

Love Virus — CGI 3

"I Love You" Virus

Key: executable content

eg: .exe .pl .vbs .js

Run "as you"

In the simplest case, when you double click on a program on your machine, it runs on your behalf. Inheriting your permission to access whatever you can access.

How Did This Happen?

Corporate Cultures

1. Microsoft

+Integration

+Market driven

customer features vs. engineer features

+Network effect

-Security

-Standards

2. Sun

+Security

-Usability

Engineer driven vs. customer driven

Microsoft: Product Integration

MS integrates products across domains -- putting a feature in product X that works specially with product Y and Z.

e.g. Exchange, IE, MS 2000,

Windows 2000 -- IIS -- Exchange -- Outlook -- Front Page - Office -- Access
Put in features where these talk to each other in a MS specific way

Exchange email/meeting server is a success story -- many companies
wanted to get an email solution. Exchange does this + meetings (no open
source competition really) and is all integrated = irresistible (until now).

Good Interpretation

The integration is a genuine feature -- if I buy into the whole MS suite, then
I can click "publish" in my MS Office doc, and interacts with Front Page
and IIS.

This works to the benefit of consumers

Bad Interpretation

This also happens to be a great way to force out competitors which is
ultimately bad for consumers

Note: delay the monopoly penalty

BTW, for this reason it's MS's interest to delay the trial penalty as long as
possible. Give the tyings as much time as possible to build domination in
the various niches. After you have domination, then breakup etc. is ok
because the network effect keeps you winning. Just, for example, be sure
to lock up the browser market with tying before the break up. Prediction:
MS makes the legal phase go as slowly as possible.

How To Do the Linking?

Text file: no

Some standard: no

VBScript: YES

Good: capable -- MS has made the +integrated -security tradeoff

Bad: complex to clone -> not replaceable

Result:

Lots of MS components linked with VBscript

Pro: neat features

Con: hard to replicate

Con: executable content = security problems

Bad Solutions

Virus "in transit" detection

Way too slow for the new, hip email viruses

Only from trusted sources

Useless advice once your friend has been taken over

"Are you sure you want to download X" dialog

Any dialog that the user sees more than 5x a week doesn't get read any more.

It would be helpful if the "alert" system could make a very strong statement about executable content vs. JPEG etc.

Good Solutions — Safe

Code Signing
"Sandbox"

Current Results

http://dailynews.yahoo.com/h/nm/20000504/wr/virus_love_11.html

...Microsoft Corp., for its part, said the virus was not indicative of any particular vulnerability associated with Microsoft Outlook.. ``Viruses are really an industry-wide issue," said Scott Culp, program manager for Microsoft's security response center. "They can be written for any platform. They can be written to use a variety of e-mail clients.

``In this case the virus author chose to target Outlook probably because it gave him better reach," he said. ``There isn't a security vulnerability in Outlook involved in this at all," Culp said.

Lying

I think the above quote is pretty amazing. MS is either pretty deeply in denial, or are just lying. This virus is 100% due to MS's lax attitude about executable content -- that was the cost of the way MS has done integration.

Future

Safe Executable Content

I expect this episode to maybe turn MS around on executable content. There are technical fixes, although they will involve tradeoffs that make integration work a little less smoothly. The tradeoff is worth it. ("Tradeoff" is my favorite word when it comes to looking at design decisions.)

CGIParse

ENV Variables

Information is passed to the CGI in environment variables -- access as
\$ENV{ } in Perl.

