

virus

BULLETIN

Fighting malware and spam

CONTENTS

2	COMMENT How much malware is really out there?
3	NEWS VB Seminar: 24 May 2011 Facebook spammers fined
3	VIRUS PREVALENCE TABLE
4	MALWARE ANALYSIS \$\$\$_+\$\$+\$\$\$_+_+\$+\$\$_+\$+\$\$_+\$\$_\$_
8	TECHNICAL FEATURE DLL hijacking
14	TUTORIAL Introduction to hostile Java analysis
18	CALL FOR PAPERS VB2011 Barcelona
19	FEATURE The top 10 spam, malware and cybersecurity stories of 2010
24	COMPARATIVE REVIEW VB100 comparative review on Ubuntu Linux 10.04 LTS
36	END NOTES & NEWS

IN THIS ISSUE

LOOK, NO ALPHANUMERIC!

JJEncode is a JavaScript encoding method that produces files that contain no alphanumeric characters. A demonstration version is freely available from the author's website, and has already been used in malware. Peter Ferrie describes how it works.

page 4

DLL VULNERABILITIES

There are a few good reasons for taking another look at DLL hijacking – including the fact that we don't learn from our mistakes. Aleksander Czarnowski takes an in-depth look at the DLL hijacking story.

page 8

VB100 CERTIFICATION ON UBUNTU LINUX

This month's VB100 test on Ubuntu Linux saw a considerably more modest field of entrants than the Windows-based tests of late, and a strong batch of performances. John Hawes has the details.

page 24





'For most people even a single piece of malware is too much – especially if they are currently affected by it.'

Robert Sandilands Commtouch

HOW MUCH MALWARE IS REALLY OUT THERE?

Since the beginning of the AV industry more than two decades ago, the amount of malware in existence has been an often-debated point. Answers range from none to an infinite amount.

If you use a custom operating system on custom hardware that is running applications that are of no importance to anybody but yourself, then you are probably right to assert that there is no malware. There are probably no financial (or other) incentives to attack such a system.

To get closer to a real answer we need to look a bit further than this contrived example – although there may be some truth in the observation that there is only as much malware as people want to know about. Whether that leaves you with no malware or with an infinite amount depends on the perspective. For most people even a single piece of malware is too much – especially if they are currently affected by it.

Let us assume that any platform that is somewhat accessible and has either a large enough user base or great enough value will eventually be attacked by malware. We can see this with the recent growth of *Mac* malware and also with something like Stuxnet that (probably) attacked a single, but very high-value target.

In the mid 90s we were in a position where we could accurately count the number of viruses that had been seen. This was possible for several reasons:

1. The number of new viruses was small enough for each sample to be identified and analysed in detail.
2. It was easy to determine which part was virus and which part was the infected application.
3. The size and complexity of the malware was quite limited.

If you took one of the polymorphic file infectors from the 1990s and infected 100 million clean files then you could get 100 million unique infected files. If you then counted that like most people count malware today you could say that you had 100 million pieces of malware. This would be incorrect, but it is how malware tends to be counted these days.

There are several reasons for this. The first is that modern malware is probably several orders of magnitude larger and more complex than the malware that was around in the mid 90s. The second major reason is the use of packers to obfuscate the malware. The last, and probably most important reason is the location of the polymorphic engine. This has moved from being inside the 1990s virus to being on the server today, where analysts generally cannot access it.

In the old days we could carefully replicate most pieces of malware in a protected and isolated environment and gain a good understanding of how each morphs and we could therefore use a small number of very efficient signatures to detect those pieces of malware.

These days most pieces of malware won't work without what appears to be a real Internet connection. They generally also won't replicate. To get a 'replicated' copy you either have to be reinfected or download a new copy of the malware.

Not only that, but analysing a specific piece of malware in detail can take weeks to months. For example, people are still busy analysing Stuxnet more than six months after the initial samples were found and we don't yet have a complete picture of the malware. Given the flood of malicious files we receive, we are rarely, if ever, able to spend such amounts of time on any specific malware or malware family.

We have a catch-22 situation. If we don't take the time to analyse the malware and understand that we are actually working with a limited set of malware families then we are dealing with a virtually infinite amount. If we take the time to understand each malware family, then proper detection for the family will take significantly longer than our customers will accept. In the end it is all about doing it fast or doing it well. You can rarely do both.

Editor: Helen Martin

Technical Editor: Morton Swimmer

Test Team Director: John Hawes

Anti-Spam Test Director: Martijn Grooten

Security Test Engineer: Simon Bates

Sales Executive: Allison Sketchley

Web Developer: Paul Hettler

Consulting Editors:

Nick FitzGerald, *Independent consultant, NZ*

Ian Whalley, *IBM Research, USA*

Richard Ford, *Florida Institute of Technology, USA*

NEWS

VB SEMINAR: 24 MAY 2011

Following the success of last year's 'Securing Your Organization in the Age of Cybercrime' seminar in London,



VB is pleased to announce that the 2nd VB seminar will be held in association with the MCT Faculty of The Open University 24 May 2011 in Milton Keynes, UK.



Aimed at a corporate audience, the seminar will give IT professionals an opportunity to learn from and interact with top security experts and take away invaluable advice and information on the latest threats, strategies and solutions for protecting their organizations.

More details and registration can be found online at <http://www.virusbtn.com/seminar/2011/>.

FACEBOOK SPAMMERS FINED

Social networking giant *Facebook* celebrated victory against spammer Philip Porembski last month, when a federal court in California ordered him to pay \$360.5 million in punitive damages to the company and ruled that he be barred from the social networking site indefinitely.

Porembski was accused of hijacking more than 160,000 *Facebook* user accounts and sending over 7.2 million spam messages to users of the service. The spam included links to sites from which Porembski earned affiliate commissions as well as links to fake *Facebook* login pages, from which Porembski was able to harvest users' login credentials and hijack accounts.

Facebook reportedly received more than 8,000 complaints about the spam, while it claims that in the region of 4,500 people went as far as closing their accounts completely as a result of the spamming.

Meanwhile, a court in Quebec has enforced a US\$873 million judgement against another *Facebook* spammer, Adam Guerbuez. As long ago as November 2008, a US district court ordered Adam Guerbuez and his company Atlantis Blue Capital to pay *Facebook* \$873 million in damages for having sent more than four million spam messages via the social network. However, Guerbuez – a resident of Quebec – appealed against the ruling, arguing that it should not be recognized in Quebec since such a hefty fine was not compatible with Canadian anti-spam laws. The Quebec Superior Court did not agree however, finding no legitimate reason not to enforce the California judgement.

Prevalence Table – December 2010^[1]

Malware	Type	%
Autorun	Worm	9.86%
Conficker/Downadup	Worm	7.50%
VB	Worm	6.89%
FakeAlert/Renos	Rogue AV	5.90%
Sality	Virus	4.45%
Downloader-misc	Trojan	3.99%
Heuristic/generic	Virus/worm	3.97%
Heuristic/generic	Trojan	3.76%
Zbot	Trojan	3.39%
OnlineGames	Trojan	3.10%
Virut	Virus	2.94%
Adware-misc	Adware	2.67%
Agent	Trojan	2.44%
Exploit-misc	Exploit	2.43%
Crypt	Trojan	2.28%
FakeAV-Misc	Rogue AV	2.02%
Dropper-misc	Trojan	1.88%
Alureon	Trojan	1.87%
Autolt	Trojan	1.85%
Mdrop	Trojan	1.64%
Bifrose/Pakes	Trojan	1.61%
Iframe	Exploit	1.24%
Tanatos	Worm	1.20%
PDF	Exploit	1.09%
Slugin	Virus	1.09%
Themida	Packer	0.89%
Crack/Keygen	PU	0.85%
WinWebSec	Rogue AV	0.82%
Encrypted/Obfuscated	Misc	0.80%
LNK	Exploit	0.78%
Injector	Trojan	0.77%
Backdoor-misc	Trojan	0.73%
Others ^[2]		13.29%
Total		100.00%

^[1]Figures compiled from desktop-level detections.

^[2]Readers are reminded that a complete listing is posted at <http://www.virusbtn.com/Prevalence/>.

MALWARE ANALYSIS

\$\$\$_+\$\$+\$\$_+_+\$ \$\$_+\$\$\$\$_+\$\$_ \$

Peter Ferrie
Microsoft, USA

Imagine a JavaScript encoding method that produces files that contain no alphanumeric characters, only symbols such as '\$', '_', and '+'. It would be difficult to imagine how it could possibly work, but unfortunately one such encoder exists. It is called 'JJEncode'. A demonstration version is freely available from the author's website, and has already been used in malware. This article provides a detailed description of how it works.

\$_+"\\"+_+\$_ \$+_ \$+"\\"+_+\$_ _+\$\$\$\$

We start with this:

```
$=~[]; $={__:++$, $$$$: (![]+"") [$], __$:++$, $_
$: (![]+"") [$], _$:++$, $__$: ({+"") [$], $$_
$: ($[$]+"") [$], _$__:++$, $$$$: (!""+"") [$], $__:++$, $
$:++$, $$_: ({+"") [$], $$_:++$, $$$$:++$, $__:++$, $
_$:++$}; $._=($._="$+"") [$.$_] + ($._=$.$._[$.
_] + ($.$=$.$.$+"") [$.$_] + ((!$)+"") [$.$_] + ($
=$.$._[$.$_] + ($.$=(!""+"") [$.$_] + ($._=(!""+"") [$
_] + $.$._[$.$_] + $._.$+$.$.$;$.$=$.$.$+(!""+"")
[$.$_] + $._.$+$.$.$;$.$=(.$.$) [$.$_] [$.$
]; $.$($.$.$.$+"\\\"+ENCODED+"\\\"") ();
```

Note that the 'ENCODED' above does not appear in encoded files, rather it is the location where the encoded host code would appear. Also note that this algorithm does not work in direct mode (that is, putting it in a .js won't work) because it requires a feature that was introduced in HTML 4.0. As a result, it must appear in an HTML page, and that HTML page must declare its need for HTML 4.0 or later using a declaration like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

The 'HTML 4.0' string can be replaced by later versions, such as 'HTML 4.1' or 'XHTML 1.0', etc.

On to the code...

\$_

```
$=~[]
```

The expression '[]' returns a reference to an empty array. The operator '~' accesses the value at that reference ('0' in this case), and then inverts that value, resulting in the value '-1'. This value is assigned to the variable '\$'.

```
$={__:++$, $$$$: (![]+"") [$], __$:++$, $$_
$: (![]+"") [$], _$:++$, $__$: ({+"") [$], $$_
$: ($[$]+"") [$], _$__:++$, $$$$: (!""+"") [$], $__:++$, $
$:++$, $$_: ({+"") [$], $$_:++$, $$$$:++$, $__:++$, $
$:++$}
```

This line can also be written as follows:

```
$=
{
  __$:++$,
  $$$$: (![]+"") [$],
  __$:++$,
  $$_$: (![]+"") [$],
  _$:++$,
  $__$: ({+"") [$],
  $$$$: ($[$]+"") [$],
  __$:++$,
  $$$$: (!""+"") [$],
  $__:++$,
  $$_$:++$,
  $__$: ({+"") [$],
  $$$$:++$,
  $$$$:++$,
  $__:++$,
  $$_$:++$
}
```

The '{' and '}' signify the creation of an object, and each line between the braces creates a property and assigns it a value during the object construction. We'll examine the lines one at a time.

```
__$:++$
```

The expression '++\$' sets the value in the variable '\$' to '0' (specifically, it is incremented by 1, from '-1' to '0'). The ':' assigns a value to a property, and the property name is '__', so the property '__' is set to '0'.

```
$$$$: (![]+"") [$],
```

The expression '[]' returns a reference to an empty array, as above. The operator '!' tests if the reference is zero, which it is not, resulting in the Boolean value 'false'. The expression '+""' causes the Boolean value to be converted to a string, after which the empty string is appended to it. The result is the string 'false'. The expression '\$' causes the string to be converted to an array¹, and a single character to be returned. The variable '\$' has the value '0' from above, so the first character of the string 'false' ('f') is returned. That value is assigned to the property '\$\$\$'.

```
__$:++$,
```

