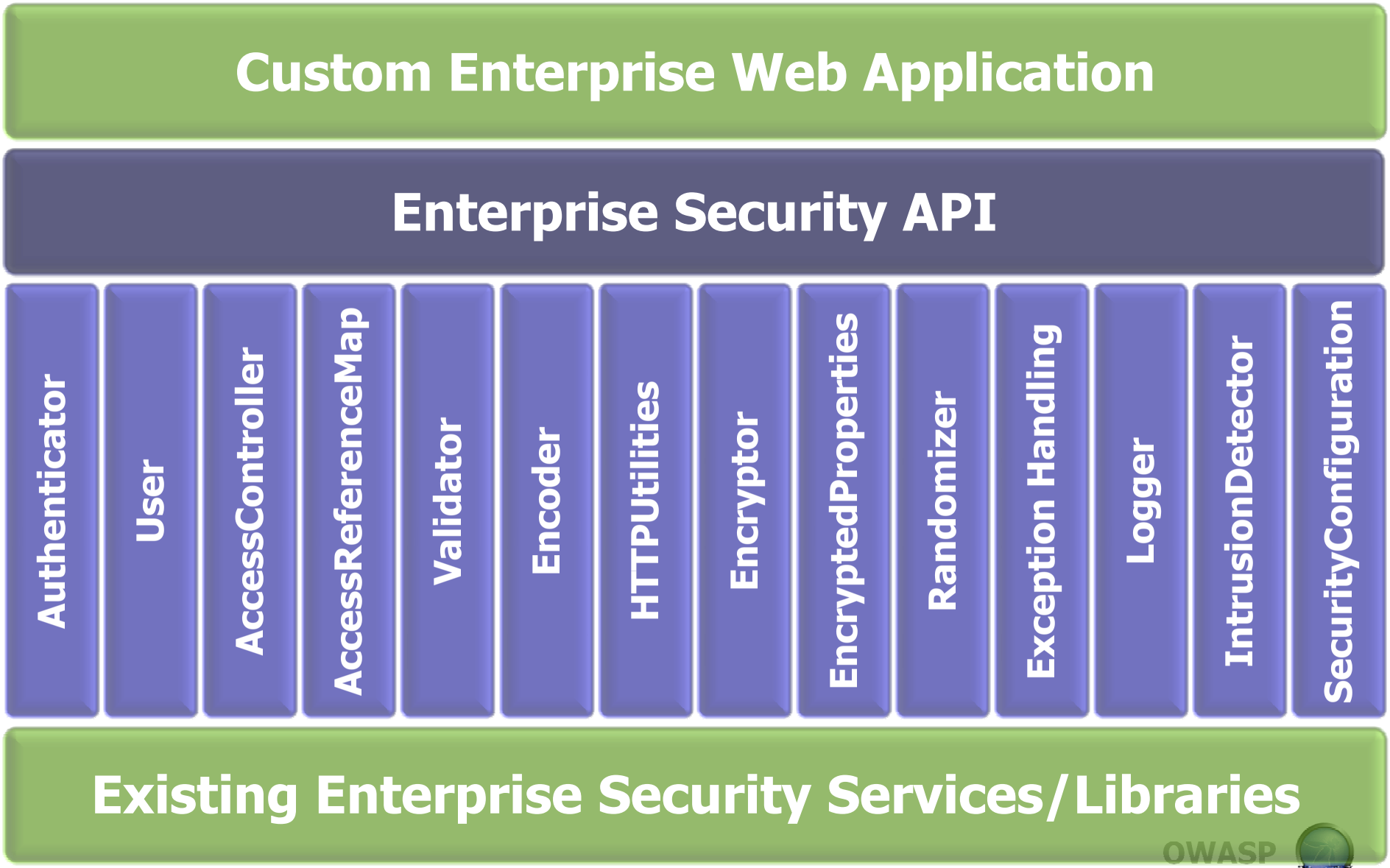
ESAPI is:

- A set of interfaces which provide functions for most of the common security needs of enterprise developers.
- A reference implementation which has implemented the most trickiest and hardest of these functions in the best possible way. Tested thoroughly by industry experts.

Jeff Williams, the chairman of OWASP and CEO of Aspect Security is the author of ESAPI.

Thanks Jeff!!

