آسیب پذیری های Format String در برنامه های پرل

مقدمه

آسیب پذیری های Format String در برنامه های C در سال های اخیر بررسی شده اند. در این نوع حملات تمرکز ما بیشتر روی اجرای کدهای دلخواه می باشد، اگرچه کارهای دیگری نیز می توان انجام داد. برای جوامع امنیتی در دنیای مجازی ثابت شده است که قطعه کدهای بیشماری در Perl از همان آسیب پذیری هایی از نوع Format String رنج می برند که در برنامه های C وجود دارد.

احتمال وجود آسیب پذیری های Format String برای بسیاری از زبان های برنامه نویسی دیگر نیز وجود دارد. چون توجهی به دیگر زبان های برنامه نویسی نمی شود، لذا با وجود اینکه برنامه ها (application ها) برای دیگر انواع آسیب پذیری بررسی می شوند (به این کار Auditing می گوییم)، احتمال زیادی وجود داردکه بسیاری از آنها این مشکلات را داشته باشند.

در هر زبانی که از رشته های قالب (format string ها) پشتیبانی شود، application هایی که در آن زبان نوشته می شوند احتمال وجود آسیب پذیری های Format String می باشد. حکمی که در این نوع آسیب پذیری ها به کار می رود به رفتارهای پشتیبانی شده توسط رشته های قالب و فعل و انفعال آنها با عوامل درونی زبان برنامه نویسی مربوط است. برای نمونه، Jack Louis یک مشکل format string را در یک برنامه وبمیل که به زبان پرل نوشته شده گزارش داده است (CVE-2005-3912). همچنین نشان داد که وجود یک مشکل در مفسر پرل[1] چگونه به دستکاری حافظه و اجرای کد در برنامه های آسیب پذیر می انجامد (CVE-2005-3962).

این مقاله تنها در سطح برنامه (application) آسیب پذیری های Format String را بررسی می کند. بایستی متذکر شد که حتی در صورتی که مشکل مفسر تصحیح گردد، تنها تاثیر آسیب پذیری های Format String کاهش می یابد و بطور کامل برطرف نمی شوند.

حتی اگر مشکلات در خود مفسر حذف شوند، آسیب پذیری های Format String در برنامه های پرل منجر به رخنه های اطلاعاتی تکذیب سرویس (زوال حافظه یا دیسک) و دستکاری متغیرهای برنامه می شوند، بطوریکه تاثیرات امنیتی را روی برنامه اعمال کنند.

برای مثال، اگر نفوذگر بتواند محتویات رشته قالب را کنترل کند، توابع sprintf() و printf() در پرل را می توان در راه نادرست (سو استفاده) بکار برد. چون توابع مشابهی از C در برنامه های پرل استفاده می شوند، لذا برنامه نویسانی که به تازگی با زبان پرل آشنا شده اند، از توابعی که مکررا در C استفاده کرده اند استفاده می کنند.

باید گفت که عیب یاب (یا بررسی کننده عیب)[2] پرل گونه های مختلف حملات Format String را بررسی نمی کند. این رفتار در Perl 5 و Perl 5.6.1 تفاوت دارد (در نسخه 5.6.1 ورودی های مخرب بیشتری بررسی می شوند). با این حال، اعمال عیب یاب جهت رفع این نوع آسیب پذیری ها کافی نیست.

۱) جزئیات حمله

در زیر بعضی از تعیین کننده های[3] خطرناک تر را بهمراه دلیل بررسی می کنیم.

---

[1] Perl Interpreter
[2] taint checker

۱) زوال حافظه یا دیسک

تعیین کننده "%s" و دیگر تعیین کننده هایی که اجازه تعریف طول فیلد می دهند را می توان برای مصرف حجم زیادی از CPU، حافظه و/یا دیسک استفاده کرد، مثلا "%99999s" (برای مصرف یک گیگابایت از حافظه یا دیسک استفاده از "%999999999s" کافی است، اما گزارش شده که این تعیین کننده سبب کرش کردن برنامه های پرل می شود).