The expression '++\$' sets to '1' the value of the variable '\$', and assigns that value to the property '_\$'.

```
$$_$: (![]+"") [$],
```

The second character of the string 'false' ('a') is assigned to the property '\$\$_'. The use of the expression '![]'

¹This is HTML-specific behaviour. Normally, a string cannot be converted to an array.

appears to be an oversight on the part of the author, since the expression '!\$' could have been used instead, now that the value of the variable '\$' is no longer zero. This change would have saved one byte.

```
_$_:++$,
```

The expression '++\$' sets to '2' the value of the variable '\$', and assigns that value to the property '_\$_'.

```
$_$$:({}+"" )[$],
```

The expression '{}' returns a reference to an empty object. As above, this reference is converted to a string. The result is the string '[object Object]'. The third character of the string '[object Object]' ('b') is assigned to the property '\$_\$\$'.

```
$$$_:($[$]+"" )[$],
```

The expression '\$[\$]' would access the third entry in the array specified by the variable '\$' if that array existed. However, since the variable '\$' is not an array, the value 'undefined' is returned. As above, this value is converted to the string 'undefined'. The third character of the string 'undefined' ('d') is assigned to the property '\$\$_\$'.

```
_$$:++$,
```

The expression '++\$' sets to '3' the value of the variable '\$', and assigns that value to the property '_\$\$'.

```
$$$_:(!""+"" )[$],
```

The expression '!' returns an empty string. The operator '!' tests if the string is zero, which it is, resulting in the Boolean value 'true'. As above, this value is converted to the string 'true'. The fourth character of the string 'true' ('e') is assigned to the property '\$\$\$_'. The use of the expression '!""+""' appears to be an oversight, since the string 'object' contains the letter 'e' immediately before the letter 'c'. Thus, this line could have been moved below the following line, and the expression '!""+""' could have been replaced with the expression '{}+""', to save one byte.

```
$__:++$,
```

The expression '++\$' sets to '4' the value of the variable '\$', and assigns that value to the property '\$__'.

```
$_$:++$,
```

The expression '++\$' sets to '5' the value of the variable '\$', and assigns that value to the property '\$_\$'.

```
$$__:({}+"" )[$],
```

The sixth character of the string '[object Object]' ('c') is assigned to the property '\$\$__'. The expression '({}+"")' is duplicated because it is not possible to reference a newly created property during the construction of an object. To reduce the size of the code, it would be necessary to use a

second variable, where the 'object' string could be stored prior to the construction of the '\$' object. Then the property assignment line would become '\$\$__var2[\$]', where 'var2' is the example name of the second variable.

```
$$_:++$,
```

The expression '++\$' sets to '6' the value of the variable '\$', and assigns that value to the property '\$\$_'.

```
$$$:++$,
```

The expression '++\$' sets to '7' the value of the variable '\$', and assigns that value to the property '\$\$\$'.

```
$__:++$,
```

The expression '++\$' sets to '8' the value of the variable '\$', and assigns that value to the property '\$__'.

```
$_$:++$,
```

The expression '++\$' sets to '9' the value of the variable '\$', and assigns that value to the property '\$_\$'.

At this point, we have the properties '___', '\$\$\$\$\$', '___\$', '\$_\$', '_\$_', '\$_\$\$', '\$_\$\$', '_\$\$', '\$\$\$', '\$_\$', '\$_\$', '\$\$_', '\$\$\$', '\$_\$\$', and '\$_\$\$'. They contain the values '0', 'f', '1', 'a', '2', 'b', 'd', '3', 'e', '4', '5', 'c', '6', '7', '8', and '9'.

\$\$\$+"\""+\$+\""

```
$.$_=($.$_=$+"" )[$.$_]+($.$_=$.$_[$.$_
$_])+(.$.$$=(.$.$+"" )[$.$_])+( (!$)+"" )[$.$_]+(.$._
=$.$_[$.$_])+(.$.$=(!""+"" )[$.$_])+(.$._=(!""+"" )[$.$_
_])+$.$_[$.$_]+$._.+$.$.$
```

This line contains multiple assignments to properties, and character concatenation. It can also be written as follows:

```
$.$_=
($.$_=$+"" )[$.$_]+
+($.$_=$.$_[$.$_])
+($.$$=(.$.$+"" )[$.$_])
+( (!$)+"" )[$.$_]
+($.$_=$.$_[$.$_])
+($..$=(!""+"" )[$.$_])
+($.._=(!""+"" )[$.$_])
+$.$_[$.$_]
+$. _
+$. _$
+$. $
```

The references to '\$' refer to the object now, not the value '9'. The '\$.' in front of each property is required to access an existing property.

```
(.$.$=$+"" )[$.$_]
```

The string '[object Object]' is assigned to the property '\$_'. The sixth character ('\$_' is '5') of the string

[object Object]’ (‘c’) is returned. There is a missed opportunity by the author here, since the ‘\$\$__’ property also contains the character ‘c’. Five bytes could have been saved by using that property instead, and moving the assignment of the ‘\$’ property to the following line.

```
+($._$=$._.$_[$._.$])
```

The second character (‘__’ is ‘1’) of the string [object Object]’ (‘o’) is assigned to the property ‘_’ and also returned.

```
+(.$.$=$(.$.$+"")[$._.$])
```

The non-existent property ‘\$’ is accessed, so the value ‘undefined’ is returned. This value is converted to a string, as above. The second character (‘__\$’ is ‘1’) of the string ‘undefined’ (‘n’) is assigned to the property ‘\$\$’ and also returned.

```
+((!$)+"")[$._.$$]
```

The expression ‘!\$’ tests if the reference to the object ‘\$’ is zero, which it is not, resulting in the Boolean value ‘false’. The value is converted to a string, as above, and the fourth character (‘_\$\$’ is ‘3’) of the string ‘false’ (‘s’) is returned. The parentheses surrounding the expression ‘!\$’ are unnecessary and could have been removed to save two bytes.

```
+(.$._=$._.$[$._.$$])
```

The seventh character (‘\$\$_’ is ‘6’) of the string [object Object]’ (‘t’) is assigned to the property ‘__’ and also returned. This line could have been written as ‘\$._’ if the line ‘__:(!\$+"")[\$]’ were placed in the object construction before the second ‘++\$’. However, this alternative saves no bytes.

```
+(.$.$=(!"+"")[$._.$$])
```

The string ‘true’ is constructed, as above. The second character (‘__\$’ is ‘1’) of the string ‘true’ (‘r’) is assigned to the property ‘\$’ and also returned.

```
+(.$._=(!"+"")[$._.$$])
```

The string ‘true’ is constructed, as above. The third character (‘_\$\$’ is ‘2’) of the string ‘true’ (‘u’) is assigned to the property ‘_’ and also returned. In this case, the entire line is poorly thought out, since the ‘_’ property could be assigned in the object constructor in a shorter way. Three bytes could have been saved by placing the line ‘_:(!\$+"")[\$]’ before the second ‘++\$’, which would access the first character of the string ‘undefined’.

```
+$.$_[$._.$$]
```

The sixth character (‘\$__\$’ is ‘5’) of the string [object Object]’ (‘c’) is returned. This appears to be another oversight since, as above, five bytes could have been saved by using the ‘\$\$__’ property instead. If the ‘c’ and ‘o’

characters were constructed using the alternative method, then the first assignment to the ‘\$’ property would be completely unnecessary, resulting in the saving of another five bytes.

```
+$.__
```

The value of the property ‘__’ (‘t’) is returned.

```
+$.__$
```

The value of the property ‘_’ (‘o’) is returned.

```
+$. $
```

The value of the property ‘\$’ (‘r’) is returned. The result is that the string ‘constructor’ is assigned to the property ‘\$’.

\$\$+"\"'+\$\$+"\"'

```
$. $$=$.$+(!"+"")[$._.$$]+$.__+$._.+$.$.$
```

This line contains more character concatenation. It can also be written as follows:

```
$. $$=
$. $
+(!"+"")[$._.$$]
+$. __
+$. _
+$. $
+$. $$
```

Once again, we will look at each line in turn:

```
$. $
```

The value of the property ‘\$’ (‘r’) is returned.

```
+(!"+"")[$._.$$]
```

The string ‘true’ is constructed, as above. The fourth character (‘_\$\$’ is ‘3’) of the string ‘true’ (‘e’) is returned. Again, the entire line appears to have been poorly thought out by the author, since nine(!) bytes could have been saved by using the ‘\$\$_’ property instead.

```
+$. __
```

The value of the property ‘__’ (‘t’) is returned.

```
+$. _
```

The value of the property ‘_’ (‘u’) is returned.

```
+$. $
```

The value of the property ‘\$’ (‘r’) is returned.

```
+$. $$
```

The value of the property ‘\$\$’ (‘n’) is returned. The result is that the string ‘return’ is assigned to the property ‘\$\$’. This assignment is completely unnecessary (see below).

```
$$$_+"\\"+_ $+$$ +$$ +$ $ _+(![]+"")[_ $ _]
```

```
$. $=( $. ___ ) [ $. $ _ ] [ $. $ _ ]
```

This translates to the expression `(0)[“constructor”][“constructor”]` [“constructor”], and is assigned to the property `$`. The use of the expression `($.__)` appears to be an oversight by the author, since the expression `[]` could have been used instead. This change would have saved five bytes. Further, by assigning to the property `$_` instead of `$` at a cost of two bytes, the five bytes that are required to assign the property `$$` and the four bytes that are required to reference it can be removed for an overall saving of seven bytes.

The expression `(0)[“constructor”][“constructor”]` is equivalent to the expression `0.constructor.constructor`, but the brackets are required to delimit the two strings. Otherwise, the expression would appear to reference a single property several levels deep (`$. $ _ . $ _`). The expression `<number>.constructor` is a reference to the constructor of a numeric object, while the expression `<object>.constructor.constructor` is a reference to the constructor of a generic object.

```
$. $ ( $. $ ( $. $ $ + "\ " + ENCODED + "\ " ) ) ( )
```

This line decodes and executes the encoded host code using two constructor calls. The first constructor call decodes the encoded host code, and the second constructor call executes it, in this way:

```
$. $ $ + "\ " + ENCODED + "\ "
```

The value of the property `$$` (`return`, however as noted above, this property reference can be replaced by the `return` concatenation from above) is used in the first constructor call to return the decoded host code that is bounded by the `""`s and represented here by `ENCODED`.

```
$. $ (return"ENCODED") ( )
```

This line translates to the expression `0.constructor.constructor(return“ENCODED”)()`, an anonymous function that returns the decoded host code as a string object. This is equivalent to executing the `eval()` function.

```
$. $ (DECODED) ( )
```

This line translates to the expression `0.constructor.constructor(DECODED)()`, an anonymous function that executes the decoded host code.

```
$$$_+$$+$$_+_$+$$ $+$$$_
```

Each character of the host code is encoded separately in one of several ways:

If the character is `''` or `\`, then the character is prepended with the `\` character (so `''` becomes `\'`, and `\` becomes `\\`).

If the character is a symbol already – that is, any of the following:

```
!"#$%&'()*+,-./:;<=>?@[^_`{|}~
```

then the character is used exactly as it appears.

If the character is numeric, or one of the letters `'a'` to `'f'`, `'o'`, `'t'`, or `'u'` (and the check is case-sensitive), then the appropriate property is used.

If the character is the letter `'l'`, then the expression `‘(![]+“”)[_ $ _]` (that is, the third character of the string `‘false’`) is used.

If the value of the character is less than 128, then the expression `\<val>` is used, where `<val>` is the decimal value of the character.

Otherwise, the expression `\u<val>` is used, where `<val>` is the hexadecimal value of the character.

