CIS 551 / TCOM 401

Computer and Network Security

Spring 2006

Lecture 23

Announcements

- Project 2 has been graded
  - You should have received your grades by e-mail
  - Class average was: 83

- Talk Today:
  - Kamin Whitehouse, U.C. Berkeley
  - "Dealing with Real Sensors and Environments for Sensor Network Applications"
  - 3:00 at Wu & Chen

Plan for today

- Wrap up viruses/worms
  - Polymorphic viruses
- Web Security

Assumptions Made by Earlybird

- Invariant Content
- Dispersion of addresses
- Frequent occurrences of suspicious content

- Attacker: Let's violate one of these assumptions…

Polymorphic Viruses/Worms

- Virus/worm writers know that signatures are the most effective way to detect such malicious code.

- Polymorphic viruses mutate themselves during replication to prevent detection
  - Virus should be capable of generating many different descendents
  - Simply embedding random numbers into virus code is not enough

Strategies for Polymorphic Viruses

- Change data:
  - Use different subject lines in e-mail
- Encrypt most of the virus with a random key
  - Virus first decrypts main body using random key
  - Jumps to the code it decrypted
  - When replicating, generate a new key and encrypt the main part of the replica
- Still possible to detect decryption portion of the virus using virus signatures
  - This part of the code remains unchanged
  - Worm writer could use a standard self-decompressing executable format (like ZIP executables) to cause confusion (many false positives)

Advanced Evasion Techniques

- Randomly modify the *code* of the virus/worm by:
  - Inserting no-op instructions: subtract 0, move value to itself
  - Reordering independent instructions
  - Using different variable/register names
  - Using equivalent instruction sequences:

    y = x + x vs. y = 2 * x
  - These viruses are sometimes called "metamorphic" viruses in the literature.

- There exist C++ libraries that, when linked against an appropriate executable, automatically turn it into a metamorphic program.

- Sometimes vulnerable software itself offers opportunities for hiding bad code.
  - Example: ssh or SSL vulnerabilities may permit worm to propagate over encrypted channels, making content filtering impossible.
  - If IPSEC becomes popular, similar problems may arise with it.

Other Evasion Techniques

- Observation: worms don't need to scan randomly
  - They won't be caught by internet telescopes
- *Meta-server* worm: ask server for hosts to infect (e.g., Google for "**powered by php**")
- *Topological* worm: fuel the spread with local information from infected hosts (web server logs, email address books, config files, SSH "known hosts")
  - No scanning signature; with rich inter-connection topology, potentially very fast.
- Propagate slowly: "trickle" attacks
  - Also a very subtle form of denial of service attacks

Witty Worm

- Released March 19, 2004.
- Single UDP packet exploits flaw in the *passive analysis* of Internet Security Systems products.
- “Bandwidth-limited” UDP worm like Slammer.
- Vulnerable pop. (12K) attained in 75 minutes.
- Payload: *slowly corrupt random disk blocks*.

Witty, con't

- Flaw had been announced the *previous day*.
- Telescope analysis reveals:
  - Initial spread seeded via a *hit-list*.
  - In fact, targeted a U.S. military base.
  - Analysis also reveals "Patient Zero", a European retail ISP.
- Written by a Pro.

Web Security

- What security concerns are there on the web?

- Class answers:
  - Links can lie -- may not take you where you think they do
  - Many more malicious users (authentication is a problem)
  - Cookies -- can reveal private information, questions of their security
  - Spyware/Malware -- mobile code
  - Eavesdropping / keylogger
  - Knowing what's going on -- configuration management
  - Embedded code / scripts / flash / ActiveX / … executable content
  - Authorization, etc. -- access control
  - Profile stealing
  - Trusting remotes sites with your confidential information

OWASP.org Top 10

- Open Web Application Security Project

1. Unvalidated Input
2. Broken Access Control
3. Broken Authentication
4. Cross Site Scripting
5. Buffer Overflows
6. Injection Flaws
7. Improper Error Handling
8. Insecure Storage
9. Application Denial of Service
10. Insecure Configuration Management

HTTP

- See http://www.w3.org/Protocols for standards document
- Request/response style protocol
- Client Requests:
  - GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT, OPTIONS
  - Plus request arguments and body content
- Server Response:

  <HTTP-Version> <Status-Code> <Reason-Phrase>
  - 1xx: Informational - Request received, continuing process
  - 2xx: Success - The access was understood and accepted
  - 3xx: Redirection - Further action must be taken to process request
  - 4xx: Client Error - Request contained bad syntax or could not be fulfilled (e.g. 404 Not Found)
  - 5xx: Server Error - Server failed (e.g. 500 Internal Server Error)