Architecture Overview



Authenticator Class

Common problems with authentication:

- Username enumeration
- Session fixation
- Parallel active sessions
- Ineffective account lockout
- Authentication bypass
- Weak credentials
- Improper session termination
- Authentication over HTTP
- Insecure implementation of “Remember me” cookies

Authenticator() handles most of this problems invisibly

Methods

- login(request, response) returns user object

Authenticates the user based on:

- Username and password
- "Remember me" cookie

If the user is already logged in then gets the currently logged in user from the session-id

Protects against:

- Session fixation
- Password guessing attacks with account lockout
- Username enumeration
- Session hijacking from a different host

Key Methods

- createUser()
- generateStrongPassword()
- getCurrentUser()
- logout()
- verifyAccountnameStrength()
- verifyPasswordStrength()

Reference implementation uses a text file as repository.

This must be extended based on the existing authentication functions used by your organization

User Class

- Provides a convenient way of referencing any User of the application.
- Use the Authenticator class to get the user object.

Examples of getting the User object:

```
User u = ESAPI.authenticator().createUser(”owasp”,”owasp123",”owasp123");
User u = ESAPI.authenticator().login(request, response);
User u = ESAPI.authenticator().getCurrentUser(); (From thread local variable)
User u = ESAPI.authenticator().getUser(”owasp”);
```

Key Methods

- changePassword()
- disable()
- enable()
- getAccountName()
- getCSRFToken()
- getLastFailedLoginTime()
- getLastLoginTime()
- getRoles()
- isInRole()
- isEnabled()
- isExpired()
- isLocked()
- resetCSRFToken()
- resetPassword()
- isSessionTimeout()
- isSessionAbsoluteTimeout()

CSRF Protection

- The User class uses the resetCSRFToken() method to create a CSRF token for an user.
- This token can be added to the URLs for CSRF protection.
- Adding this token to the URLs and verifying them can be done from the HTTPUtilities classs.

Token creation:

```
String token = u.resetCSRFToken();
```

Accessing the token:

```
String token = u.getCSRFToken();
```

HTTPUtilities Class

- Provides a safe version of the request and response object (Automatic canonicalization)
- Add and verify CSRF tokens
- Set remember me cookies in secure way
- Safe upload of files
- Encryption and decryption support for:
  - Session data in Cookies
  - Querystrings
  - Hidden form fields

Key Methods

- addCSRFToken()
- verifyCSRFToken()
- assertSecureRequest()
- changeSessionIdentifier()
- encryptHiddenField()
- decryptHiddenField()
- encryptQueryString()
- decryptQueryString()
- encryptStateInCookie()
- decryptStateFromCookie()
- getSafeFileUploads()
- safeSendForward()
- safeSetContentType() - prevents encoded XSS payloads from working
- setNoCacheHeaders()
- setRememberToken()

Safe file upload feature

- Common problems with file upload:
  - Directory traversal
  - Failure to verify file extensions
  - Null byte injection or other validation bypass
  - Command injection
- Reference implementation validates the filename, extension and length of the uploaded file
- Reference implementation does not do content content-type validation and virus check
- Developers may add file-header check and virus scan depending on the types of files allowed and how they are used.

AccessController Class

- Access control decisions are required through out the application
- Many times the control logic is complicated:

```
If ((role == admin)&&(day != 'Sunday')&&(IP == 10.*.*.*))
 {
      Show Admin Console
 }
 Else
 {
      Ask user to get a life
 }
```

- Making these complex decisions all over your code makes the application hard to develop and maintain

ESAPI's approach

- ESAPI lets you maintain a separate Access Control List which contains all the complex rules to make decisions
- The AccessController class provides methods which refer these ACLs.
- The developer has to simply use these methods in the code
- Making a change to the control logic only requires a change to the ACL and not to the actual code
- Greatly reduces complexity

Key Methods

- isAuthorizedForData()
- isAuthorizedForFile()
- isAuthorizedForFunction()
- isAuthorizedForService()
- isAuthorizedForURL()
- assertAuthorizedForData()
- assertAuthorizedForFile()
- assertAuthorizedForFunction()
- assertAuthorizedForService()
- assertAuthorizedForURL()

Method categorization

- isAuthorizedFor:
  - Returns a Boolean.
  - To be used in presentation layer
- assertAuthorizedFor:
  - Throws an AccessControlException when unauthorized
  - To be used in business logic layer
  - Easy to detect attacks