It is interesting that some seemingly obvious encoding opportunities were missed. For example, since the numbers `'0'–'9'` are all available, it would be possible to use one of them to index the entire string for the special texts `‘false’`, `‘true’`, `[object Object]` and `‘undefined’` (the string `‘constructor’` exists, but all of the characters in that string are present in the other four strings; the string `‘return’` also exists, but all of the characters in that string are present in the strings `‘true’` and `‘undefined’`). Those four strings offer six more lower case alphabetic characters (`‘ijlnrs’`, leaving only `‘ghkmpqvwxyz’`), and only the numbers `'0'–'5'` are needed to access the entire set. If the host does not require all of the numbers, then several lines could be removed from the object construction code. That would allow the code to be shortened further, since some variables could then use shorter names.

CONCLUSION

As it stands, JEncode carries a relatively large constant body (even after applying the suggested size optimizations), which makes it easy to recognize. It would remain easy to recognize even if some ‘polymorphism’ were applied by using alternative indexes for the shared characters in the special texts (for example, the letter `'c'` is at position `'0'` in the word `‘constructor’`, and at position `'6'` in the string `‘[object]’`). It would also remain easy to recognize even if the variables were renamed as a result of discarding the unused numeric assignments. The only difficulty is in knowing at a glance what the encoded host does. However, the first constructor call (`$. $ (...)`) can be replaced with a function to display the result, or even to write it to disk, instead of executing it. Fortunately, the only way that someone could defend against that would be to change the code to the point where it is no longer JEncode.

TECHNICAL FEATURE

DLL HIJACKING

Aleksander P. Czarnowski
AVET INS, Poland

Is there any good reason to write any more about DLL hijacking? After all, by the time you read this all the hype about those vulnerabilities will (probably) be over... Who needs to read another article about a popular class of vulnerability?

Well, there are a few good reasons – the best one I can think of is that we don't learn from our mistakes. The first mistake is that, for a lot of people, security is not risk-driven but hype-driven. Of course it is important to remediate every vulnerability as quickly as possible, and a vulnerability in *iTunes* will receive more attention than a similar one in an *SVN* client.

The second mistake is that clearly documented and well described functionality can, after more than 10 years, suddenly become a vulnerability. Not only that, but it will also trigger a lot of research all around the world. The third mistake is that, while for the last couple of years there have been a few different attempts to build strong vulnerability taxonomies and dictionaries supporting them, we still have not learned how to fully exploit this knowledge.

Looking at the good old `CreateProcess()` problems one can easily imagine the DLL hijacking issue. So why did nobody see the danger earlier? In fact, the problem goes back as far as 1999/2000. In 1999, *Microsoft* published its MS99-006 advisory, and on 18 September 2000 Georgi Guninski posted the 'Microsoft Windows DLL search path weakness' advisory to the bugtraq mailing list.

CREATEPROCESS() VULNERABILITIES

When discussing the DLL hijacking issue one cannot forget about similar problems with `CreateProcess()`. As the name implies, the aim of this function is to create (and run) a new process. The definition of `CreateProcess` is as follows:

```

BOOL CreateProcess(
    LPCTSTR lpApplicationName,
    LPCTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);

```

The problem is that `lpApplicationName` should contain the name of the program (module) to be created by the system loader. Unfortunately, this parameter can also be set to `NULL`, which causes the system to interpret the first space delimited token from `lpCommandLine` as the module name. Imagine a call like this:

```

BOOL bOk = CreateProcess(NULL, "C:\\Program files\\
some_dir\\module.exe", [...]);

```

In such a case the system loader will try to expand tokens from `lpCommandLine` in order to find the first match of executable module location. Therefore dangerous combinations will be checked, such as `c:\Program.exe files\some_dir\module.exe`

If `Program.exe` exists in the `c:\` directory it will be executed even though the author of the code wanted to execute `module.exe` from `c:\Program files\some_dir\`. This behaviour is clearly described in [1].

It turns out that such insecure coding practices can lead to serious vulnerabilities, and we've seen a stream of exploits for this type of problem.

IMPORTING FUNCTIONS

On the *Windows* platform there are two legal ways of importing functions exported by dynamic link libraries:

- By PE file Import Address Table (IAT) – for the sake of this discussion we will omit delay load import tables and late binding.
- By `LoadLibrary()/GetProcAddress()` calls.

The import function does not exist in the caller module but can be loaded into the caller address space. Thus it is the job of the operating system image loader to parse the import table and load dynamic link libraries accordingly before passing execution to the main thread of the newly created process. This process can be observed with help from the *Windows* Debugging API (more on this later). The system loader finds the IAT by using the `OptionalHeader` member of the `IMAGE_NT_HEADERS` structure. The `IMAGE_OPTIONAL_HEADER` structure contains arrays of the `IMAGE_DATA_DIRECTORY` structure (there are 16 members). Member 12 contains the Import Address Table. It is worth remembering that some functions are not exported by name, but by ordinal numbers only.

The second option is based on `LoadLibrary/LoadLibraryEx`. These *Windows* API functions enable the loading of dynamic link libraries during runtime regardless of the content of the Import Address Table. `GetProcAddress` allows the address of the function within the loaded DLL to be acquired.

Several DLLs of the same name can exist within the filesystem as long as they are located in different folders. Furthermore, *Windows* even supports such a situation by providing a Dynamic Link Library Redirection mechanism. To enable it the user must create a redirection file which must follow the naming scheme: app_name.local.

If the application just calls `LoadLibrary`, passing only the DLL filename without the fully qualified path, then it leads to the DLL hijacking problem.

DLL HIJACKING VULNERABILITY

Microsoft provides several aids in loading dynamic link libraries. The most important libraries are specified in the KnownDLLs registry key: HKLM/System/CurrentControlSet/Control/Session Manager/KnownDLLs. In case of legacy 16-bit DLLs the correct key on *Windows XP/2000* is: HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\WOW. If an executable module wants to load a library from this list then the system image loader will know where to look for it and load the correct file. Inside KnownDLLs there is a DllDirectory key which specifies where the system should look for known DLLs (%SystemRoot%\system32 by default for 32-bit systems). In fact, this simple mechanism used to be vulnerable on the *Windows NT* platform (consult *Microsoft Security Bulletin MS99-006* [2] for details). The MS99-066 bulletin can be considered one of the grandfathers of the DLL hijacking attack vector.

DLL hijacking was possible due to the loading algorithm used by *Windows* in the case of an insecure `LoadLibrary()` call. *Microsoft* made the mistake of making the current directory first on the list of places to look for DLLs. This was fixed by the introduction of the SafeDllSearchMode registry value (HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode), which allows the DLL search process to be controlled in case the full path is missing. Quoting from *MSDN*: if SafeDllSearchMode is enabled, the search order is as follows:

1. The directory from which the application loaded.
2. The system directory. Use the `GetSystemDirectory` function to get the path of this directory.
3. The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
4. The Windows directory. Use the `GetWindowsDirectory` function to get the path of this directory.
5. The current directory.
6. The directories that are listed in the PATH environment variable. Note that this does not include

the per-application path specified by the App Paths registry key. The App Paths key is not used when computing the DLL search path.

If SafeDllSearchMode is disabled, the search order is as follows:

1. The directory from which the application loaded.
2. The current directory.
3. The system directory. Use the `GetSystemDirectory` function to get the path of this directory.
4. The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
5. The Windows directory. Use the `GetWindowsDirectory` function to get the path of this directory.
6. The directories that are listed in the PATH environment variable. Note that this does not include the per-application path specified by the App Paths registry key. The App Paths key is not used when computing the DLL search path.

Furthermore, the application can have some additional control on DLL loading either by calling `LoadLibraryEX` with the `LOAD_WITH_ALTERED_SEARCH_PATH` flag, or by calling `SetDllDirectory`. Unfortunately, many applications don't use either method and lazy programmers issue `LoadLibrary` with just a DLL name.

When the first matching DLL filename has been found by the system image loader, *Windows* abandons any further search. This 'first find wins' strategy allows an attacker to plant a DLL in the directory that is searched before the one containing the legal library if the application is loading a DLL only using the filename. What is even more important is that network shares can also be searched for DLLs.

DLL HIJACKING DETECTION

In [3] the authors describe detection methods not only for the *Windows* platform but also for other operating systems. On *Windows*, detection using dynamic analysis is a simple process thanks to the availability of the Debugging API and a great set of debuggers like *OllyDBG* and *IDA Pro*. We just need to hook the `LoadLibrary` call and inspect the first argument passed to it. The definition of the `LoadLibrary` function is as follows:

```
HMODULE WINAPI LoadLibrary(
    __in LPCTSTR lpFileName
);
```

The above definition comes from the *Windows* platform SDK. However, `Kernel32.dll` exports two versions of this function:

```

.text:00432260 ; ===== SUBROUTINE =====
.text:00432260
.text:00432260
.text:00432260 sub_432260 proc near ; CODE XREF: sub_431E10+3A61p
.text:00432260 push offset LibFileName ; "C0DBCLog.dll"
.text:00432265 call ds:LoadLibraryA
.text:00432268 mov hModule, eax
.text:00432270 test eax, eax
.text:00432272 jz short locret_4322E3
.text:00432274 push offset aInitializing0d ; "Initializing 008C log module..."
.text:00432279 mov ecx, dword_44AEF0
.text:0043227F push 40h
.text:00432281 call sub_42E000
.text:00432286 push offset a?Initialize@ay ; "?Initialize@VAHXZ"
.text:00432288 mov eax, hModule
.text:00432290 push eax ; hModule
.text:00432291 call ds:GetProcAddress
.text:00432297 test eax, eax
.text:00432299 jnz short loc_4322A2
.text:0043229B push offset aFailedToGetPoi ; "Failed to get pointer to Initialize()"
.text:0043229D jmp short loc_4322AD

```

Figure 1: Example of passing a static string as lpFileName to the LoadLibraryA function.

1. LoadLibraryA (ANSI)
2. LoadLibraryW (Unicode)

Therefore we need to hook these function calls (don't forget about LoadLibraryExW and LoadLibraryExA) in order to catch all possible DLLs loading during runtime. Next we run our module, catch all LoadLibrary* calls and inspect the lpFileName argument for the full, proper path definition. If the path location is invalid or missing we have found a vulnerability. Theoretically, this makes the detection process trivial, allowing almost anyone to find such a vulnerability (which posts to the bugtraq mailing list seem to confirm).

Unfortunately, finding a vulnerability and proving that there is no such vulnerability in a module are two completely different things. The problem lies in the code coverage and execution flow. Until we can prove that all execution paths that call LoadLibrary* functions have been covered by our analysis, we cannot claim that a module is not vulnerable. This problem can partially be solved with the help of static analysis as we can enumerate all LoadLibrary* calls within the module and then enumerate all cross references to those functions or methods. Most LoadLibrary() calls are made with a static value of lpFileName. If all calls are made with static names (as in Figure 1) than we can perform all checks using only static methods.

Returning to a dynamic analysis approach, hooking LoadLibraryA and LoadLibraryW calls is only one possible method. Hooking can be done with *Microsoft Detours* library, int 3 breakpoints or hardware breakpoints. However, a much better approach is to use the *Windows Debugging API*:

1. Start the debugging process with CreateProcess() with the DEBUG_PROCESS flag.
2. Start module execution.
3. Process the debug event with WaitForDebugEvent().

4. Check for LOAD_DLL_DEBUG_EVENT and process it.
5. ContinueDebugEvent() in order to resume process execution.
6. When EXIT_PROCESS_DEBUG_EVENT has been caught, quit the debugging loop.

Another option is proposed in [3], based on the LdrLoadDll function from ntdll.dll. To equip the binary in order to trace system image loader activity we don't need any special tools besides WinDbg [4]:

1. Run gflags.exe from the WinDbg main directory.
2. Click on the Image File tab and enter the image filename.
3. Press the tab key to enable the checkbox options and select 'Show loader snaps', as in Figure 2.
4. Click OK to dismiss the dialog box.
5. Run WinDbg and select Open Executable (Ctrl+E). In the file dialog box choose notepad.exe and load it.
6. You will see the list of loaded modules and then debug information from the system image loader, as shown in Figure 3.