۲) دستکاری یا تغییر متغیرها

با استفاده از تعیین کننده "%n"، نفوذگر می توان مقادیر متغیرهای خاصی را که به عنوان آرگومان برای printf/sprintf ارائه شده اند تغییر دهد و به این طریق شاید بتوان برنامه را طوری تغییر داد که تاثیرات امنیتی داشته باشد. متغیر به یک عدد، اساسا یک 0 تغییر داده می شود.

دلیل این مشکل مبتنی بر چگونگی استفاده برنامه از متغیرها می باشد. شبه-کد زیر را برای یک روتین اعتبارسنجی در نظر بگیرید:

```
$input = GetInputFromUser();
if (UserHasAuthenticated($user))
{
$a = 0;
}
else
{
$a = 1;
}
$str = sprintf($input, $a);
if ($a)
{
PromptUserToAuthenticate($user);
}
else
{
DoThingThatOnlyUsersShouldDo($user);
}
```

اگر $input بصورت "%10s" باشد، آنگاه str با عملیات padding از ۱۰ فضای خالی (space) قالب بندی شده و $a تغییر داده نمی شود؛ اما اگر $input بصورت "%n" باشد، آنگاه $a به 0 تغییر داده می شود و نفوذگر بطور موثری می تواند بررسی اعتبارسنجی را دور بزند.

۳) شیفت کردن یا تغییرمکان آرگومان[۴]

تعیین کننده "%p" یک اشاره را برای پردازش آرگومان بعدی در یک فراخوانی به printf* قالب بندی می کند. این کار سبب می شود که تمام آرگومان های باقیمانده از تعیین کننده های قالبی که برنامه نویس مد نظر داشته، تغییر مکان دهند یا اشتباه راهنمایی شوند.

در دنیای پرل به عباراتی که همراه علامت % باشند، اصطلاحا Specifier یا تعیین کننده می گوییم و در زبان C این عبارات، Format Character ها یا کاراکترهای قالب نامیده می شوند.

[4] Argument Shifting

شیفت کردن آرگومان را می توان برای دورزدن عملگرهای تنظیفی برای دیگر آسیب پذیری ها نیز بکار برد که این کار با با شیفت کردن مقادیر غیرتنظیف (منظم) به متغیرهایی که مقادیر منظم را دارا می باشند انجام می شود.

۴) تغییر خروجی های مدنظر

هر تعیین کننده قالب می تواند قالب مد نظر برای خروجی های ساخت یافته را تغییر دهد. با تغییر خروجی هایی که برنامه نویس مد نظر داشته است، می توان باعث خرابی فایل ها شد و از نگاه دیگر می توان آسیب پذیری هایی را در برنامه هایی که از این خروجی ها استفاده می کنند اکسپلویت کرد. برای مثال، تعیین کننده "%p" (که مقدار اشاره گر را چاپ می کند) را می توان برای تولید مقادیر صحیحی بکار برد که از محدوده اعداد مورد انتظار برای ورودی فراتر باشند. مثالی را در زیر می بینید:

```
$index = GetUserInput();
if (($index > 32) || ($index < 0))
{
print STDERR "Error: Index must be between 0 and 32.";
}
($sec,$min,$hour,$mday,$mon,$year) = gmtime(time);
printf DATABASE, "$index %4d/%02d/%02d %02d:%02d:%02d\n",
$year+1900, $mon+1, $mday;
```

اگر $index برابر با "1" باشد، آنگاه نتیجه شاید به صورت زیر باشد:

1 2002/10/01 06:58:42

اما اگر $index برابر با "%p" باشد، شرایط خطا تشخیص داده نمی شود (چون رشته برابر با ۰ ارزیابی می شود) و شاید نتیجه بصورت زیر شود:

130690 10/01/06 58:42:00

در اینجا نه تنها مقدار 'index' از ماکزیمم ۳۲ فراتر رفته است، بلکه تمام مقادیر نیز غلط هستند! به این دلیل که تعیین کننده %p برای قالب بندی یک اشاره گر به عبارت $year+1900 مورد استفاده قرار گرفته است. در این صورت تمام آرگومان های دیگر بغلط راهنمایی شده و به تعیین کننده قالب اشتباه (غیرمجاز) اعمال شده اند. لذا مقدار ماه بصورت سال قالب بندی شده و مقدار ثانیه به صورت دقیقه قالب بندی می شود و ... .

۵) دور زدن عملگرهای منظم (تنظیفی)

عملگرهای منظم (که فاصله ها (space) را حذف می کنند) را می توان با استفاده از "%2s" یا دیگر تعیین کننده های قالب که فاصله ها را تولید می کنند فریب داد. شاید برنامه ها ذر هنگام انتقال آرگومان ها به دستورها یا قالب بندی داده ها، سعی در حذف فاصله ها کنند. در زیر مثالی را می بینید:

```
opendir(DIR, ".");
while ($file = readdir(DIR))
{
if ($file =~ /\s/)
{
print STDERR "Warning: '$file' has spaces, replacing with _\n";
$file =~ s/\s/_/g;
}
if ($file =~ /^-(fiprR)+$/)
{
print STDERR "Warning: '$file' matches switches for /bin/cp!\n";
```

```
# skip this one.
next;
}
$backup = sprintf("$file.bak"); # C programmers might do this
system("/bin/cp $file $backup"); # but this *is* just an example
}
closedir(DIR);
```

اگر $file برابر با "-R%2ssubdir" تنظیم شود، آنگاه بررسی برای "سوئیچ های خطرناک" ناکام مانده و فرخوانی سیستمی زیر نتیجه می گردد:

system("/bin/cp -R subdir subdir.bak");

۶) دیگر سناریوهای حمله ای

چندین سناریوی حمله ای ممکن برنامه های پرلی را درگیر کار می کنند که پیام ها یا گزارش های log را تولید می کنند:

– نام فایل ها که حاوی تعیین کننده های قالب هستند می تواند وضعیت پردازش فایل ها را تغییر دهد.

– آدرس های IP که بعنوان نتیجه gethostbyname() دریافت می شوند را می توان در عملیات DNS reverse lookup طوری بکار برد که حاوی رشته های قالب باشند. فایل های log را می توان بسادگی با استفاده از رشته های سبک "%999s" پر کرد.

۲) بحث راجع به رشته های قالب و عیب یاب

*** نسخه 5.004:**

عیب یاب ظاهرا نام فایل را مبنی بر عیب داشتن بررسی نمی کند (مثلا هنگامی که نام فایل توسط تابع readdir() دریافت می شود). احتمالا، دیگر انواع ورودی های غیرمستقیم (یعنی ورودی هایی که به وسیله ورودی استاندارد یا صفحه کلید دریافت نمی شوند) دارای این عیب نیستند. به هر حال، این عیب یاب منابع مستقیم از ورودی مثل stdin و متغیرهای محیطی را تشخیص میدهد.

*** نسخه 5.6.1:**

در صورتی که نام فایل ها دارای عیب باشد، عیب یاب برنامه را می بندد. اگرچه برنامه از جنبه استفاده از فراخوانی های خطرناک ایمن می باشد، اما هنوز احتمال رخداد تکذیب سرویس وجود دارد که می تواند در یک برنامه پردازش log رخ دهد.

به خاطر داشته باشید که تا زمانی که یک متغیر دارای عیب printf* به یک فراخوانی خطرناک مثل system() منتقل نشود، عیب یاب متوقف نمی شود. پس، اگر برنامه با تعیین کننده هایی مثل "%n" (که یک آرگومان را برای printf* تغییر می دهد) چک نشده باشد، آنگاه احتمال دارد که بررسی عیب کشف نگردد).

حملاتی مثل مصرف منبع و تغییر داده های قالب هنوز کارکرد دارند. با این حال، اگر طوری عیب یاب را تغییر دهیم که به محض رویارویی با printf/sprintf خارج شود، در این صورت باعث از بین رفتن و عیب یابی برنامه های حاضر خواهد شد.

"آزمایش" sprintf/printf با نام های فایل معمول عیب یاب را مورد ارزیابی دقیق قرار نمی دهد، مگر اینکه %n در نام فایل قرار بگیرد. لذا، اگر برنامه نویس کد پرل را آزمایش کند، اما گزینه '%n' را در بررسی های

خود وارد نسازد، عیب یاب بدرستی وظیفه خود را انجام نخواهد داد. به هر حال، یک ورودی دیگر با '%n' سبب می شود که برنامه ناگهان متوقف شود (halt کند) که این امر ناشی از خطای عیب یاب است.

توجه: عیب یاب هنگام فراخوانی system() با آرگومان هایی به صورت زیر، قادر به بازگو کردن مشکلات نیست:

```
system("/bin/echo", $tainted_var1, $tainted_var2);
```

مثال زیر با استفاده از ورودی از stdin (ورودی استاندارد)، خطایی را از عیب یاب تولید می کند:

```
$a = <STDIN>;
chomp($a);
$str = sprintf("$a.txt");
system("/bin/echo $str");
```

مثال زیر نیز با استفاده از یک عملیات Directory Listing، خطایی را از عیب یاب تولید می کند:

```
opendir(DIR, ".");
while ($file = readdir(DIR))
{
system("/bin/echo $file");
}
closedir(DIR);
```

Statement From Perl Language Developer:

این مسائل خودشان حفره های امنیتی را در پرل نشان نمی دهند. نسخه های بعدی پرل ممکن است بررسی های معایب را برای Format String ها یا تنها جنبه های مشخصی از قالب ها (مثل %n) توسعه دهد.

۳) برنامه های آسیب پذیر در دنیای واقعی!

برای سادگی برنامه هایی را از سال ۲۰۰۲ مثال می زنیم. چرا که این برنامه ها از اولین برنامه هایی بوده اند که علیه حملات Format String آسیب پذیر می باشند:

۱. ftplogchech
۲. perl-nocem
۳. وب سرور WASD OpenVMS

* برنامه ftplogcheck

ftplogchech برنامه ای است که برای پردازش log های wu-ftpd مورد استفاده قرار گرفته و نتایج آماری آنرا تولید می کند. این برنامه، به عنوان بخشی از توزیع wu-ftp توزیع نمی گردد. بخشی از برنامه ftplogcheck لیست هایی را گزارش می دهد که توسط کاربر "anonymous" فایل هایی در سرور آپلود شده اند. این کد به صورت زیر می باشد:

```
printf REPORT "$time $host $filesize $filename $name\n";
```

اگر سرور wu-ftp طوری پیکربندی شده باشد که اجازه آپلود از کاربران anonymous را بدهد، آنگاه نفوذگران می توانند فایل هایی را آپلود کنند که نام آنها حاوی رشته های قالبی خطرناکی باشد. مسلم است که این نام به متغیر $filename داده می شود.

در این مورد، نفوذگر می تواند با تولید یک گزارش بسیار طولانی سبب مصرف حافظه یا دیسک شود (اگر $filename برابر با "%999999s" باشد) یا اینکه نام فایلی که آپلود می شود را غلط جلوه دهد (اگر $filename برابر با "word1%1sword2" باشد که سبب تولید رشته "word1 word2" می شود). می توانید این برنامه را در آدرس زیر بیابید:

http://www.landfield.com/software/ftp.landfield.com/wu-ftpd/tools/ftplogcheck

* برنامه perl-nocem

perl-nocem اسکریپتی است که ظاهرا در INN 2.0 Beta وجود دارد، اما این اسکریپت به صورت مستقیم با INN 2.3.3 و هر نسخه به صورت 2.x ارائه نشده است.