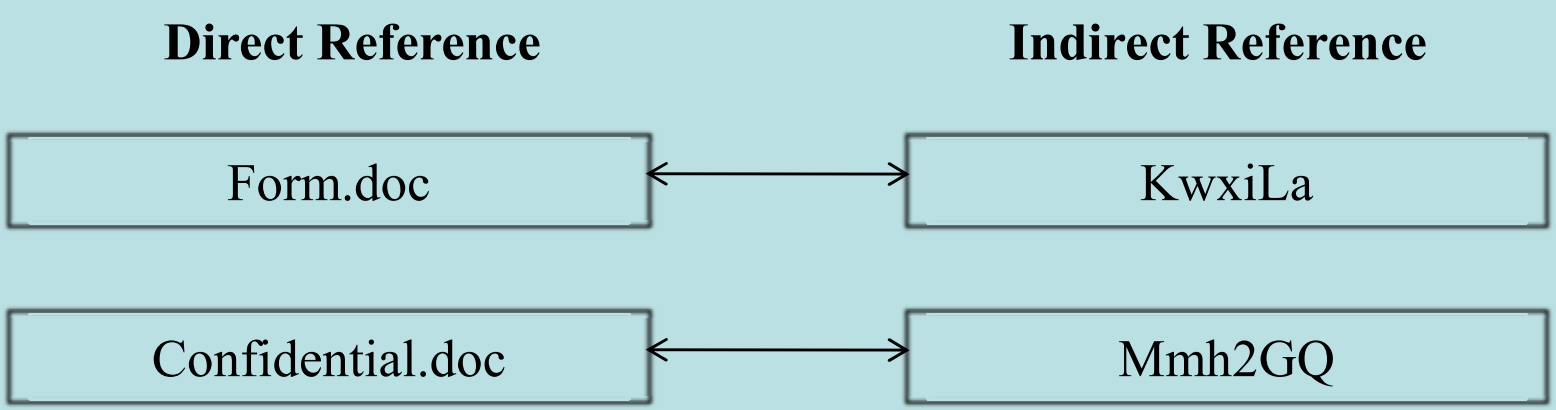
AccessReferenceMap Class

- Direct Object reference is an OWASP Top 10

Eg: http://www.example.com/download.aspx?file=form.doc
http://www.example.com/profile.aspx?accID=1011

- Can result in:
  - Data mining
  - Injection attacks
  - Privilege escalation
  - Arbitrary file downloads
- ESAPI provides an Access Reference Map which contains an indirect object reference mapped to every direct object reference and only the indirect reference is exposed to the user

Access Reference Map



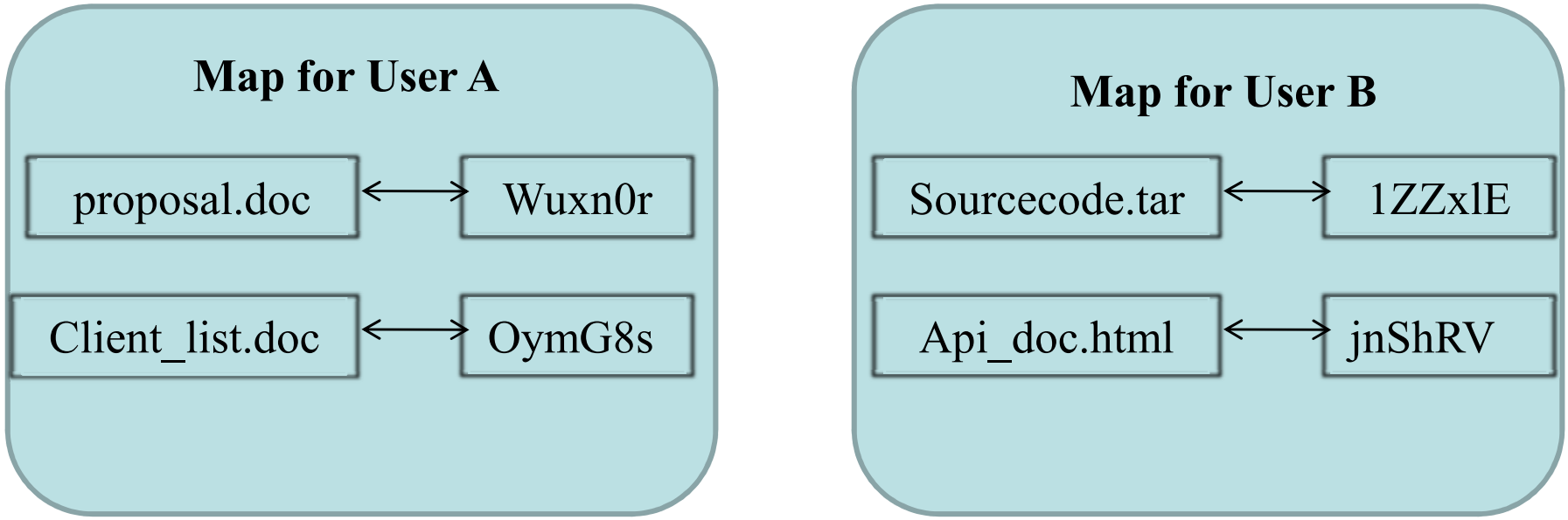
http://www.example.com/download.aspx?file=form.doc

