GOTHAM
DIGITAL • SCIENCE

Web Hacking

Day 2

NYU:poly
POLYTECHNIC INSTITUTE OF NYU

Web Hacking

Session Outline

- Same-Origin Policy
- Authentication & Authorization
- Cross-Site Request Forgery
- Web Services
- Cryptography

And why XSS is so important

SAME-ORIGIN POLICY

DOM Same-Origin Policy

Basic Premise:

JavaScript executing in context of one document should not be allowed to access context of another document, unless:

- **protocol**, **hostname** and **port** all match!

- This tuple defines a document's **Origin**

Same-Origin Policy Check

http://example.com/a/

Accessed Document	Outcome	Reason
http://example.com/b/		
http://www.example.com/		
https://example.com/b/		
http://example.com:81/		

Same-Origin Policy Check

http://example.com/a/

Accessed Document	Outcome	Reason
http://example.com/b/	✓	
http://www.example.com/		
https://example.com/b/		
http://example.com:81/		

Same-Origin Policy Check

http://example.com/a/

Accessed Document	Outcome	Reason
http://example.com/b/	✓	
http://www.example.com/	✗	**Different host**
https://example.com/b/		
http://example.com:81/		

Same-Origin Policy Check

http://example.com/a/

Accessed Document	Outcome	Reason
http://example.com/b/	✓	
http://www.example.com/	✗	**Different host**
https://example.com/b/	✗	**Different protocol**
http://example.com:81/		

Same-Origin Policy Check

http://example.com/a/

Accessed Document	Outcome	Reason
http://example.com/b/	✓	
http://www.example.com/	✗	**Different host**
https://example.com/b/	✗	**Different protocol**
http://example.com:81/	✗	**Different port**

Same-Origin Policy Check

http://example.com/a/

Accessed Document	Outcome	Reason
http://example.com/b/	✓	
http://www.example.com/	✗	**Different host**
https://example.com/b/	✗	**Different protocol**
http://example.com:81/	✗	**Different port**

- IE doesn't always observe port numbers

Cross-Domain Policies

Extends SOP beyond a document's origin

- Permit applets originating from another domain access to resources
- Permit issuing arbitrary HTTP requests with whitelisted headers

CORS

Cross-Origin Resource Sharing

- Browser allows XMLHttpRequest's to access response data return from cross-origin requests when:
  - Response contains **Access-Control-Allow-Origin** header
  - Request's Origin value is defined in set

Wildcard Domains

Using wildcard (“*”) policies is ill-advised

- Expose content on your domain to script access from ANY/ALL origins

Same-Origin Policy

DEMO

Cross-Site Request Forgery

CSRF

Cross-Site Request Forgery

Tricking victims' browsers into performing unsuspecting actions

- Server doesn't verify request was initiated from the expected client-side origin
- Browser naively submits credentials when attempting to retrieve resources

Exploiting CSRF

1. User, **Dan** logs into his account at bank.com
2. In another tab, **Dan** visits a site that sources an image from:

   ```
   <img src="http://bank.com/xfer.do?
   frmAcct=Dan&toAcct=Joe&amt=100000" />
   ```

3. Dan's ***browser*** sends a GET request for the "image"
4. Dan unknowingly just transferred 100k into Joe's account!

Cross-Site Request Forgery

DEMO

Other Examples

CSRF vulnerabilities are everywhere

- Can be used to exploit otherwise “admin-only” vulnerabilities
  - Router admin pages, etc
- A simple mitigation, often hard to implement
  - Include secret user/session specific value with request

Gaining and Elevating Privileges

AUTHENTICATION & AUTHORIZATION

Authentication

Applications can verify identity based on:

- Something the user KNOWS
  - Password, Passphrase, PIN, etc
- Something the user HAS
  - Smartcard, Security Token, Mobile Device
- Something the user IS or DOES
  - Fingerprint, Voice Recognition

Common Authentication Methods

- HTTP Authentication
- Form-Based
- Kerberos/NTLM
- Single Sign-On (SSO)
- Certificate Based

Know Your Authentication Scheme

Basic Authentication

- Credentials are Base64 encoded
  - in the format: *username:password*
- Each subsequent request includes credentials within the **Authorization** HTTP request header

```
GET /home.jsp HTTP/1.1
Host: www.acme.com
User-Agent: Mozilla/4.0
Authorization: Basic YWRtaW46YWRtaW4=
```