QUERY_STRING

REQUEST_METHOD

Get -- use query string
Post -- read from stdin

pi=3.14&target=foo%32bar

CGIParse.pl

```
#!/usr/bin/perl
## CGIParse.pl

# These are two subroutines provided to demonstrate very basic CGI in Perl
# for CS193i -- they are free for all uses. Nick Parlante.
# For real work, you should use the CGI.pm module which does this and more.

# 1) ReportError(str) print out an error message as HTML and exit
# 2) ExtracBindings -- read in the HTTP request, and build and return
#    an associative array of the attribute/value bindings of the request.

# Takes a string describing the error, prints it out in HTML to STDOUT,
# and exits.
# My strategy here is that I send error descriptions back to the client
# rather than dumping them to STDERR on the host. This sends HTML, so
# in effect it assumes that the "content-type: text/html" has already
# been sent. For that reason, the client script should send the HTTP
# header before before doing things which could fail.
#####
sub ReportError {
    my($errString) = @_;
    print "<h1>Error</h1>\n$errString\n";
    print $trailer;
    exit(0);      # could exit(1) + print to stderr to write to server err log
}

my($request_method, $content_length);

# Read a HTTP request from the environment vars and STDIN
# return the result in an associative dictionary with one entry
# for each key/value pair sent from the client.
# Does not support more than one value per key.
# May err out if there is an error in the request.
# Supports GET, POST, and a testing mode where it reads
# the query string out of $ARGV[0].
```

```
#####
sub ExtractBindings {

    my($encoded);
    $main::request_method = $ENV{"REQUEST_METHOD"}; # These are global, but that
    $main::content_length = $ENV{"CONTENT_LENGTH"}; # might not be all bad.

# If the method is the empty string, figure we're being invoked
# from the shell for testing and so use the string from ARGV[0]
    $encoded = "";
    if (!$main::request_method) {
        $encoded = $ARGV[0];
    }
    elsif ($main::request_method eq "POST") {
        read(STDIN, $encoded, $content_length); # POST => read from STDIN
    } elsif ($main::request_method eq "GET") {
        $encoded = $ENV{"QUERY_STRING"}; # GET = > read from ENV variable
    } else {
        ReportError("Unrecognized CGI method '$main::request_method'.");
    }

# The following is the standard hack for decoding the bindings
    my(@pairs, $pair, $key, $value, %bindings);
    %bindings = ();

# 1) Find the pairs based on the &
    @pairs = split(/\&/, $encoded);

# 2) For each pair
    foreach $pair (@pairs) {
        # 3) separate at the '='
        ($key, $value) = split(=/, $pair);
        # 4) Decode the '%20' chars
        $key =~ s/%([\da-fA-F][\da-fA-F])/pack("C", hex($1))/ge;
        $value =~ s/%([\da-fA-F][\da-fA-F])/pack("C", hex($1))/ge;
        # 5) Put in hash table
        $bindings{$key} = $value;
    }
    return(%bindings);
}

# included files need to "return" true, so here it is...
1;
```

CGI Strategy

1. Run with no bindings -> return form
2. User fiddles with form

3. User "submit"
4. Run with bindings from submit
5. Compute and return HTML

dbsearch Notes

- Form trick on no bindings
- << trick for text in Perl code
- security concerns with bindings

dbsearch.pl

```
#!/usr/bin/perl -w

## dbsearch.pl

# Implements a simple search on a local tab-delimited database
# and returns the results in an HTML table.
# When called with no arguments, returns a simple search form.
# A submit of the search form returns a table of all the
# records which contain the target search string.

# Uses the simple CS193i "CGIParse.pl" code to parse the GET/POST data
require "CGIParse.pl";

## Text for: header, trailer, and form
## (uses the "<<" trick to put the text in the file)

$header = <<EOT;
<html><head>
<title>Database Search</title>
</head>
<body bgcolor=white>
<h1>Database Search</h1>
EOT

$trailer = "</body></html>\n";

## Form contains a fileNum binding for which database to search +
## a text field and submit button.
## (Alternatly, we could look in a particular directory for *.txt files
## to dynamically generate the list of available databases.)
$form = <<EOT;
<form method=get>

<select name=fileNum>
<option value=0>movies
<option value=1>books
</select>
```

```

<p>
Search string:<input type=text name=target size=20><br>

<p>
<input type=submit name=submit value="Submit Search">
<input type=reset>
</form>
EOT

# We could put an action in the above form like...
# action = "http://www-cs-faculty.stanford.edu/cgi-bin/nick/dbsearch.pl"
# but if you just omit it, HTTP assumes the URL which produced
# the form which is just what we want.

# List of filenames we are willing to open. This is the simplest way
# to make sure that the client only opens the server files we really intend.
@fileNames = ("movies.txt", "books.txt");

# 1) Begin our HTTP response...
#####
$| = 1;      ## set STDOUT to be unbuffered

print "Content-type: text/html\r\n\r\n";

print $header;

%bindings = ExtractBindings();

## Bindings our form sets for us
# fileNum = 0 based index of the filename to use
# target = string to search for

# %bindings = ("fileNum", 0, "target", "biffo"); # for testing

# 2) If there is no data at all, send back a form...
#####
if (scalar(%bindings) == 0) {
    print $form;
    print $trailer;
    exit(0);
}
else {

# 3) Otherwise we process the database search request...
#####
my($fileNum, $target, $fileName);

$fileNum = $bindings{"fileNum"}; # pull out the attributes we want
$target = $bindings{"target"};
$fileName = $fileNames[$fileNum]; # only allow pre-approved file names (security)

print "<p>Search for '<b>$target</b>' in the database '<b>$fileName</b>'\n";