getIndirectReference("Form.doc") = KwxiLa

http://www.example.com/download.aspx?file=KwxiLa

getDirectReference("KwxiLa") = Form.doc

Access control with 'per User AccessReferenceMap'



AccessReferenceMap Ref_map = new RandomAccessReferenceMap(set);
session.setAttribute("Ref_map", Ref_map);
Options:

- Integer Access Reference Map
- Random Access Reference Map

Executor Class

- Provides an interface for safe execution of files
- Reference implementation validates the path of the executable and throws and exception when any path manipulation is discovered
- Reference implementation encodes the user supplied parameters specific to the Operating System to escape any special characters
- Method:

  executeSystemCommand();

Escaping Special Characters in Windows with ^

```
E:\esapi>dir 1 | time/T
12:38 PM

E:\esapi>dir ^1^ ^|^ ^t^i^m^e^/^T
 Volume in drive E has no label.
 Volume Serial Number is

 Directory of E:\esapi\1

11/17/2008  12:36 PM    <DIR>          .
11/17/2008  12:36 PM    <DIR>          ..
11/17/2008  12:36 PM    <DIR>          1
               0 File(s)              0 bytes

 Directory of E:\esapi


 Directory of E:\esapi

File Not Found
```

Encoded '|' (pipe) symbol loses its 'metaness'

Randomizer Class

- Strong randomization was never this easy
- Reference implementation uses JCE
- Methods:
  - getRandomBoolean()
  - getRandomFilename(java.lang.String extension)
  - getRandomGUID()
  - getRandomInteger(int min, int max)
  - getRandomLong()
  - getRandomReal(float min, float max)
  - getRandomString(int length, char[] characterSet)

Encryptor Class

- Provides for the common crypto needs of the developer
- Reference implementation builds on JCE
- Hashing uses salts to defeat rainbow attacks and hashes over and over 1024 times to increase strength
- Provides a method of storing data in encrypted format along with an expiration time called 'Seal'.

Key Methods

- Decrypt(String ciphertext)
- Encrypt(String plaintext)
- Hash(String plaintext, String salt)
- Seal(String data, long timestamp)
- Sign(String data)
- Unseal(String seal)
- verifySeal(String seal)
- verifySignature(String signature, String data)

EncryptedProperties Class

- Secure extension to the properties class
- All values are encrypted before storage and decrypted when called for
- Methods:
  - getProperty(String key)

    Returns the decrypted property value
  - setProperty(String key, String value)

    Set the property value after encrypting it

Logger Class

- Reference implementation builds on the java logger class
- Log levels are implemented in to methods:
  - debug() - FINE
  - trace() - FINEST
  - info() - INFO
  - warning() - WARNING
  - error() - ERROR_LEVEL(Custom level)
  - fatal() - SEVERE
- Log level is set by default through SecurityConfiguration class (ESAPI v1.4)

Method template

- Two types of methods calls:

info(EventType type, boolean success, String message)

info(EventType type, boolean success, String message, Throwable throwable)

Event Type should be one of the following values:

- FUNCTIONALITY
- PERFORMANCE
- SECURITY
- USABILITY

Sample Log: (ESAPI v1.3)

Nov 23, 2008 1:26:34 PM null:IntrusionDetector IntrusionDetector
WARNING: SECURITY: Anonymous(80861)(unknown) -- Incorrect password provided for lavakumar
AuthenticationLoginException @ org.owasp.esapi.reference.DefaultUser.loginWithPassword(null:-1)

EnterpriseSecurityException Class

- This is the base class for all exceptions thrown by ESAPI
- Extends the java exception class by adding two types of messages:
  - User Message
    - Non-verbose message which will be returned to user.
    - This is sent to the base exception class
  - Log Message
    - Verbose message which will be automatically logged
    - This can act as the debug information for developers
- Solves the age old developer dilemma

Example

```
new EncryptionException("Decryption failed",
   "Decryption problem: " + e.getMessage(), e);
```