There is one interesting call: LdrLoadDll. Inspection of this function reveals that it calls another function call, LdrpLoadDll. This is the work horse that does most of the

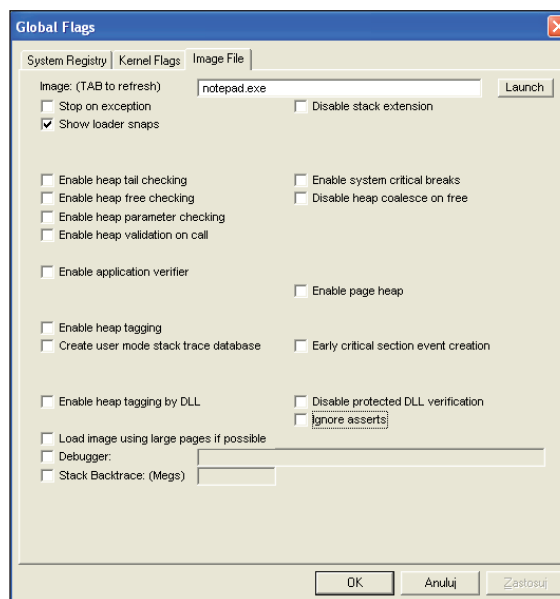


Figure 2: Select the 'Show loader snaps' option.

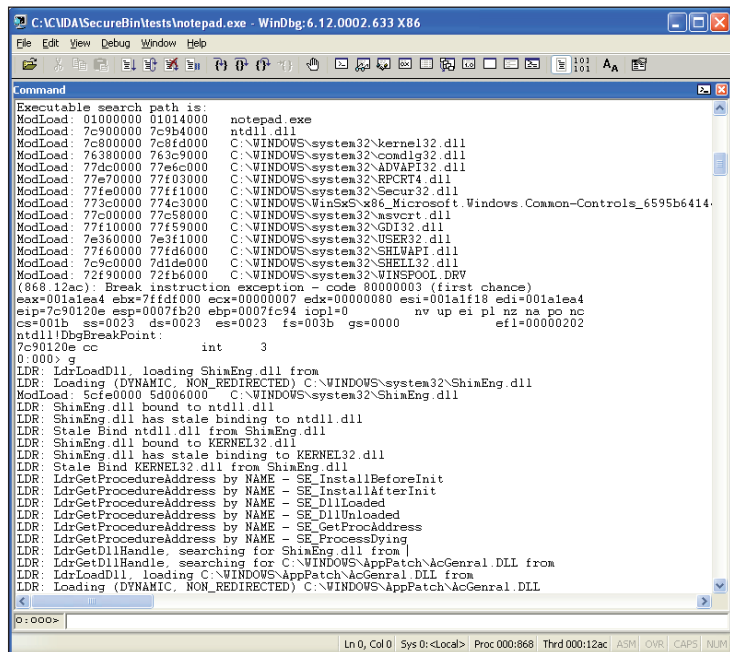


Figure 3: Using WinDbg to catch system image loader activity.

Address	Called function
1 .text:7C9163CD	call _SEH_prolog
2 .text:7C916424	call RtlpDphDisableFaultInjection@0; RtlpDphDisableFaultInjection()
3 .text:7C91646C	call RtlDosApplyFileIsolationRedirection_Ustr@36; RtlDosApplyFileIsolationRedirection_Ustr(x,x,x,x,x,x,x,x,x)
4 .text:7C91649C	call _LdrLockLoaderLock@12; LdrLockLoaderLock(x,x,x)
5 .text:7C9164CE	call _LdrLoadDll@24; LdrLoadDll(x,x,x,x,x,x,x)
6 .text:7C9164E7	call loc_7C91661F
7 .text:7C9164FA	call RtlpDphEnableFaultInjection@0; RtlpDphEnableFaultInjection()
8 .text:7C916504	call @_security_check_cookie@4; _security_check_cookie(x)
9 .text:7C916509	call _SEH_epilog
10 .text:7C91662D	call _LdrUnlockLoaderLock@8; LdrUnlockLoaderLock(x,x,x)
11 .text:7C935587	call _local_unwind2
12 .text:7C93825E	call _DbgPrintEx
13 .text:7C93C85D	call _DbgPrint
14 .text:7C93C88B	call _DbgPrint
15 .text:7C93C8BE	call _DbgPrint
16 .text:7C93C8FD	call _DbgPrint
17 .text:7C93C92F	call _DbgPrint
18 .text:7C93C943	call _RtlFreeAnsiString@4; RtlFreeAnsiString(x)

Figure 4: LdrLoadDll (Windows XP SP3) call list generated by IDA Pro.

work during the loading of the DLL and mapping it into the process address space. The complete list of calls by the LdrLoadDll function is presented in Figure 4.

Another detection option is to use Process Monitor and catch all CreateFile and LoadImage operations with paths containing .dll, .DLL, .sys and .SYS. Exclude events with SUCCESS results and paths that end with pagefile.sys. Now run Process Monitor and look for failed DLL load attempts.

When considering detection approaches keep in mind that there is a set of applications that ‘dislike’ being debugged.

Several copy protection schemes are good examples. The most simple check is to invoke IsDebuggerPresent(), which can easily be bypassed either by patching this function, changing the return value or changing the IsDebuggerPresent flag in the process PEB. Of course there are many other anti-debugging tricks around [5–19]. The point here is that some applications cannot be instrumented easily with the Debugging API and other approaches like Process Monitor must be used. However, the best detection option is to search the source code for LoadLibrary* calls.

EXPLOITATION PROCESS

Another thing which makes DLL hijacking or binary planting (as it’s called in the original ACROS advisory) so similar to the CreateProcess() vulnerability is the ease with which it can be exploited. An attacker just needs to ‘plant’ his own DLL in the proper folder so that his library is loaded first (the second – real – one never gets its chance to load). Wouldn’t it be easier simply to overwrite the original DLL file? The answer is yes, and no. An attacker might not have write privilege, or else Windows Resource Protection (the newer version of Windows File Protection on Vista/Windows 2008 Server systems) might protect files from being overwritten. However, the attacker might have write access to a directory which will be searched for DLLs before inspecting those protected by the Windows security model. So, in the end, DLL hijacking can provide some value to the attacker, both for local and remote attacks.

In order to conduct the attack an attacker needs his own DLL. This can be written in any language supporting DLL files. A simple DLL template written in assembly language is presented below:

```

format PE DLL
entry DllEntryPoint

include 'win32ax.inc'

section '.text' code readable executable

proc DllEntryPoint hinstDLL, fdwReason, lpvReserved

    cmp [fdwReason], DLL_PROCESS_ATTACH
    je pattach
    cmp [fdwReason], DLL_PROCESS_DETACH
    je pdetach
    cmp [fdwReason], DLL_THREAD_ATTACH
    je tattach
    cmp [fdwReason], DLL_THREAD_DETACH
    je tdetach

exit:
    mov     eax, TRUE
    
```

```

        ret

pattach:
    call exploit
pdetach:
tattach:
tdetach:
    jmp exit
endp

proc exploit
    mov edi, edi    ;simulate hotpatching entry

    nop            ;make space/call for the debugger or
    int 3          ;detour if not using mov edi, edi for
    ;it
    nop

    invoke MessageBox, NULL, 'DLL Hijacker:
exploit', 'Exploited', MB_ICONERROR+MB_OK
    ret
endp

section '.idata' import data readable writeable

    library kernel, 'KERNEL32.DLL', \
        user, 'USER32.DLL'

    import user, \
        MessageBox, 'MessageBoxA'

```

This can be compiled using FASM [20]. Of course, for more complicated DLLs, C/C++ might be a better option. The reason for choosing assembly language in the first place was its small output size and ability to insert shellcode in place of the MessageBox call. However, for testing or demonstrating vulnerabilities, the MessageBox call will do its job perfectly. Now you just need to plant the DLL and find a vulnerable application. The approaches discussed so far should be enough to get started. There are already tools that automate the whole process. A good example is DLLHijackAuditKit v2 from the MetaSploit project [21]. This kit will build test cases for DLLs in your system (01_StartAudit.bat) and generate proof-of-concept 'exploits' executing calc.exe for vulnerable cases (02_Analyze.bat).

DEFENCE STRATEGIES

All of the attack vectors mentioned have resulted in the addition of new features to the *Windows* operating system over time:

- Windows Resource Protection
- The SafeDllSearchMode registry key
- The SetDllDirectory function
- The SetSearchPath function
- The SetSearchPathMode function

- The CWDIllegalInDllSearch registry key

SafeDllSearchMode is enabled by default on recent *Windows* systems. This key controls the DLL search order. The key HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode should be set to 1 to enable this feature. How this setting affects the search order has already been described in the 'Dll hijacking vulnerability' section.

The SetSearchPathMode function is a newly created API to allow *IE* to force the current directory to be searched after the system location has been checked. Of course, nothing stops programmers from using it in their own applications. Similar to SetSearchPath, SetSearchPathMode affects only the current process and has no impact on other running processes.

The CWDIllegalInDllSearch registry key enables a system administrator to:

- Remove the current directory from the search path when loading DLLs
- Disable DLL loading from the WebDAV location
- Disable DLL loading from WebDAV and remote UNC address locations.

The settings above can be applied system-wide or on a per-application basis. To use the CWDIllegalDllSearch key it must be added to:

- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager in order to enable it for the whole system
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\<application binary name> to enable it for a specified application only.

Values for this key are described in [22].

Obviously, the best defence is the secure calling of the LoadLibrary() and LoadLibraryEx() functions. The safeguards mentioned previously will not address all possible vulnerabilities and some options cannot be deployed in certain configurations, for example. This brings us to the point where patching vulnerable applications is the best safeguard possible. In order to fix this type of vulnerability efficiently it must be addressed on a source code level. The following are some tips for developers on using LoadLibrary* API:

- Always specify the fully qualified path.
- Remove the current directory from the DLL search path by using SetDllDirectory with an empty string during application initialization.

- If possible, use DLL redirection or a manifest to ensure the proper DLL will be loaded.
- SearchPath should not be used to locate a DLL unless safe process search mode is enabled. Avoid using SearchPath if possible.
- Never base your assumption about system version/ service pack level on successful DLL loading. Use the GetVersionEx() function and base your assumption on its results.
- Enable safe DLL search mode.

SUMMARY

In the end, the whole DLL hijacking story wasn't so dull after all. It made me look inside LdrLoadDll and browse through ntdll.dll, which is always fun and you can always learn something new. It also demonstrated a few strong points about security. The most important one – from both a customer's and a developer's perspective – is that, while easy to detect, vulnerabilities might not be easy to fix at the operating system level. Secondly, it is possible for automatic or manual detection of simple vulnerabilities to fail and provide false results.