Example HTTP Request/Response

```
GET /get/a/URL HTTP/1.1
Referrer: http://another.host/their/URL
Connection: Keep-Alive
Cookie: Flavor=Chocolate Chip
User-Agent: Mozilla/2.01 (X11; I; BSD/0S 2.0 i386)
Host: some.random.host:80
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
------------------

HTTP/1.0 200 OK
Set-Cookie: Flavor=peanut-butter; path=/
Date: Thursday, 13-Apr-06 11:34:23 EST
Server: NCSA/1.7
MIME-version: 1.0
Content-type: text/html
<html>
…
```

URIs are complicated

- Uniform Resource Identifier (URI)
  a.k.a. URL
- URI is an extensible format:

  URI ::= scheme ":" hier-part ["?" query] ["#" fragment]

Examples:

- ftp://ftp.foo.com/dir/file.txt
- http://www.cis.upenn.edu/
- ldap://[2001:db8::7]/c=GB?objectClass?one
- tel:+1-215-898-2661
- http://www.google.com/search?client=safari&rls=en&q=foo&ie=UTF-8&oe=UTF-8

URI's continued

- Confusion:
  - Try going to www.whitehouse.org or www.whitehouse.com (instead of www.whitehouse.gov)
- Obfuscation:
  - Use IP addresses rather than host names: http://192.34.56.78
  - Use Unicode escaped characters rather than readable text http://susie.%69%532%68%4f%54.net

Maintaining State

- HTTP is a stateless protocol
  - Server doesn't store any information about the connections it handles (each request is treated independently)
  - Makes it hard to maintain session information
- Encode state in the URL:
  - …/cgi-bin/nxt?state=-189534fjk
  - Used commonly on message boards, etc. to track thread
- Use HIDDEN input fields
  - When user fills in web forms, the POST request gives server the data
  - You can embed state in invisible "input" fields
- Cookies
  - Store data on the client's machine

Cookies (Client-side state)

- Server can store cookies on the client machine by issuing:

```
Set-Cookie: NAME=VALUE; [expires=DATE;]
[path=PATH;] [domain=DOMAIN_NAME;]
[secure]
```

- Domain and Path restrict the servers (and paths on those servers) to which the cookie will be sent
- The "secure" flag says that the cookie should only be sent over HTTPS

Cookies (cont'd)

- When the client requests a URL from a server, the browser matches the URL against all cookies on the client.
- If they match, then the client request includes the line:

  `Cookie: NAME1=STRING1; NAME2=STRING2;…`

- Notes:
  - New instances of cookies overwrite old ones
  - Clients aren't required to purge expired cookies (though they shouldn't send them)
  - Cookies can be at most 4k
  - To delete a cookie, the server can send a cookie with expires set to a past date
  - HTTP proxy servers shouldn't cache Set-cookie headers…

Scripts & Mobile Code

- Client side: embedded in HTML sent to the client
  - Java Applets, JavaScript, ActiveX, Flash
- Server Side: receive & process arguments from forms filled in by blient
  - CGI "Common Gateway Interface"
    - Allows server to call code written in any language, commonly C or Perl
    - Code typically stored in /cgi-bin directory
  - PHP "PHP Hypertext Preprocessor"
    - Embed dynamically generated content into HTML pages

Example PHP

```
<html>
  <head>
  <title>PHP Test</title>
  </head>
  <body>
   <?php echo '<p>Hello World</p>'; ?>
  </body>
</html>
```

Cross Site Scripting (XSS)

- Consider the following scenario:
  - You click on a URL: http://www.cis.upenn.edu/~stevez/foo.html
  - What happens? Server responds:

> **Not Found**
> The requested URL /~stevez/foo.html was not found on this server.
>
> *Apache/1.3.33 Server at www.cis.upenn.edu Port 80*

- What's the problem?

XSS continued

- Suppose that the malicious URL contained HTML tags for an embedded script:

  http://www.cis.upenn.edu/~stevez/<script>alert('hello')</script>

- If the server generates the error page naively, it might accidentally include the script in the page displayed to the client!
  - (Fortunately, CETS here at Penn gets this right…)

XSS

- These techniques can be used to steal cookies, redirect users to bogus web pages, grab data entered by user.

- Other tricks:
  - Attackers can encode malicious part of the URL to make it harder to detect (e.g. use Unicode)
  - Not all attacks need the "<" and ">" symbols

- What can be done?
  - Validate URLs at the server side
  - Rewrite "problematic" inputs to HTML entity codes:
    < becomes &#60
    > becomes &#62
    …

Broken Access Control

- Insecure ID's (security through obscurity)
- Easily skipped security checks -- e.g. user enters "protected" URL directly
- Path Traversal: URLs or other data that contain relative paths
  - E.g. ../../some_dir/some_file
- Incorrectly used File Permissions

Insecure Storage

- Failure to encrypt critical data
- Insecure storage of keys, certificates, and passwords
- Improper storage of secrets in memory
- Poor sources of randomness
- Poor choice of algorithm
- Attempting to invent a new encryption algorithm