```

```

# print "$fileNum $target $dataFile\n"; # for testing

# Security note: we do not open an arbitrary filename
# we got "off the net" since that could be misused to
# return files we don't want returned or to run processes
# on the server with names like "|rm *". That file opens
# can run processes is a Perl feature.
# Allternate security idea: allow the filename to come
# from the client, but only allow simple characters
# and not things like ".." or "/" which might allow them to
# move around in the directory structure...
# if ($fileName =~ m"[\w\d\.\-_|\.\.]") # allow only letters, numbers, .-_, but
not ..
#     {ReportError("The filename '$fileName' contains illegal characters");}

open(DATA, "$fileName") || ReportError("Cannot open '$fileName'");

## Make a table out of all the rows which matched...
print "<table border=1 width = 100%>\n";

my($labels);
$labels = <DATA>; # grab the labels from the first line
chop($labels);
TableRow($labels, "th");

my($count) = 0;
while (defined($line = <DATA>)) {
    $target = quotemeta($target);
    if ($line =~ /$target/i) { # Case insensitive search for target
        chop($line);
        TableRow($line, "td");
        $count++;
    }
}

close(DATA);

print"</table>\n";

# print count with the pluralization correct
if ($count == 1) { print "<p>1 matching record\n"; }
else { print "<p>$count matching records\n"; }

print $trailer;
exit(0);
}

# Helper which prints out one row of a table.
# The text should be tab delimited, and the type
# should be "th" for table header, or "td" for
# a table row (the default).
sub TableRow {
    my($text, $type) = @_;

    my(@fields, $elem);

```

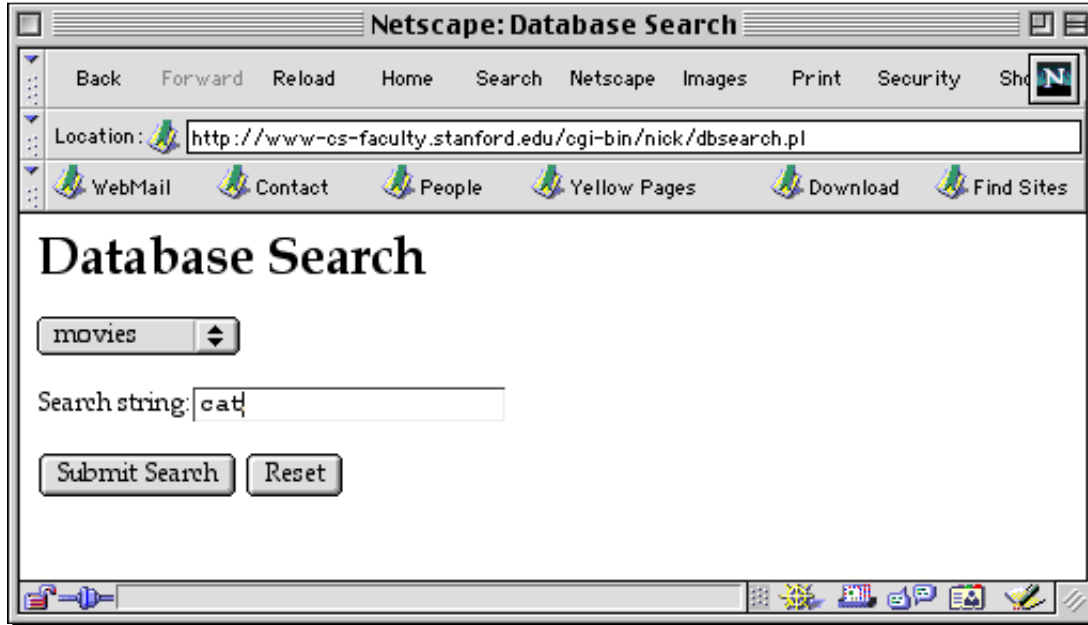
```

@fields = split(/\t/, $text, -1);

print "<tr>\n";
foreach $elem (@fields) {
    print "<$type>$elem</$type>";
}
print "\n</tr>\n";
}

```

Form:



Results:

