The Perl Jam
Exploiting a 20 Year-old Vulnerability

Netanel Rubin
Perl

• The coding style sucks
• The OOP sucks
• The data types suck
  • BAD

• BUT
  • It’s here since late 1987
  • Many legacy systems use it
  • Most sys admins use it
  • Too many security experts use it
“Perl is worse than Python because people wanted it worse.”

Larry Wall -
The creator of the Perl programming language
Perl – Data Types

Scalars
• Just regular scalars
• $scalar = 5; $scalar = ‘hello’;

Arrays
• Just regular arrays
• Use square brackets
• $array[0] = 1

Dictionaries
• Just regular dictionaries
• Called ‘hashes’
• Use curly brackets
• $hash{‘a’} = ‘b’
## Perl Lists

<table>
<thead>
<tr>
<th>@array = (1, 2, 'a', 'b', 'c');</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>print $array[0];</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$sclr = (1, 2, 'a', 'b', 'c');</td>
<td>1</td>
<td>'c'</td>
</tr>
<tr>
<td>print $scalar;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@list = (1, 2, 'a', 'b', 'c');</td>
<td>'c'</td>
<td>5</td>
</tr>
<tr>
<td>print scalar @list;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%hash = (1, 2, 'a', 'b', 'c');</td>
<td>undef</td>
<td>'b'</td>
</tr>
<tr>
<td>print $hash{'a'};</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Not a data type!
- They are just expressions
- Created to confuse us all
**CGI**

- A core module up to Perl 5.20
- Used to access HTTP parameters (GET, POST, COOKIE, etc.)
- Most (if not all) Perl web applications use it (Bugzilla, Twiki, MovableType)
- Has been there for 15 years

```perl
print $cgi->param('foo');
print $cgi->param('bar');
index.cgi?foo=1&bar=a
index.cgi?foo=1&foo=2&bar=a&bar=b
```
• How do you ask for an array?
• You don’t ask for not an array
• A list is the default context in case of a multivalued parameter

FETCHING THE VALUE OR VALUES OF A SINGLE NAMED PARAMETER:

1. @values = $query->param('foo');
   -or-
2. $value = $query->param('foo');

http://perldoc.perl.org/CGI.html
According to OWASP, CGI->param() Returns first occurrence only

Not what happens in real life

OWASP and the documentation mislead programmers
Lists abused

@list = ('f', 'lol', 'wat')
$hash = {'a' => 'b',
    'c' => 'd',
    'e' => @list}
};

print $hash;

- Lists get automatically expanded
- '=>' is just a pretty '(),' 

- Actually known since 2006
  (Dragos Ruiu - http://seclists.org/vulnwatch/2006/q4/6)
- Got no attention whatsoever
- No vulnerabilities published
- NO VULNERABILITIES
**Recap**

- Lists are **dangerous for your health**
  - context, not a **data type**!
- CGI parameters can become lists
- List in hashes **expands the hash**

**super easy to miss!**
Bugzilla

• That Bugzilla.
  (Linux Kernel, Mozilla, Red Hat, MediaWiki, KDE, Gnome, Eclipse, Open Office, other shit)

• Some **privileges are given via email regex**
  • Example: *@mozilla.org can view confidential firefox bugs*
• New user email gets validated (prior to completing registration) using an emailed token to prove email ownership
• Post-validation the user is asked for a password and a real name
• Then, this code happens:

```perl
my $otheruser = Bugzilla::User->create({
    login_name => $login_name,
    realname => $cgi->param('realname'),
    cryptpassword => $password
});
```

$login_name => Email address validated (extracted from the DB)
$password => The user defined password (as a scalar)
$cgi->param('realname') => Bingo!
CVE-2014-1572 – User Verification Bypass

- Super simple vulnerability
- Been there for over 7 years
Recap 2

• Lists are **messy and broken**
• Hashes behavior was already public
• Hashes can’t be the only place

**What else can we mess with?**
Lists severely abused

```perl
sub test {
    ($a, $b, $c) = @_;
    print ($a, $b, $c);
}

test('a', 'b', 'c'); # Regular call

@list = ('b');
test('a', @list); # List call

@list = ('b', 'c');
test('a', @list); # List call

@list = ('b', 'c');
test('a', @list, 'd'); # List call??
```

<table>
<thead>
<tr>
<th>Expected</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>['a'];</td>
<td>['a'];</td>
</tr>
<tr>
<td>['b'];</td>
<td>['b'];</td>
</tr>
<tr>
<td>['c'];</td>
<td>['c'];</td>
</tr>
<tr>
<td>['a'];</td>
<td>['a'];</td>
</tr>
<tr>
<td>['b'];</td>
<td>['b'];</td>
</tr>
<tr>
<td>['c'];</td>
<td>['c'];</td>
</tr>
<tr>
<td>['a'];</td>
<td>['a'];</td>
</tr>
<tr>
<td>['b', 'c']];</td>
<td>['b', 'c']];</td>
</tr>
<tr>
<td>['a'];</td>
<td>['a'];</td>
</tr>
<tr>
<td>['b', 'c']];</td>
<td>['b', 'c']];</td>
</tr>
<tr>
<td>['a'];</td>
<td>['a'];</td>
</tr>
<tr>
<td>['b', 'c']];</td>
<td>['b', 'c']];</td>
</tr>
</tbody>
</table>
DBI;

- Core module
- **The typical database handler**
  - Almost everyone uses it
- Built-in SQL filtering/escaping function
  - `DBI->quote()`

```perl
print 'select * from users where username=' . $dbh->quote($cgi->param('user')));
```

```
index.cgi?user=user  ───> select * from users where username = 'user'
```

```
index.cgi?user=user'  ───> select * from users where username = 'user\'
```

Live Demo
sub quote ($$;$) {
  my ( $self, $str, $type ) = @_;

  ... defined $type && ($type== DBI::SQL_NUMERIC() || $type== DBI::SQL_TINYINT())
  and return $str;

  ...
}

Exploiting all the Perl

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

Just a small portion of what could really be achieved
Summary

• Lists are hazardous, bizarre expressions
• Perl is a hazardous, bizarre language

• Now’s the time to stop using Perl!
  • Stop the write-only code
  • Stop the miss-functional OOP
  • Stop the security breaches all over the place

• At least know your language “features”
Thanks!