REFERENCES

- [1] CreateProcess function. <http://msdn.microsoft.com/en-us/library/ms682425>.
- [2] Microsoft Security Advisor Program: Microsoft Security Bulletin (MS99-006). <http://www.microsoft.com/technet/security/bulletin/ms99-006.msp>.
- [3] Kwon, T.; Su, Z. Automatic Detection of Vulnerable Dynamic Component Loadings, University of California.
- [4] Debugging Tools for Windows. <http://www.microsoft.com/whdc/devtools/debugging/default.msp>.
- [5] Ferrie, P. Anti-unpacker tricks. <http://pferrie.tripod.com/papers/unpackers.pdf>.
- [6] Ferrie, P. Virus Bulletin December 2008. Anti-unpacker tricks – part one. <http://www.virusbtn.com/pdf/magazine/2008/200812.pdf>.
- [7] Ferrie, P. Virus Bulletin January 2009. Anti-unpacker tricks – part two. <http://www.virusbtn.com/pdf/magazine/2009/200901.pdf>.
- [8] Ferrie, P. Virus Bulletin February 2009. Anti-unpacker tricks – part three. <http://www.virusbtn.com/pdf/magazine/2009/200902.pdf>.
- [9] Ferrie, P. Virus Bulletin March 2009. Anti-unpacker tricks – part four. <http://www.virusbtn.com/pdf/magazine/2009/200903.pdf>.
- [10] Ferrie, P. Virus Bulletin April 2009. Anti-unpacker tricks – part five. <http://www.virusbtn.com/pdf/magazine/2009/200904.pdf>.
- [11] Ferrie, P. Virus Bulletin May 2009. Anti-unpacker tricks – part six. <http://www.virusbtn.com/pdf/magazine/2009/200905.pdf>.
- [12] Ferrie, P. Virus Bulletin June 2009. Anti-unpacker tricks – part seven. <http://www.virusbtn.com/pdf/magazine/2009/200906.pdf>.
- [13] Ferrie, P. Virus Bulletin May 2010. Anti-unpacker tricks – part eight. <http://www.virusbtn.com/pdf/magazine/2010/201005.pdf>.
- [14] Ferrie, P. Virus Bulletin June 2010. Anti-unpacker tricks – part nine. <http://www.virusbtn.com/pdf/magazine/2010/201006.pdf>.
- [15] Ferrie, P. Virus Bulletin July 2010. Anti-unpacker tricks – part ten. <http://www.virusbtn.com/pdf/magazine/2010/201007.pdf>.
- [16] Ferrie, P. Virus Bulletin August 2010. Anti-unpacker tricks – part eleven. <http://www.virusbtn.com/pdf/magazine/2010/201008.pdf>.
- [17] Ferrie, P. Virus Bulletin September 2010. Anti-unpacker tricks – part twelve. <http://www.virusbtn.com/pdf/magazine/2010/201009.pdf>.
- [18] Ferrie, P. Virus Bulletin October 2010. Anti-unpacker tricks – part thirteen. <http://www.virusbtn.com/pdf/magazine/2010/201010.pdf>.
- [19] Ferrie, P. Virus Bulletin November 2010. Anti-unpacker tricks – part fourteen. <http://www.virusbtn.com/pdf/magazine/2010/201011.pdf>.
- [20] Flat Assembler. <http://flatassembler.net/>.
- [21] DLLHijackAuditKit. v2 <http://blog.metasploit.com/2010/08/better-faster-stronger.html>.
- [22] A new CWDIllegalInDllSearch registry entry is available to control the DLL search path algorithm. <http://support.microsoft.com/kb/2264107/en-us>.

TUTORIAL

INTRODUCTION TO HOSTILE JAVA ANALYSIS

Ed Jones

Independent researcher, USA

Java is a powerful platform-independent programming language that is widely used within web applications and mobile media. Fraudsters have abused Java to obfuscate attacks, hinder research and response, and maximize profits. Security experts must have an understanding of common Java-based attacks and their implications in order to best respond to emerging threats in the wild.

INTRODUCTION TO JAVA

The Java programming language was developed by *Sun Microsystems* and first released in 1995. Its strength is that it is platform independent. As a result, a Java program can be deployed to many operating systems as a standalone solution. For example, a Java program may be authored and placed within a web medium, then be able to be downloaded and run on *Windows*, *Macintosh* and Unix machines.

Source files normally have the extension ‘.java’, while compiled Java files use the extension ‘.jar’. JAR files (Java ARchives) are an aggregation of class files and metadata such as images and text used within a Java application. JAR files can be unpacked using tools like *WinZip*, as they contain a PK header. Security experts analysing hostile JAR files must first unpack the sample and then look to decompile the class files found within the archive. Figure 1 shows the header and class string references found in a JAR file for a hostile exploitation component used within the *Eleonore* exploit kit.

Notice that the meta-info and manifest data exist within the strings of this file. This is also a visual cue for identifying Java content, as JAR files always include both class files and a manifest with metadata.

Java source code is compiled to create what is known as a class file. Class files contain executable content for Java.

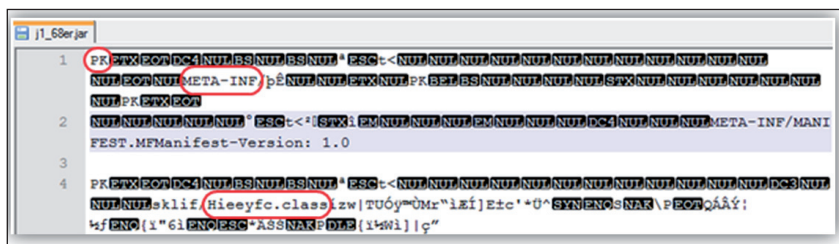


Figure 1: Hostile JAR file has PK header and strings of interest.

Class files may also contain multiple files within them, known as inner classes. Security experts researching hostile Java applets (small web applications) look to capture, decompile and analyse code related to all class files found within a hostile JAR file.

JNANA JAVABOT

Jnana Javabot is a prime example of the advancing nature of malicious Java usage. The code was first reported publicly by *Symantec* in October 2010 [1]. Jnana Javabot is a new botnet that uses Java as the command and control (C&C) infrastructure, making it platform independent. It already has modular payloads for *Windows* and *Macintosh* and could easily be extended to *Droid* and other platforms of interest.

Jnana Javabot also leverages tactics seen with other former major threats, which clearly indicates that the developers are current and progressive and/or possibly affiliated in some way with these former threats. Like *Zlob*, Jnana Javabot contains a fake codec trick to spread via *Facebook*. It also utilizes a complicated domain generation algorithm similar to that seen in *Conficker*, and unique P2P features reminiscent of the infamous *Storm* worm.

JRE ATTACKS

While Jnana Javabot sets the stage for botnets of the future – including mobile platforms – current threats related to Java primarily include exploitation of Java itself and hostile Java applets.

The Java Runtime Environment (JRE) is one of the most popular targets for criminals to exploit. It has widespread popularity in the underground, and is considered one of the best new vectors of opportunity on machines that might not otherwise be compromised. Multiple new exploits for Java enable criminals to compromise computers of interest that are not patched.

Enterprise networks regularly use older, unpatched versions of Java because of business requirements to run specific legacy versions for compatibility with proprietary solutions. To make matters worse, many of these legacy-based Java solutions run on critical servers with important assets at risk. Such exploitation leads to a variety of possible payloads for the system that is compromised.

Exploitation of JRE is frequently performed through malicious JAR

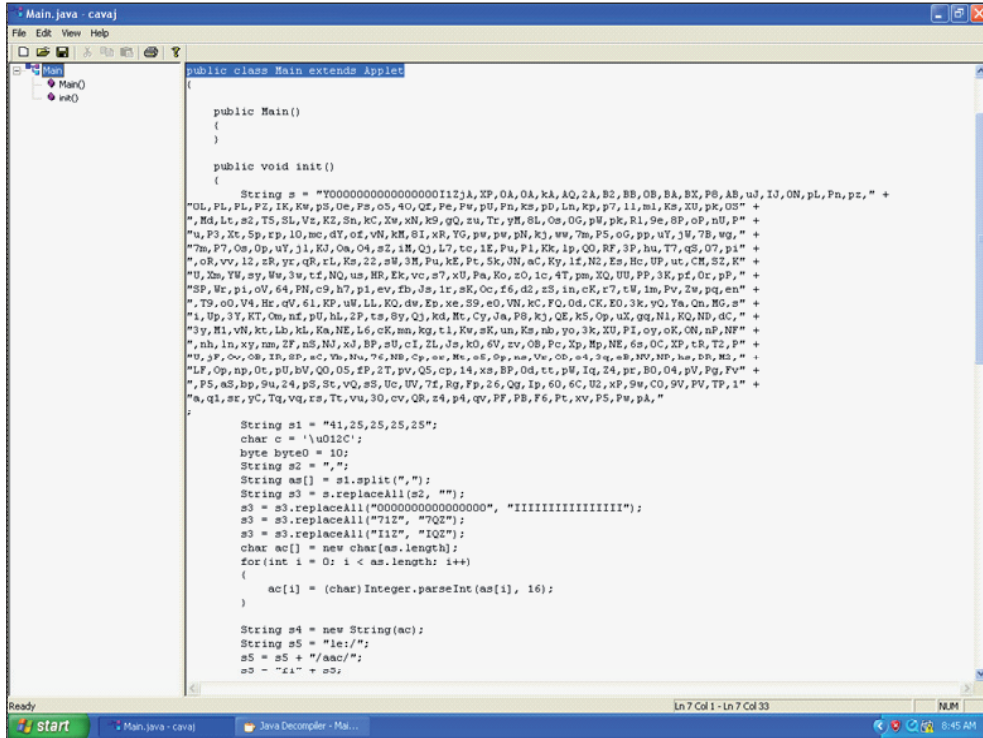


Figure 2: Cavaj freeware default view.

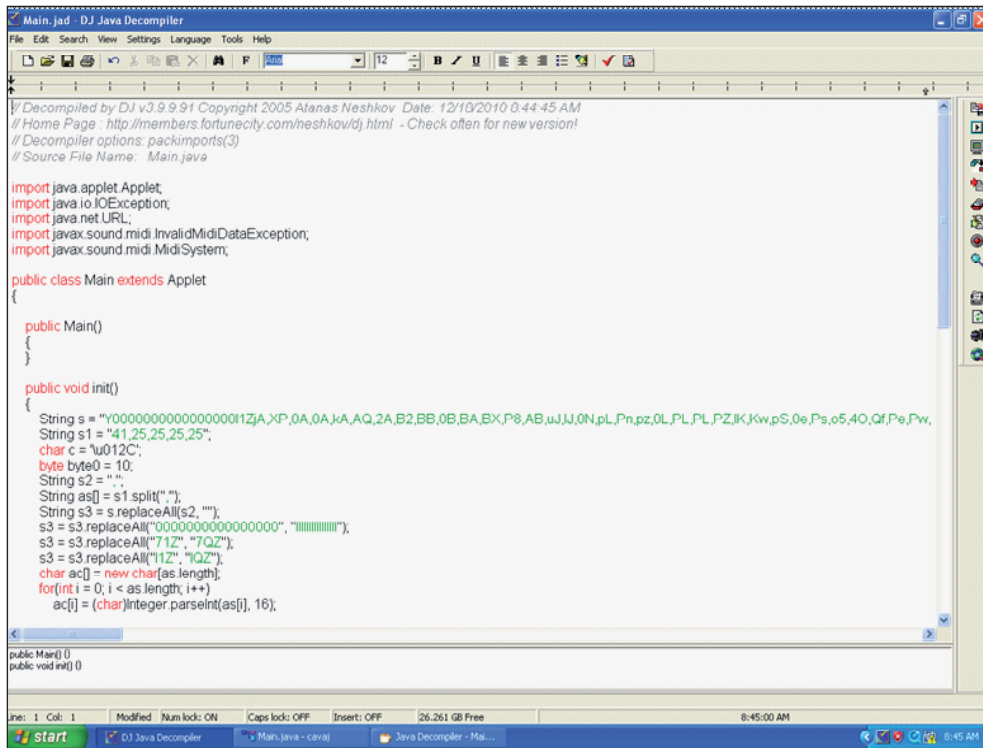


Figure 3: DJ Java Decompiler default source code view.