http://www.isc.org/ml-archives/inn-workers/2001/05/msg00177.html

این اسکریپت اخطارهای NoCeM را پردازش می کند. سرور می تواند این اسکریپت را برای پردازش اخطارهای لغو در بند های نشانه گذاری شده با PGP بکار برد. در do_nocem()، پس از اضافه شدن مقادیر $nid و $issuer به رشته قالب، یک فراخوانی به sprintf() انجام می شود:

```
logmsg(sprintf("processed notice $nid by $issuer"
. " ($nr ids, %.5f s, %.1f/s)", $diff, $nr / $diff));
```

مقدار $nid از هدرهای جدید از بند "notice-id" گرفته می شود. این مقدار بدرستی جهت عیب بررسی نشده است. لذا، رشته های قالب مضر را می توان به این فرخوانی sprintf() اضافه کرد. متغیر $issuer از هدر "issuer" گرفته می شود، اما این مقدار بایستی توسط فایل کنترلی perl-nocem مجاز شناخته شود. امکان استفاده از یک کاراکتر wildcard و تطابق دادن هر issuer وجود دارد.

متغیر $nr حاوی کل تعداد بندهایی است که باید لغو گردند و متغییر $diff سعی در اندازه گیری زمانی می کند که برای لغو بندها لازم است که این مقدار بر اساس یک باگ آشکار برابر با 0.01 می باشد.

اما با اندکی فکر در می یابیم که هدف این حمله محدود است چرا که "پیام تنها زمانی چاپ خواهد شد که بند nocem برای PGP بررسی شده باشد، لذا نفوذگر باید یکی از لغو کننده های مورد اعتماد[5] باشد".

ورودی نمونه:

فرض کنید که ۱۰ بند (یا مقاله) باید لغو گردند ($nr = 10) و $diff برابر با 0.01 می باشد. با یک $nid (هدرNotic-ID) از "NID" و یک $issuer (هدرIssuer) برابر با "ISSUER@example.com"، خروجی پیام log به صورت زیر می باشد:

processed notice NID by ISSUER@example.com (10 ids, 0.01000 s, 1000.0/s)

---

[5] Trusted Cancellers

مصرف یا زوال حافظه/دیسک

با یک Notice-ID برابر با "%9999999s"، مقدار زیادی برای ناحیه حافظه ای مورد استفاده و/یا فضای فایل log در نظر گرفته می شود:

```
processed notice
[etc.]
```

تغییر متغییر $diff

با یک notice-id برابر با "%n"، perl-nocem متغیر $diff را به محض تضاد با مقدار اصلی خود (معمولا 0.01) به ۱۷ تغییر می دهد (طول زیر رشته "Processed Notice"). این مسئله، سبب می شود که در پیام خطا، مدت زمانی که برای لغو بندها صرف شده است به اغراق زیاد نمایش یابد:

```
processed notice by ISSUER@example.com (10 ids, 1000.00000 s, 0.0/s)
```

(به فضای دو برابر در "notice by" که notice-id بایستی به جای آن باشد توجه کنید).

توجه کنید که اگر perl-nocem از رشته قالبی استفاده کرده باشد که با متغیر "$nid" شروع شود (مثلا "$nid Notive Processed" بجای "Processed Notive $nid")، آنگاه متغیر $diff بایستی به مقدار ۰ تنظیم شود و عبات "$nr / $diff" بعلت وجود یک خطای تقسیم-بر-صفر، احتمالا سبب خروج از برنامه می شود (توجه کنید که تقسیم عدد بر صفر را بی نهایت یا تعریف نشده می نامیم).