- User Sees: “Decryption failed”
- Developer Sees: "Decryption problem: Encryption block has wrong block type! “

Extensions

- AccessControlException
- AuthenticationAccountsException
- AuthenticationCredentialsException
- AuthenticationException
- AuthenticationHostException
- AuthenticationLoginException
- AvailabilityException
- CertificateException
- EncodingException
- EncryptionException
- EnterpriseSecurityException
- ExecutorException
- IntegrityException
- IntrusionException
- ValidationAvailabilityException
- ValidationException
- ValidationUploadException

IntrusionDetector Class

- Gives intrusion detection capability to the application
- Methods:
  - addException()
  - addEvent()
- If an exception or event exceeds the allowed quota then a predefined security action is taken
- Each exception automatically calls the addException method
- addException internally calls addEvent

Sample rules

Rule in actual syntax:

org.owasp.esapi.errors.ValidationException.count=10

org.owasp.esapi.errors.ValidationException.interval=10

org.owasp.esapi.errors.ValidationException.actions=log,logout

Literal Transalation:

If more than 10 input validation exceptions are detected in a period of 10 seconds then log the event and logout the user.

SecurityConfiguration Class

- Provides a single point of access to the security configuration settings of the application
- Stores values like:
  - Number of login attempts allowed before acc lockout
  - Whitelists for input validation
  - Algorithm to use for hashing and encryption
  - Master password to be used for all encryption
  - Intrusion detection rules
  - Logging level etc
- Reference implementation stores all config data in 'ESAPI.properties file' and reads from it

Key Methods

- getAllowedFileExtensions()
- getAllowedFileUploadSize()
- getAllowedLoginAttempts()
- getDigitalSignatureAlgorithm()
- getEncryptionAlgorithm()
- getHashAlgorithm()
- getMasterPassword()
- getMasterSalt()
- getQuota(String eventName)
- getRandomAlgorithm()
- getRememberTokenDuration()
- getResourceDirectory()
- getResponseContentType()
- getUsernameParameterName()
- getPasswordParameterName()
- setResourceDirectory(String dir)

Validator Class

- Input validation is one half of the protection needed against Injection and parameter manipulation attacks
- Black-list based filters will always be bypassed
- White-list based filters can sometimes be bypassed by clever encoding attacks like double encoding
- Validator's approach:
  - White-list based filters
  - Data canonicalization before being passed to filters
  - Detect double-encoding and throw an exception

ESAPI's Approach to Input validation

Allow Good Input && Block Bad Input

Whitelists for:

- Filenames
- HTTP parameters
- Email Ids
- IP addresses etc…

Double Encoding

Default white-lists:

- **SafeString**=^[\p{L}\p{N}.]{0,1024}$
- **Email**=^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\\.[a-zA-Z]{2,4}$
- **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]?)$
- **URL**=^(ht|f)tp(s?)\\:\\/\\/[0-9a-zA-Z]([-.\\w]*[0-9a-zA-Z])*(:(0-9)*)*(\\/?)([a-zA-Z0-9\\-\\.\\?\\,\\:\\'\\/\\\\\\+=&%\\$#_]*)?$
- **CreditCard**=^(\\d{4}[-]?){3}\\d{4}$
- **SSN**=^(?!000)([0-6]\\d{2}|7([0-6]\\d|7[012]))([-]?)(?!00)\\d\\d\\3(?!0000)\\d{4}$
- **AccountName**=^[a-zA-Z0-9]{3,20}$
- **SystemCommand**=^[a-zA-Z\\-\\/]{0,64}$
- **RoleName**=^[a-z]{1,20}$
- **Redirect**=^\\/test.*$
- **HTTPParameterName**=^[a-zA-Z0-9_]{0,32}$
- **HTTPParameterValue**=^[a-zA-Z0-9.\\-\\/+=_]*$
- **HTTPCookieName**=^[a-zA-Z0-9\\-_]{0,32}$
- **HTTPCookieValue**=^[a-zA-Z0-9\\-\\/+=_]*$
- **HTTPHeaderName**=^[a-zA-Z0-9\\-_]{0,32}$
- **HTTPHeaderValue**=^[a-zA-Z0-9()\\-=*\\.\\?;,+\\/:&_]*$
- **FileName**=^[a-zA-Z0-9.\\-_]{0,255}$
- **DirectoryName**=^[a-zA-Z0-9.-_]{0,255}$