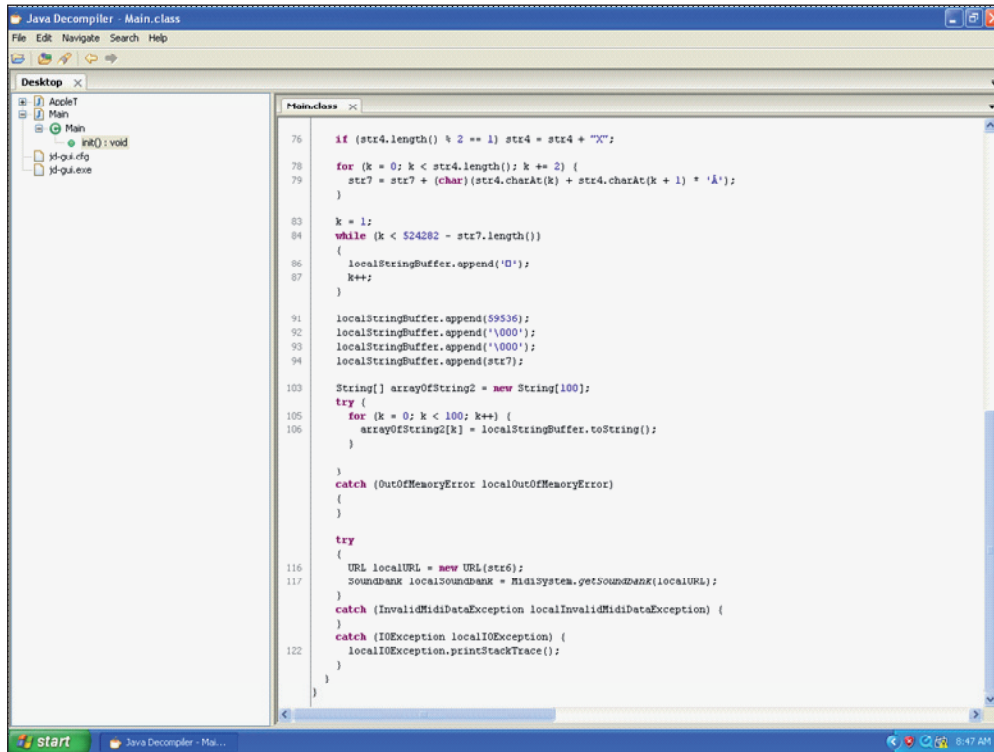


Figure 4: Java Decompiler default view.

files containing hostile Java applets designed to exploit a vulnerability on the remote computer.

HOSTILE JAVA APPLETS

Java applets are small, web-based applications. Instead of having code that is readily visible, such as in JavaScript-based attacks, the code is compiled within a Java file.

An investigation into hostile Java artefacts commonly begins with the capture of a questionable JAR file. Once the JAR file has been properly analysed additional research and response is initiated to better understand the JRE exploit vector or behaviour/intent of the hostile Java attack.

A wealth of applications are available to work with JAR files. One such tool is the *Mobilefish* Java class decompiler [2]. This free, web-based tool provides an amazing amount of analysis for a submitted file. Simply browse for the file to analyse, complete a CAPTCHA input entry, and click on ‘decompile’. A wealth of related Java information is also available on the *Mobilefish* site [3, 4].

Some of the easiest WYSIWYG Windows-based GUI tools include *Cavajdemo*, *DJ Java Decompiler* (djdec39) and *Java Decompiler* (JD-GUI). Unlike command-line

tools, these offer solid decompiling capabilities along with an organization of functions and/or colour-coding of decompiled scripts.

Cavajdemo requires a set-up program to run and then for the user to locate the installed application in the Program Files directory to run it, but is worth the price of admission: it’s free. *DJ Java Decompiler* also requires a set-up and is not freeware, but it does include a nice option to switch between source code and byte code views. *Java Decompiler* (JD-GUI) is the easiest to use, with drag-and-drop functionality and no set-up required, and it is free. It also contains a ‘Save Sources’ menu option for exporting all decompiled scripts.

After loading a class file of interest into *Java Decompiler* it is easy for the analyst to identify the primary functions

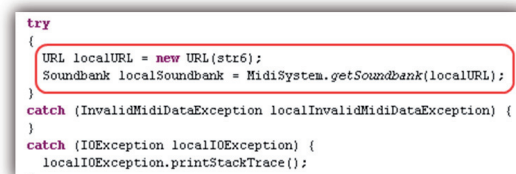


Figure 5: Strings of interest in questionable JAR and class file.

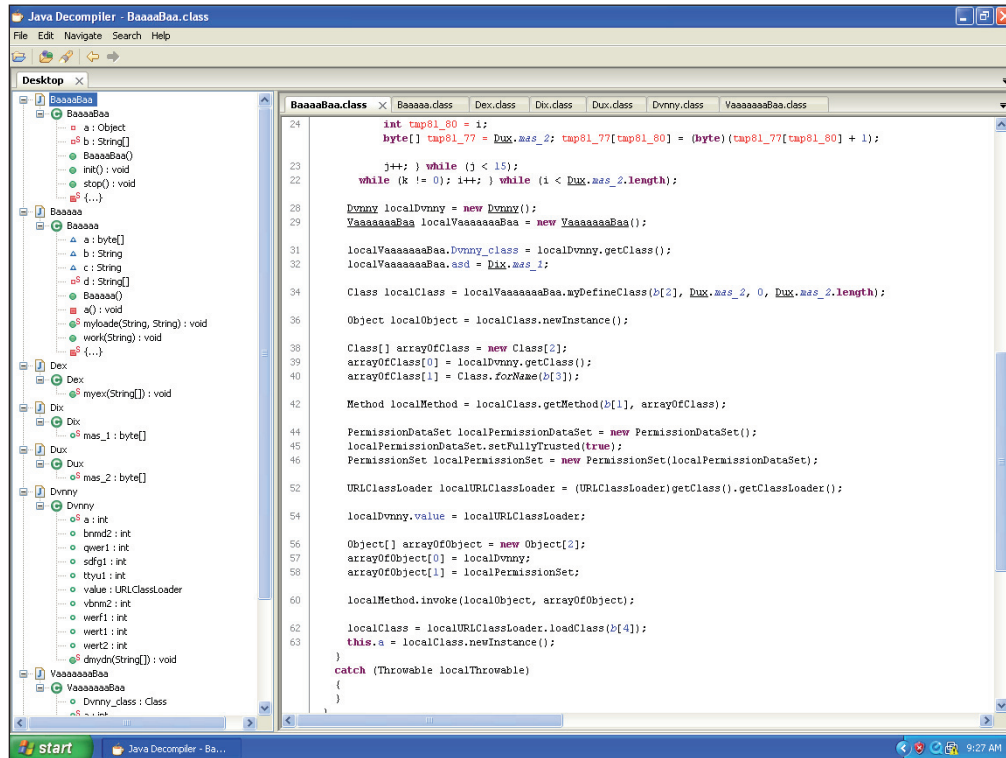


Figure 6: Analysis of hostile class files.

of the script indented by the tool automatically ('Main()' and 'init()' in this example). The analyst may then review the decompiled source code looking for URLs (not often seen in 2010 and later), CLSID (Windows Class Identifier) values possibly related to exploitation vectors, and similar strings of interest. In this example, the strings shown in Figure 5 are found near the bottom of the code.

The text 'soundbank' is linked to URL activity. Within the context of malicious code attacks this is a string of significant interest. The first result in an Internet search for 'soundbank java exploit' identifies a JRE vulnerability for CVE-2009-3864. When this is looked up on the Mitre website [5] it is clear that the file under analysis probably exploits a JRE vulnerability impacting JDK/JRE 5.0 before update 22 and JDK/JRE before update 17 when a non-English version is used. This information can be cross-checked with other research and response data to better qualify this possible threat vector. Additionally, the exact versions of the software on the possibly compromised machine may be compared to what is known to be vulnerable for this vector of attack.

If the machine is found to be running the software versions that are known to be vulnerable to this exploit vector, additional work may be performed to further qualify the

threat, including anti-virus scanning, behavioural tests, and reverse engineering.

CASE STUDY EXAMPLE

In 2008, a hostile iFRAME at hxxp://www.psu.com/poll.php led to a CGI redirection to hxxp://asvsutra.info/in.cgi?7. This then led to hxxp://liveinternets.com/all/update.php containing two layers of obfuscation leading to nine exploit pages, eight of which were functional at the time of the incident. One of the exploit vectors was an Exploit.Byteverify (MS03-011) attempt via java.php. An analysis of hostile artefacts found on a compromised machine included several class files of interest:

- Baaaaa.class
- BaaaaBaa.class
- Dex.class
- Dix.class
- Dux.class
- Dvnnny.class
- VaaaaaaaBaa.class

Anti-virus scans of these files led to detection names for ClassLoader (ByteVerify). An analysis of the decompiled code also reveals the same functionality (Figure 6).

A search for the MD5 values of each class file also resulted in a VX Heavens match for a ClassLoader trojan (ByteVerify exploitation), confirming this component of the incident involving Java. This information arms security personnel with at least one component of exposure during the attack. Security teams are then able to identify the method of exploitation to patch and/or harden against it in addition to following up on exploit mitigation and finding other machines that may be vulnerable to the same attack.

CONCLUDING REMARKS

Early Java-based threats were simple tricks to move hostile URLs out of JavaScript and HTML-type environments into compiled Java files. Java-based attacks have greatly matured since the early days of exploitation, now frequently including many layers of redirection and obfuscation and the use of many artefacts to hinder research and response. Criminals are also leveraging Java to manage their own platform-independent attacks as well as exploit vulnerable versions of JRE.

Java-based threats have never been more real and likely than they are today, making it essential for all incident response teams to have a good understanding of this threat vector and knowledge of how to perform initial analysis of such attacks. More importantly, all security staff should be prioritizing Java-based security measures, given the widespread exploit vectors available for various versions of JRE and the popularity of this vector amongst criminals exploiting the drive-by vector.

REFERENCES

- [1] Trojan.Jnanabot: Trojan Affecting Multiple Platform. <http://www.symantec.com/connect/blogs/trojanjnanabot-trojan-affecting-multiple-platforms>.
- [2] Mobilefish Online Java class decompiler. http://www.mobilefish.com/services/java_decompiler/java_decompiler.php.
- [3] The Java Community Process Program. JSRs: Java Specification Requests. <http://jcp.org/en/jsr/detail?id=20>.
- [4] Mobilefish Java Quick Guide. http://www.mobilefish.com/tutorials/java/java_quickguide_classfile.html.
- [5] CVE-2009-3864. <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3864>.

CALL FOR PAPERS

VB2011 BARCELONA

Virus Bulletin is seeking submissions from those wishing to present papers at VB2011, which will take place 5–7 October 2011 at the Hesperia Tower hotel, Barcelona, Spain.



The conference will include a programme of 30-minute presentations running in two concurrent streams: Technical and Corporate.

Submissions are invited on all subjects relevant to anti-malware and anti-spam. In particular, *VB* welcomes the submission of papers that will provide delegates with ideas, advice and/or practical techniques, and encourages presentations that include practical demonstrations of techniques or new technologies.

A list of topics suggested by the attendees of VB2010 can be found at <http://www.virusbtn.com/conference/vb2011/call/>. However, please note that this list is not exhaustive, and the selection committee will consider papers on these and any other anti-malware and anti-spam related subjects.

SUBMITTING A PROPOSAL

The deadline for submission of proposals is **Friday 11 March 2011**. Abstracts should be submitted via our online abstract submission system. You will need to include:

- An abstract of approximately 200 words outlining the proposed paper and including five key points that you intend the paper to cover.
- Full contact details.
- An indication of whether the paper is intended for the technical or corporate stream.

The abstract submission form can be found at <http://www.virusbtn.com/conference/abstracts/>.

One presenter per selected paper will be offered a complimentary conference registration, while co-authors will be offered registration at a 50% reduced rate (up to a maximum of two co-authors). *VB* regrets that it is not able to assist with speakers' travel and accommodation costs.

Authors are advised that, should their paper be selected for the conference programme, they will be expected to provide a full paper for inclusion in the VB2011 Conference Proceedings as well as a 30-minute presentation at VB2011. The deadline for submission of the completed papers will be Monday 6 June 2011, and potential speakers must be available to present their papers in Barcelona between 5 and 7 October 2011.

Any queries should be addressed to editor@virusbtn.com.

FEATURE

THE TOP 10 SPAM, MALWARE AND CYBERSECURITY STORIES OF 2010

Terry Zink
Microsoft, USA

2010 was a year filled with plenty of security stories – spam, malware and general security topics all hit the headlines¹. It was a jam-packed year, so let's take a look at the biggest newsmakers. (Please note that the views and opinions expressed in this article are the author's own and do not necessarily state or reflect those of *Microsoft*.)

1. RUSTOCK CLOGS UP BANDWIDTH AND THEN STOPS

This story began in December 2009. It was then that Rustock, the world's largest botnet, first started sending spam over Transport Layer Security, or TLS.