Decodes to:

admin:admin

Bruteforcing Usernames & Passwords

Common way to break into applications

- Pay close attention to response contents
- Is password strength and complexity enforce?
- Be wary of account lockout thresholds
  - Usually leave until the end of your testing
  - Average 5 invalid attempts
  - If no lockout, automate! (Hydra, Brutus, etc)

Login Errors

So what's the difference?

```
if auth_result == USER_DOES_NOT_EXIT:
  return "Incorrect username or password, "
         "please try again."

elif auth_result == INVALID_PASSWORD:
  return "Incorrect username or password, "
         "please try again"
```

Bypassing Authentication

- Predicting session tokens
  - What is the character composition of cookie?
- Session hijacking via fixation/trapping
  - Same session cookie used pre and post authentication?
- Exploiting an injection flaw in login routine
  - SQL Injection

Bypassing Authentication (Client-Side)

- Does application set persistent authentication cookies?
  - Look for “Remember Me” functionality on login page
  - Look at *Expires* attribute of **Set-Cookie** header

- “Back” button to steal cached credentials
  - Browser may prompt user to resubmit form
  - Basic authentication, credentials stored in browser memory

Authentication Testing

Is more than just logging in

- Ancillary authentication functions
  - Password Resets
  - Remember Me
  - Registration
- Logout
- Session Management

Weak Authentication

Does application authenticate requests across all resources?

- Issue direct requests to resources (forceful browsing)
  - Guess common file names
- Inventory resources authenticated & request anonymously
  - Authenticate, crawl through UI, and record requests/responses
  - Re-issue those requests unauthenticated and diff responses

Authorization

System of determining whether a user is allowed to access a particular resource

- Role-based authorization
- Access decisions can be made at the:
  - Resource-level
  - Function/Method-level
  - Record-level

Authorization Attacks

Elevation of Privileges

- Vertical
  - Elevate to a More Privileged User
  - Access Restricted Functionality
  - Edit Records Intended to be Read Only
- Horizontal
  - Access Another User's Data

Parameter Manipulation

aka Insecure Direct Object Reference…

Example: DB record id's exposed to user

Normal URL	Exploit URL
/AccountInfo.aspx?AcctId=0396280	/AccountInfo.aspx?AcctId=0396290

- Weak access control at the record-level
- Difficult for automated scanners to detect
- Have context – know what to manipulate!

Parameter Manipulation

DEMO

URL and Function-Level Authorization

Failure to restrict URL access

- Protect sensitive functionality by
  - disabling the display of links, button, URLs, etc
  - hidden URLs or parameters
- Forceful Browsing is a common attack technique
  - Typically results in vertical escalation
  - Administrative interfaces

Hitting up the backend

ATTACKING WEB SERVICES

XML Web Services

Facilitate Machine to Machine interaction

- Usually implemented as middleware, though sometimes called directly by the client
- Often implemented using *Simple Object Access Protocol* (SOAP)
- Request and Response structure defined by *Web Services Definition Language* (WSDL)

Attacking Web Services

Step 1: Locate the Web Service Endpoint

- Pay close attention to proxy logs
- Look for common web service endpoints
  - .asmx, .svc
  - /axis, /axis2

Attacking Web Services

Step 2: Obtain Metadata

- Try appending **?WSDL** or **.wsdl** to endpoint URL

<portType>: Operations performed by the Web Service

<message>: Messages used by the Web Service

<types>: Datatypes used by the Web Service

<binding>: Protocol used by the Web Service

Attacking Web Services

Step 3: Invoke the Web Service

- Issue SOAP requests directly to end point
  - SoapUI
- Fuzz inputs just like any other parameter
  - Same vulnerabilities apply, in addition to some others

Exploiting XML Parsers

Web services often vulnerable to common attacks on XML parsers

- Entity Expansion Attacks
  - Denial of Service against XML parser
  - Infinite recursion occurs during parsing
- XML External Entity attacks
  - Information disclosure to almost anything

XML External Entity Attacks

1. Define an XML entity in the DTD
2. Reference defined entity in XML body

```
<!DOCTYPE test [<!ENTITY x3 SYSTEM "/etc/passwd">]>
<body>
    <e1>&x3;</e1>
</body>
```

Parser reads `/etc/passwd` contents into **e1**

Exploiting XML Parsers

DEMO

Just sprinkle a little for security++

CRYPTOGRAPHY

Crypto? Not so fast!