دیگر روش ها برای تغییرخروجی

با یک notice-id برابر با "%p"، پیام log به صورت زیر نتیجه می شود:

```
processed notice 130498 by [ISSUER] (10 ids, 0.50000 s, 0.0/s)
```

که "130498" یک Notice-ID نادرست می باشد.

* برنامه WASD

آقای Gailly وجود یک مشکل Format String را در وب سرور WASD OpenVMS تذکر داده است. یک برنامه نمونه با نام PerlRTE_example1.pl حاوی کد آسیب پذیر زیر بود:

```
printf ("$name=\"$ENV{$name}\"\n");
```

که نفوذگر می تواند متغیر $name را طوری دستکاری کند که حاوی رشته های قالب باشد (که این کار از طریق یک رشته گزارشی انجام می شود).

۴) اجتناب از آسیب پذیری های Format String در خلال توسعه برنامه ها

هنگامی که برنامه های پرل را می نویسید رهنمودهایی زیر را دنبال کنید:

- از رشته های ثابت برای قالب دهی (formatting) استفاده کنید.
- متغیرهای پرل را مستقیما به رشته های قالب ندهید (برای مثال "$bad %10s" یا "%10s " . $bad).
- تا جای امکان از توابع printf و sprintf استفاده نکنید.

- اگر استفاده از این توابع ضروری است، قرار دادن نشانه نقل قول در تعیین کننده "%" را رسیدگی کنید. توجه کنید که این کار را باید قبل از قرار دادن داده های کاربری درون یک رشته قالب انجام دهید.
- حتما قابلیت Taint Checking را فعال کرده و سپس برنامه های خود را اجرا کنید. این قابلیت بسیاری از مشکلاتی را که در این مقاله گفته شد شناسایی می کند.

تذکراتی در یافتن آسیب پذیری در کد منبع:

تشخیص کدهای مضر در پرل نسبت به کدهای C اندکی سخت تر می باشد. رشته های ثابت می توانند حاوی شناسه های پرل باشند، از قبیل متغیرها یا ارجاع هایی که قبل از انتقال رشته ها به printf/sprintf به رشته اضافه گردیده اند.

```
$fmt = <USER_INPUT>;
printf("THIS IS A POTENTIALLY VULNERABLE $fmt FORMAT STRING\n");
```

۵) پیشنهاد برای تحقیق

این مقاله کار جامعی نیست، لذا تحقیقات بیشتری مورد نیاز است. توسعه دهندگان نرم افزار و پژوهندگان آسیب پذیری بایستی به صورت فعال در صدد یافتن مشکلات **Format String** در تمام زبان های برنامه نویسی باشند (نه فقط **C** و پرل). لذا در این راستا موضوعاتی را ارائه می کنیم:

- برای هر زبان برنامه نویسی، تمام توابع کتابخانه ای معمول و درون ساخت که از رشته های قالب استفاده می کنند را پیدا کنید.
- ابزارهای آنالیز کدهای باینری یا کدهای منبع را برای استفاده نامناسب از این توابع توسعه دهید.
- برنامه های شخصی (و کوچک) که قبلا خالی از آسیب پذیری تصور می شده اند را با تمرکز روی مشکلات رشته های قالب بررسی کنید.
- مفاهیم و استدلال های موجود در فعل و انفعال بین یک زبان برنامه نویسی سطح بالا و زبانی که در آن نوشته می شود را بررسی کنید. برای مثال، ممکن است مشکلات رشته قالب در بایت پوچ در برنامه های پرل و **PHP** در فعل و انفعال آنها با کدهای **C** وجود داشته باشد (که این مورد به تازگی کشف شده است).
- بررسی های بیشتری روی **Taint Checker** انجام دهید.

۶) برنامه های نمایشی

این برنامه ها چندین مشکل را که در بالا توصیف شد نشان می دهند (متذکر می شویم که برنامه ها را برای سهولت کار با comment های انگلیسی رها کرده ایم).