TLS is a protocol that is used as a form of security in email by encrypting the communication channel between the sender and the receiver². Usually, it is used between two legitimate parties who do not want anyone to eavesdrop on their communication. Like any encryption protocol, TLS is computationally expensive. It requires the sender and receiver to negotiate a certificate exchange and uses up quite a bit more computation resources. This reduces both the number of messages a sender can transmit and the number of messages a receiver can receive.

This is what makes Rustock's behaviour puzzling. Rustock is known to send large bursts of messages, but only about one per email envelope. It also sends from a very wide swathe of IP addresses. By sending spam over TLS, Rustock was effectively limiting the amount of mail it could send. Since spam requires very wide distribution in order to be effective, this behaviour seemed counterintuitive. Why would Rustock start doing this? Perhaps its creators wanted the messages to be seen as legitimate – a receiver might consider a sender over TLS more likely to be legitimate (on the assumption that a botnet is unlikely to devote the resources to using the protocol). Another possibility is that they wanted to encrypt all of the communication channels from nodes to bot-controllers, and this logically extended to the sending of email as well. Ultimately, why they did it is still unclear.

Yet as mysteriously as it started, Rustock stopped sending spam over TLS about halfway through the year. It's possible

¹ In this article, when I use the term 'spammers', I use it in a generic sense to refer to people who send spam, distribute malware, perform black search engine optimization, etc.

² See <http://www.wisegeek.com/what-is-tls.htm> for more details.

that when its creators discovered that sending spam over TLS didn't improve delivery but mostly impeded it, they abandoned the encryption protocol in favour of the efficiency of sending mail over plain text. In any case, they reverted to type and continued to spam as usual.

2. SPAMHAUS RELEASES A WHITELIST

Spamhaus is best known for its work in categorizing IP space with a bad reputation. It is very well respected within the anti-spam community and many consider it indispensable for filtering out spam. Indeed, without the IP block lists maintained by *Spamhaus* (combined with its reliably low false positive rate), many filtering services would be unable to function as the filtering resources would be overwhelmed with more expensive content filtering that would drag down network resources and introduce unacceptably high latency.

Spamhaus expanded on its bad guy identification with a domain block list in March 2010, but then in September it moved forward with the inverse of what it is known for – a whitelist of known good senders. While still a small list, this is a reversal of what the anti-abuse community has typically done until now (identify the bad players).

Moving to a whitelist model is not exactly revolutionary, but it does put an interesting twist on spam filtering. If we start looking for the good guys, can we be more aggressive against the bad guys? Perhaps we can use it to be more efficient in spam filtering by conserving resources on content filtering. Or perhaps we can use it to drive down false positives. Or perhaps this is something that we will have to do anyhow if we ever start to use IPv6 addresses to send email, as this will make the use of IP block lists much more difficult. In any case, the shift from looking for bad guys to looking for good guys is an intriguing development.

3. AUSTRALIA BOOTS INFECTED PCS OFF ITS NETWORKS; COMCAST STOPS JUST SHORT OF DOING THAT

For years, many home computer users have been oblivious to the malware running on their systems and to just how much abuse they have been responsible for. As these infected machines act as bot nodes, the inattentiveness of their owners causes real harm to the rest of the world. In January, the federal government of Australia urged ISPs to come up with a mechanism to take infected computers offline and submitted its own draft copy of a voluntary code of conduct. The abuse mitigation plan proposed slowing down an infected customer's Internet connection or changing their password so that the user would be forced to

call the support desk (at which point they could be informed that their machine was infected with malware and advised as to how to clean up and prevent future infections).

Later on in the year, American ISP *Comcast* experimented with its own bot-detection programme before rolling it out to its entire customer base. *Comcast's* approach is to detect when a customer's computer is connecting to a known botnet. Then, when the computer next connects to the Internet, *Comcast* directs the user's homepage to a landing portal – a quarantine of sorts. This page informs the user that they have an infected machine, and provides information about why they were quarantined and what to do about it.

The Australian government and *Comcast* are being proactive and attempting to change user behaviour by making it inconvenient for users to have bad computer hygiene. Most computer users are not aware of the malware lurking in the background on their machines, but if they receive a notification saying that they must take action or else their Internet experience will be impeded, then that prompts them to change their behaviour. At the very least, it should prompt them to install the latest updates and run a malware check.

It's a good strategy and a nice change – prompting the user to take action and doing the heavy lifting for them (detecting infection) instead of assuming that they will engage in best practices of taking care of their computer.

4. GAWKER HACKED, TWITTER SPAMMED, LINKEDIN FORCES USER RESETS

It is not unusual these days for websites to be hacked. Indeed, many of the vulnerabilities that were in play 10 years ago are still in play today (such as SQL injection attacks and cross-site scripting). 2010 was no stranger to these attacks.

In the summer of 2010, the social media website *Gawker* was attacked, and in December the hackers made their findings public. The hackers managed to acquire nearly 1.5 million usernames and decrypted about 200,000 of them. The most commonly used passwords were 'password', '123456' (or some variant thereof) and variants of 'qwerty', the first six letters on a standard keyboard.

The attack on *Gawker* is one of a long list of compromises. In 2009, hackers posted hundreds of thousands of *Hotmail*, *Yahoo!* and *Gmail* usernames and passwords. Then, as now, the most common passwords were 'password', '123456' and other such variants. User behaviour hasn't changed much in a year, and statistical guessing games are all an attacker needs in order to compromise an account. If password reuse is such a common occurrence among users, then cyber thieves don't need many skills in order to break into a system. All they need is a list of usernames (usually

email addresses) and then, starting with the most commonly used passwords, can use a process of trial-and-error until they find one key that works.

What made this particular attack so heinous was not its size per se, but rather its reuse and downstream implications. After the hacking group posted the usernames and passwords, a flurry of spam activity started to emanate from *Twitter* where some of the usernames were sending out spammy tweets. Some users of *Gawker* must have been using the same credentials to log into their *Twitter* accounts. Even if the passwords were not decrypted, it was a fairly good bet that they contained some of the most common ones in the set. In other words, the hackers gave the spammers a gift by publishing the *Gawker* credentials, and other services like *Twitter* paid for it.

LinkedIn, another social networking site, decided to take evasive action and force its entire user base to reset their passwords as there were likely to be a number of reused usernames and passwords in *LinkedIn's* user list, too.

The lessons we've learned from this particular hack are:

- a) Users do not choose good passwords.
- b) Other sites will pay the penalty for such insecurity.
- c) Hacking incidents like these are not going to go away anytime soon.

5. SHUTDOWNS ABOUND

2010 saw botnets infiltrated and shut down in droves as security researchers resorted to a variety of tactics in order to knock them offline.

- In January, the Lethic botnet was shut down by *Neustar*.
- In February, *Microsoft* acquired a court order which effectively shut down *Waledac*.
- In March, a group of security researchers worked together with law enforcement and shut down the *Mariposa* botnet.
- In August, security company *Lastline* took down the *Cutwail* botnet.
- In September, authorities arrested a large money mule operation associated with the *Zeus* botnet.
- In November, the *Bredolab* botnet was taken down by anonymous security folks.
- In late December, *Rustock* stopped sending so much spam.

Each of these takedowns had a small effect in the immediate aftermath. However, as we have learned since the 2008 *McColo* shutdown, botnet operators have become

increasingly resistant to major disturbances and it no longer takes very long for them to get back online. Indeed, what we see now compared to back then is that botnet operators have evolved to make their infrastructure smarter. They have also started getting smaller; rather than huge botnets consisting of lots of nodes, they have lots of botnets with smaller numbers of nodes. This means that shutting down a botnet now has less impact on global spam levels.

6. MORE ARRESTS OF CYBERCRIMINALS

Every year there are stories of cybercriminals being arrested or being hit with huge fines. Every year, anti-abuse professionals hope that this will reduce the amount of abuse going on, and every year they are disappointed. However, hope springs eternal that maybe this time it's different. Below are some of the more notable cases of 2010:

- In June, *Microsoft* sued Boris Mizhen, accusing him of sending millions of spam messages to *Hotmail*. The lawsuit also claimed that Mizhen attempted to circumvent its filters by abusing the Junk Email Reporting Program and its Smart Network Data Services system. Mizhen has a long history of being accused of spamming; he has previously been listed on *Spamhaus's* Register of Known Spam Operators (ROKSO) and had previously been sued by *Microsoft* (in 2003) for sending spam to *Hotmail*.
- In July, a 23-year-old Slovenian was arrested by authorities on suspicion of helping to develop the Mariposa botnet.
- In August, notorious alleged spammer Leo Kuvayev was arrested in Russia. Kuvayev had previously been charged with spamming by the state of Massachusetts and at one point was listed by *Spamhaus* as one of the world's three most prolific spammers. This time, however, Kuvayev was arrested and jailed on multiple charges relating to child abuse³.
- In October, authorities in the UK arrested 19 people in connection with the ZeuS malware gang. The group of individuals behind the spyware ring was thought to be part of a multinational operation that was responsible for stealing \$10 million over a three-month period and may have been responsible for up to \$30 million. What was significant about this arrest was the amount of international cooperation that went on to break up this ring. ZeuS is one of the major pieces of malware out there and it is hoped that going after the people behind it will knock part of it offline, at least for a little while.

³ <http://krebsonsecurity.com/2010/08/spam-king-leo-kuvayev-jailed-on-child-sex-charges/>.

- In November, 23-year-old Oleg Nikolaenko, believed to be the operator of the Mega-D botnet, one of the largest spamming botnets in the world, was arrested and scheduled to be arraigned in a federal court in Milwaukee, Wisconsin.

Defence in depth is always important in computer security, but disruption of the activities of those behind the threats also has the potential to impact cyber abuse in a positive way.

7. SPAM VOLUME DROPS

For years, users of the Internet have been plagued mercilessly with junk mail filling up our inboxes. This has spawned a new industry – the anti-spam industry. Since the start of the anti-spam industry (less than 15 years ago), we have seen spam levels continuing to rise every year. Indeed, according to some metrics, spam comprises 90% of all email, and 97% of email flowing over the public Internet.

Yet, in the second half of 2010, spam volumes started to decline. This was noticed by a number of vendors. The CBL – an IP block list that populates its lists with widely distributed honeypots, and a contributor to the *Spamhaus* feed – saw a steady decline starting in May 2010:

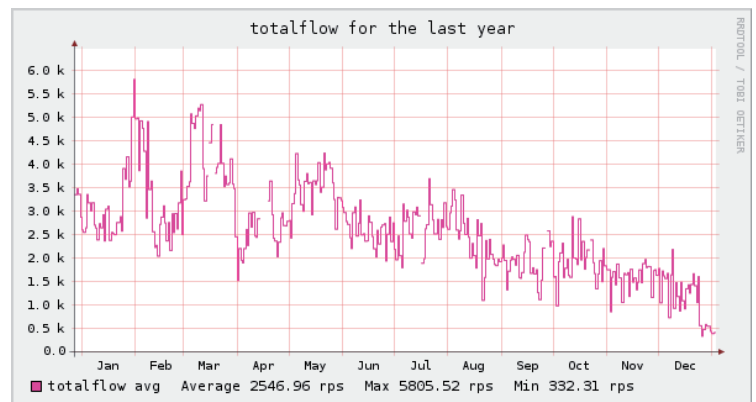


Figure 1: CBL monthly spam volume hits⁴.

Over at *Microsoft Forefront Online*, a similar trend was observed. After ramping up during the start of the year, a gradual decline was seen in the volume of spam hitting the servers, and total spam volume continued to drop for the rest of the year (Figure 2).

McAfee's third quarter threat report⁵ showed the same pattern, as did reports from *Cisco's Senderbase* page.

⁴ <http://cbl.abuseat.org/totalflow.html>.

⁵ <https://prod.secureforms.mcafee.com/content/verify?docID=3B02BC3A-1283-4782-AB7E-2FA2D23B7031&CID=WB193&src=web&aType=report&locale=us>.