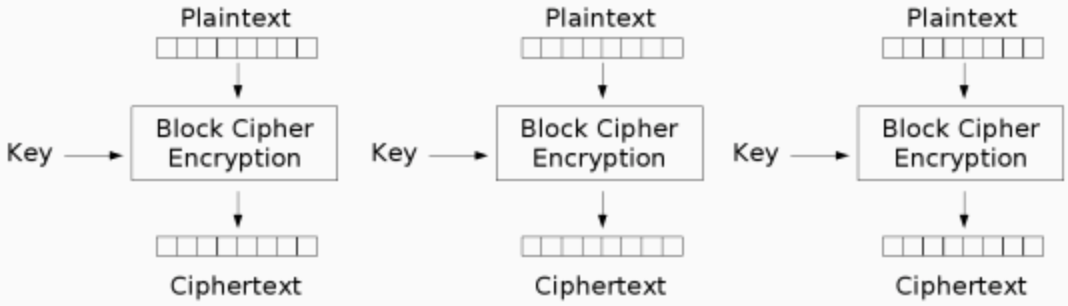
Not always as complex as you think…

- Crypto often presents a stumbling block for many testers
- Often data is just encoded in base64, hex, etc.
- Other times, it's just compressed (Gzip)
- Regardless, it is ALWAYS worth investigating

Block Cipher Encryption

Electronic Code Book (ECB) Mode

- Simplest (and often the default) block cipher mode
- Message is split into blocks and each is encrypted separately

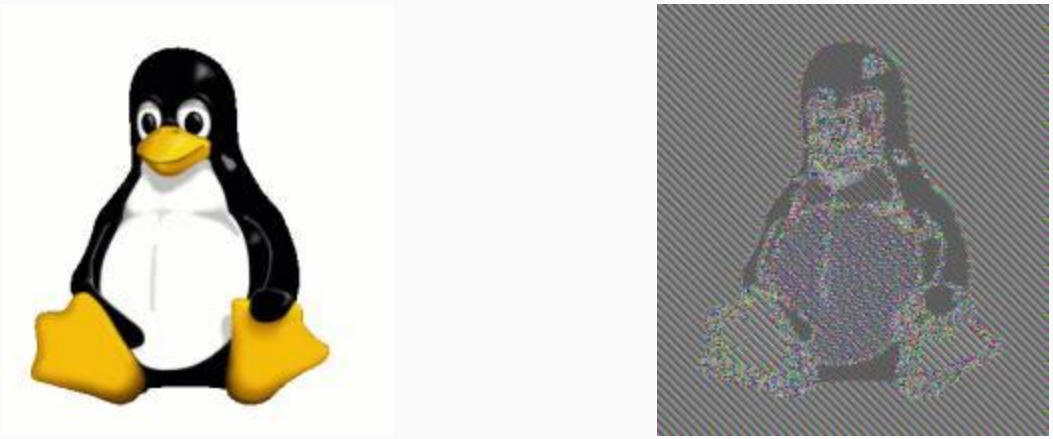


Electronic Codebook (ECB) mode encryption

Block Cipher Encryption

Disadvantage of ECB Mode

- Identical plaintext block encrypts to identical cipher text block



A pixel-map version of the image on the left was encrypted with ECB mode to create the image on the right

Stimulus / Response Testing

```
A x 1     A

          73D4ABC882C52727
```

Stimulus / Response Testing

```
A x 1       A
            73D4ABC882C52727
A x 2       AA
            17AC5DEB5859C3F8
```

Stimulus / Response Testing

```
A x 1      A
           73D4ABC882C52727
A x 2      AA
           17AC5DEB5859C3F8
A x 3      AAA
           8B44E406462A468B
```

Stimulus / Response Testing

```
A x 1     A
          73D4ABC882C52727
A x 2     AA
          17AC5DEB5859C3F8
A x 3     AAA
          8B44E406462A468B
A x 7     AAAAAAA
          880FB2EF51A7FC14
```