۱. دستکاری آرگومان

مثال ۱:

```
#!/bin/env perl

# when run with taint checking (-T), this seems to properly barf about
# dependency errors (try a "clean" format string like "5s%s%s" vs. a
# dirty one with a "%n" in it).
```

```
$ENV{"PATH"} = "";

# try as input: "%s%n%s" --> modifies $b

$a = "A";
$b = "B";
$c = "C";
$x = sprintf($ARGV[0], $a, $b, $c);

print "\$a='$a'; \$b='$b'; \$c='$c'\n";

print "$x\n";

system("/bin/echo $a $b $c");
```

مثال ۲:

```
# Create a directory that contains files with these names:
# X%10sX
# %p
# %s
# abc%ndef

# This was gleaned from some real-world code, but the print was
# changed to printf.

# Change what filenames are processed via format strings in
# the filenames, such as a file named "%p%n"
#
# You can "erase" a filename by using '%s', and having this "blank"
# filename could throw off the argument count to system or exec calls,
# which could alter behavior. Consider a backup command like
# exec("/bin/cp", file1, file2) where file1 can be "blanked" out
#
# Similarly, you could "erase" portions of a filename with "%n" or
# "%s". The filename ABC.TXT would be equivalent to ABC%n.%nTXT
#
# You can create very long filenames by using '%999s' (for example).

opendir(DIR, ".");
while ($file = readdir(DIR))
{
print "Real filename: $file\n";
printf "Filename in format string: $file\n\n";
}
closedir(DIR);
```

۲. استفاده نادرست از رشته قالب در پردازش log برای بسیاری از برنامه ها که در پرل نوشته شده اند.

این مورد باعث نوشته شده رشته های بسیار بزرگ در فایل ها یا ارسال به پروسه ها می شود (بزرگی این رشته ها مورد انتظار برنامه نویس نبوده است). کدهایی که نیازمند ورودی های سالم (با قالب صحیح) هستند (منظور کدهایی است که برنامه نویس در آن مشکلات را بررسی نکرده و استفاده از کد را به بهترین نحو پیش خود تصور کرده و در نتیجه استفاده صحیح از برنامه را به عهده کاربر گذاشته است) ممکن است نسبت به سرریزبافر یا دیگر مشکلات آسیب پذیر باشند. در برنامه های بسیاری به شخصه استفاده از کدهایی مانند زیر را دیده ام:

```
printf "A=$a\n"
```

منابع و ارجاع ها

[1] Jean-loup Gailly
"remote SYSTEM compromise in WASD OpenVMS http server" Bugtraq post September 26, 2002
http://marc.theaimsgroup.com/?l=bugtraq&m=103307640806862&w=2

[2] Arjan de Vet "Security aware programming with Perl" http://www.madison-gurkha.com/publications/yapc2001/text0.htm

[3] Steve Christey "An Alternate View of Recently Reported PHP Vulnerabilities" Bugtraq
http://seclists.org/lists/bugtraq/2003/Apr/0085.html

[4] http://www.cansecwest.com/archives.html

[5] Jack Louis "Webmin miniserv.pl format string vulnerability" November 29, 2005
http://lists.immunitysec.com/pipermail/dailydave/2005-November/002685.html

[6] Jack Louis "Perl format string integer wrap vulnerability" December 1, 2005
http://marc.theaimsgroup.com/?l=full-disclosure&m=113345191421286&w=2

[8] WASD PerlRTE_example1.pl http://wasd.vsm.com.au/ht_root/src/perl/readmore.html

[9] perl-nocem: http://www.isc.org/ml-archives/inn-workers/2001/05/msg00177.html

[10] INN-workers security report
http://marc.theaimsgroup.com/?l=inn-workers&m=103643921519928&w=2
http://marc.theaimsgroup.com/?l=inn-workers&m=103644050021431&w=2

ترجمه و تالیف: سعید بیکی (cephexin@secumania.net)

Secumania Security & Vulnerability Research Lab
www.secumania.net