Methods

- assertIsValidHTTPRequest()
- assertIsValidHTTPRequestParameterSet()
- assertValidFileUpload()
- getValidCreditCard()
- getValidDate()
- getValidDirectoryPath()
- getValidDouble()
- getValidFileContent()
- getValidFileName()
- getValidInput()
- getValidInteger()
- getValidListItem()
- getValidNumber()
- getValidPrintable()
- getValidRedirectLocation()
- getValidSafeHTML()
- isValidCreditCard()
- isValidDate()
- isValidDirectoryPath()
- isValidDouble()
- isValidFileContent()
- isValidFileName()
- isValidFileUpload()
- isValidHTTPRequest()
- isValidHTTPRequestParameterSet()
- isValidInput()
- isValidInteger()
- isValidListItem()
- isValidNumber()
- isValidPrintable()
- isValidPrintable()
- isValidRedirectLocation()
- isValidSafeHTML()

Adding and using custom white-list

- A custom white-list can be added by making an entry in to the 'ESAPI.properties' file(or any other place which the SecurityConfiguration class would refer)

  Eg:

  Sample Employee ID - LK0001

  Corresponding white-list: **Validator.EmpId= ^[A-Z]{2}[0-9]{4}$**

- Custom white-lists can be used for validation with the 'getValidInput' or 'isValidInput' methods

  Eg:

  isValidInput("Employee ID", "MN1002", "EmpId", 5, false**);**

Method types

- 'isValid' methods

  These methods return a Boolean indicating if the input is valid or not

- 'getValid' methods

  These methods thrown a 'validationexception' when input is not valid

  Each method of this type is overloaded with a variant which adds the 'validationexception' to an 'ValidationErrorList' taken in as a parameter

The magic of AntiSamy library

- ESAPI uses the AntiSamy library to accept safe HTML
- AntiSamy has a white-list of safe HTML tags and attributes and only allows those. This white-list can be tuned to make it more restrictive
- Methods:
  - isValidSafeHTML()

    This function checks if the input HTML is safe
  - getValidSafeHTML()

    This function removes all unsafe content from the input HTML and returns safe HTML.

Encoder Class

- Output encoding is the second half of the protection against injection attacks
- Provides canonicalization support with the 'canonicalize' method
- Has codecs for :
  - Base64
  - CSS
  - HTMLEntity
  - JavaScript
  - MySQL
  - Oracle
  - Percent Encoding
  - Unix Shell escaping
  - VBScript
  - Windows Shell escaping

Methods

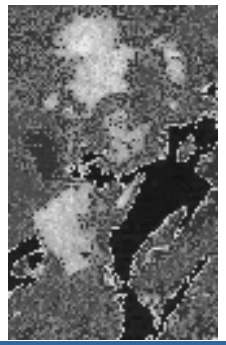
- canonicalize()
- decodeFromBase64()
- decodeFromURL()
- encodeForBase64()
- encodeForCSS()
- encodeForDN()
- encodeForHTML()
- encodeForHTMLAttribute()
- encodeForJavaScript()
- encodeForLDAP()
- encodeForSQL()
- encodeForURL()
- encodeForVBScript()
- encodeForXML()
- encodeForXMLAttribute()
- encodeForXPath()
- normalize()

ESAPI Implementation Overview

Class	Extension to Reference Implementation	Implementation priority
Encoder	None	High
Validator	Minimal	High
Encryptor	None	Medium
Randomizer	None	High
Executor	None	High
AccessReferenceMap	Minimal	High
AccessController	Moderate	Medium
User	Moderate	Medium
HTTPUtilities	None	Medium
Authenticator	Substantial	Medium
SecurityConfiguration	Minimal	High
Logger	Substantial	Low
IntrusionDetector	Moderate	Low
EncryptedProperties	Minimal	Moderate

You might be thinking..

I was waiting for this day…now I can finally get rid of my penetration tester
I have ESAPI now, I don't need that pest
Imagine the cost saving!!!
Moreover the developers don't like him anyway



HOLD IT!!

- Penetration testers aren't paid much so you wont be saving a lot ☺
- ESAPI is only an API, if not used properly your applications will have holes
- If your are not careful in extending the reference implementation you could actually make it vulnerable
- ESAPI cannot do much about the logic and design flaws in your application
- Don’t make any changes to your existing application audit practice

Where can I find out more?

- Start off with Jeff's excellent talk at OWASP NY, the video is available at http://www.owasp.tv
- The API is documented in detail, refer the HTML documentation
- Read the source code of reference implementation, clearly written, very easy to follow.
- Invite me for beer ☺

Thanks for still sticking around ☺

Got questions ???
You gotta ask me nicely