Stimulus / Response Testing

A x 1	A	
	73D4ABC882C52727	
A x 2	AA	
	17AC5DEB5859C3F8	
A x 3	AAA	
	8B44E406462A468B	
A x 7	AAAAAAA	
	880FB2EF51A7FC14	
A x 8	AAAAAAAA	
	578A75B73BBB948F	78A4D70C0F6F3FF2

Stimulus / Response Testing

A x 1	A	
	73D4ABC882C52727	
A x 2	AA	
	17AC5DEB5859C3F8	
A x 3	AAA	
	8B44E406462A468B	
A x 7	AAAAAAA	
	880FB2EF51A7FC14	
A x 8	AAAAAAAA	
	578A75B73BBB948F	78A4D70C0F6F3FF2
A x 9	AAAAAAAA	A
	578A75B73BBB948F	73D4ABC882C52727

Stimulus / Response Testing

A x 1	A	
	73D4ABC882C52727	
A x 2	AA	
	17AC5DEB5859C3F8	
A x 3	AAA	
	8B44E406462A468B	
A x 7	AAAAAAA	
	880FB2EF51A7FC14	
A x 8	AAAAAAAA	
	578A75B73BBB948F	78A4D70C0F6F3FF2
A x 9	AAAAAAAA	A
	578A75B73BBB948F	73D4ABC882C52727

Stimulus / Response Testing

A x 1	A	
	73D4ABC882C52727	
A x 2	AA	
	17AC5DEB5859C3F8	
A x 3	AAA	
	8B44E406462A468B	
A x 7	AAAAAAA	
	880FB2EF51A7FC14	
A x 8	AAAAAAAA	
	578A75B73BBB948F	78A4D70C0F6F3FF2
A x 9	AAAAAAAA	A
	578A75B73BBB948F	73D4ABC882C52727
A x 10	AAAAAAAA	AA
	578A75B73BBB948F	17AC5DEB5859C3F8

Stimulus / Response Testing

A x 1	A	
	73D4ABC882C52727	
A x 2	AA	
	17AC5DEB5859C3F8	
A x 3	AAA	
	8B44E406462A468B	
A x 7	AAAAAAA	
	880FB2EF51A7FC14	
A x 8	AAAAAAAA	
	578A75B73BBB948F	78A4D70C0F6F3FF2
A x 9	AAAAAAAA	A
	578A75B73BBB948F	73D4ABC882C52727
A x 10	AAAAAAAA	AA
	578A75B73BBB948F	17AC5DEB5859C3F8

Live Demonstration

ECB BLOCK SWAPPING DEMO

Attacking Randomness

Good randomness is a vital part of many cryptographic operations

- Two common attacks against a PRNG:
  a. PRNG state is reconstructed from its output
  b. Same PRNG state is used more than once

Reconstructing PRNG State

```
>>> rnd = Random(seed)
>>> rnd.getrandbits(8)
216L
…

>>> rnd.getrandbits(8)
227L
>>> rnd.getrandbits(8)
107L
>>> rnd.getrandbits(8)
???
```

- If *seed* or the internal state of the PRNG is ever exposed, you can determine what the next value will be
- A PRNG can never recover back to a secure state

seed = 0

Insecure Seeding

Statistically random != secure random

- If a PRNG is seeded with a value the attacker can influence, the state of the PRNG is likely compromised
- Use of an insecure PRNG results in insecure output

Insecure Seeding

Seed Race Condition Attacks

- System clock often used to seed PRNG
- Submit 10's or 100's of requests at a time
  - Seed a PRNG with the same system clock
  - Output will be the same

Other Crypto Vulnerabilities

Some other areas to explore

- Padding Oracles
  - http://bit.ly/twluHj *(blog.gdssecurity.com)*
- MD5/SHA-1 Length Extension
  - http://bit.ly/1g3mz8 *(netifera.com, PDF)*
- Stream ciphers and key-reuse

Web Hacking – Day 2

WRAP UP

Further Reading

- The Web Application Hacker's Handbook
- The Tangled Web
- SQL Injection Attacks and Defenses
- Cryptography Engineering