Microsoft Forefront Online - Total Weekly Spam

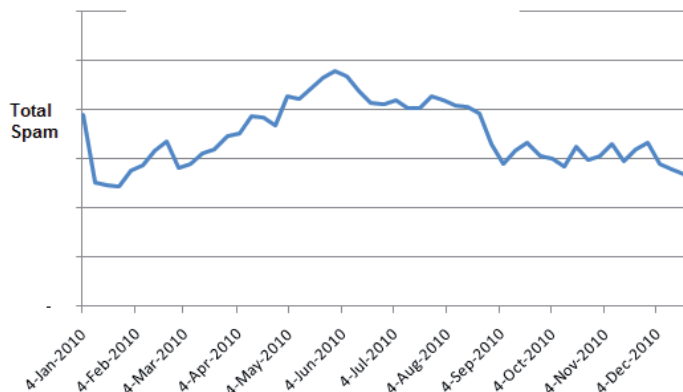


Figure 2: Microsoft Forefront Online weekly spam volume in 2010.

This is a very interesting development. There are numerous possible explanations for the drop in spam:

- In September, a spam affiliate programme known as Spamit decided to close down, blaming its closure on increased public attention⁶. Affiliates like this are commonly associated with the Canadian Pharmacy spam advertisements that we regularly see, not just in our email but also in our search engine query results and the comment sections of various blogs. With Spamit out of the way, perhaps pharmaceutical spammers lost a major source of revenue. Without the money, there is no incentive to spam.
- In October, two rival sets of malware developers – those behind Zeus and SpyEye – decided to merge code bases⁷. It takes time to combine two pieces of software and if you are not an organization that is used to having to support older versions, those older versions of the software can simply lie ‘stranded’ (i.e. nobody pays attention to them and they are not supported). Perhaps the technicality of merging the two code bases is not going smoothly in the criminal underground.
- Perhaps the shut downs of all of the various botnets *are* having an effect on the amount of spam that we receive. Spammers, tired of being shut down, are moving on to other things because once you get good at spamming, authorities start to take notice. It may be that the risk/reward ratio is not what it once was.

⁶ <http://krebsonsecurity.com/2010/09/spam-affiliate-program-spamit-com-to-close/>.

⁷ <http://krebsonsecurity.com/2010/10/spyeye-v-zeus-rivalry-ends-in-quiet-merger/>.

Will this particular trend seen in 2010 continue into 2011? That remains to be seen. It could be that more lucrative types of abuse are starting to attract the major criminal players – and that leads into the next major story of 2010.

8. FACEBOOK UNVEILS ‘THIS IS NOT EMAIL’

In November 2010, social networking site *Facebook* announced a new messaging platform using the *Facebook* interface – a new communication platform that works a lot like email.

This messaging platform is basically a way to talk to people from within *Facebook* regardless of whether they are users of the site. You use the *Facebook* interface to talk to your friends. If your friends are on SMS (i.e. text via cell phone), the communication exchange will be sent to the *Facebook* chat. If your friend is using a chat platform, then the message is routed through the *Facebook* chat window into their chat program. Similarly, if all they use is email, then you can have a conversation with them. You are given a you@facebook.com email address. You send your friend the message, and they reply to you. The message then goes to your *Facebook* messages view where you can read it. In this case, the email address you are given is an SMTP address to which the mail is delivered, and then the *Facebook* messaging API parses the message and renders it for you in your own window.

Facebook was pretty insistent that this platform is not email, nor is it meant to directly replace email. The model of this communication platform is more like chat – no subject lines, and instant communication, all while allowing the user to use one platform. It didn’t take long for others to predict that this would be the first nail in the coffin for email (my own position is that email will always be a useful platform because you need more than just chat to communicate electronically). However, the point was made – email is not quite the growth platform it was 15 years ago. The major growth is in other avenues of communication, especially through social networking, and that is where advertisers will innovate.

Folks in the technology business may not realize it, but if teens and advertisers start flocking to platforms other than email, then spammers will do so as well. Indeed, there are plenty of ways for spammers to advertise – through fake tweets on *Twitter*, through black search engine optimization, through compromised social networking accounts and friendship requests, and so on. These are all major problems that newly successful companies of the past five years have had to deal with. Once you become popular, you become a magnet for abuse. Unfortunately, many such

companies haven't quite learned how to deal with abuse yet. Email spam has been around for a long time, and both email providers and (most) users have a good understanding of how to combat it. Perhaps the reason spam dropped in 2010 is because spammers are seeing other, more profitable ways to abuse the Internet where the defences are not yet well established.

Facebook did not unveil a platform that does away with the need for email. However, it may have signalled to the rest of us that the focus for abuse is moving someplace else⁸.

9. HACKTIVISM IS HERE TO STAY

He may not have won *Time Magazine's* Person of the Year award, but *WikiLeaks* founder Julian Assange made a major impact on global diplomacy. With the release of thousands of documents about the wars in Iraq and Afghanistan, people started to take notice. However, with the release of thousands of diplomatic cables in November, and the ensuing heated public debate over the legality of those disclosures, corporations that were passively involved in providing financial services to *WikiLeaks* started to distance themselves from the site. By the end of the year, *PayPal*, *MasterCard* and *Visa* were reported to have stopped processing *WikiLeaks* donations and transactions.

Driven by an ideology of hacktivism, which uses hacking as a means to advance political goals, and informationism, which is the belief that information should be allowed to flow freely throughout the Internet, volunteers belonging to a group known as 'Anonymous' launched distributed denial-of-service attacks against the websites of these companies. Believing that these companies were passively involved in a conspiracy to suppress free access to information (or at least conspiring to act as a barrier to it), Anonymous succeeded in taking down the targeted firms for a period of time with an all-volunteer DDoS attack.

PayPal, *MasterCard* and *Visa* are not small websites. They handle a lot of traffic every day in order to deal with their very large user bases. For these to be overwhelmed with a DDoS attack means that Anonymous had to mobilize a lot of resources. They did so in a short period of time, primarily tapping a large and vocal group of readers of a particular website.

Ideologically driven individuals with technical skills, such as those who constructed the DDoS tools taken up by Anonymous – or even those without technical skills of their

⁸ I don't mean that spam will ever go away – it won't. There's too much money in it. What I mean is that the problem will not get exponentially worse the way it did in the late 1990s and first half of the 2000s.

own but with access to simple tools of Internet disruption – can do a lot of damage when motivated to do so, as the *WikiLeaks* backlash attests.

10. STUXNET APPEARS

No story generated more hype, mystery and intrigue in 2010 than that of Stuxnet, and deservedly so. In July 2010, a new strain of malware was detected that utilized four zero-day vulnerabilities in *Windows*. It also exploited a vulnerability used in 2009's major Conficker outbreak. Stuxnet was a computer worm that was designed to cause major disruptions in the circuit boards of industrial control systems, reprogram programmable logic controller boards, and then hide its changes. A certain number of infections were discovered in Iran, and a large proportion of those were discovered in Iranian nuclear power plants. In November, Iran confirmed that its nuclear programme had been damaged by the worm. Stuxnet appears to have been intended to cause damage to various components of nuclear plants – making almost imperceptible changes that could lead to great cumulative disarray.

Stuxnet was not like a traditional piece of malware that sneaks into a system. From the early discovery that it had been digitally signed with certificates from legitimate tech companies to subsequent speculation and revelations concerning its origin, purpose, architects and effect, theories have abounded. Those stories have been told in other venues so I won't repeat them here⁹, but already rumours of copycat versions (or next-generation versions, or rumours of the vulnerability being sold on the black market) are spreading – even though the four zero-day vulnerabilities have been patched for months.

Stuxnet is a game changer because it is believed to be a remarkably sophisticated instance of cyber espionage. Attacks in 2007 against Estonia and in 2008 against Georgia took the pattern of a cyber riot wherein nationalistic hooligans with technical skills took down a country's infrastructure using electronic attacks. Stuxnet, on the other hand, hints at a remarkably well-planned and well-organized effort involving multiple knowledge bases and skills.

CONCLUSION

Well, that's the way I saw the world this year. There were lots of other stories that I could have mentioned, but these ten were the most memorable, and the most pivotal. I look forward to seeing what stories unravel in 2011.

⁹ Some of them require the use of a tinfoil hat, and deservedly so.

COMPARATIVE REVIEW

VB100 COMPARATIVE REVIEW ON UBUNTU LINUX 10.04 LTS

John Hawes

2010 saw some setting – and breaking – of records in the VB test lab, with several tests proving to be of truly epic proportions. 2011 promises no let up in the steady onslaught of new solutions participating in our tests, and in the coming months we hope to see more of the diversity and innovation spotted in some of 2010's products. We also hope to see less of the fragility, instability, lack of clarity and general bad design we saw in many others.

For now, however, we leave the cluttered *Windows* space behind to make our annual probe into the murkier, less cuddly but generally more robust world of *Linux*. In a market space that is much less crowded with small-time niche players, our *Linux* tests tend to be less over-subscribed than many others, and generally feature only the most committed, comprehensive security providers (most of whom make up our hardest core of regular entrants). For only the second time, we decided to run the test on the explosively popular *Ubuntu* distribution, which was first seen on the VB100 test bench almost three years ago (see *VB*, June 2008, p.16).

Despite being later than usual, the product submission deadline of 5 January caused problems for developers in some regions thanks to varying holiday times. For at least a couple of major vendors there was no submission this month, either due to lack of support for the platform chosen or due to a lack of resources to prepare a submission so close to the New Year. Other vendors chose to skip this month's test for other reasons. Nevertheless, a strong field of 14 entrants arrived on deadline day, covering the bulk of our regulars.

PLATFORM AND TEST SETS

Having last looked at *Ubuntu* version 8.04 a few years ago, we expected a few improvements in the current Long Term Support version 10.04, released in mid-2010. However, the installation process showed little sign of such improvement, with a fairly rudimentary command-line-driven set-up system, which nevertheless did the job adequately. The most fiddly part was the software selection system, which seemed far from intuitive, but fortunately we required little beyond the basics of a fileserver, intending to add in any additional dependencies on a per-product basis. Once the vagaries of the interface had been conquered, the actual work of installing was rapid and relatively undemanding, and with the system up and running, standard controls

enabled implementation of all the settings we required in short order.

As in the previous test on this platform, the installer provided no graphical desktop by default, which seems a sensible approach for a server platform; graphical interfaces are generally unnecessary in the day-to-day running of services, and can be both a performance drain and a security risk. It seems likely that many if not most machines running the platform under test would operate like this, and indeed even in a setting as small as the VB test lab we run a number of *Linux* machines with no windowing system, including some with older versions of *Ubuntu*. Nevertheless, some of the vendors taking part indicated that their solutions were geared towards graphical operation, so we had to hope that traditional command-line methods would also be supported.

The main issue we expected to see was with on-access scanning, which is always slightly fiddly on *Linux*. In the past there have been three main approaches: protecting *Samba* shares only, using *Samba* vfs objects, which usually entails little more than an added line or two in the *Samba* configuration file; the open-source *dazuko* system, which allows more granular control of protection over different areas of the system; and proprietary methods, which can vary greatly from provider to provider. *Dazuko* has been somewhat awkward to set up in the past, involving compilation from sources and often with special flags required depending on the platform, but in some quick trials this month there were no problems in getting it up and running. A new and improved version, *dazukofs*, is also available and looked likely to be used by some products.

Building this month's test sets proved something of a challenge however, after some problems with hardware, software and the human factor set things back several weeks. The imposed delays allowed time to integrate several new malware feeds into our collection processes, which added considerably to the number of samples included in the raw sets. With time pressing, test sets were built with minimal initial filtering – the verification and classification process continued while the tests were run. These issues having eaten heavily into our already shortened month, and well aware that the large test set sizes would mean longer test times for all, we were in some hurry to get things moving.

Fortunately, little work was required in building the core certification sets. The latest WildList available on the deadline date (the November list, released in late December) featured mainly standard worms and online gaming password-stealers, and none of the major new file infectors of the sort that have been causing problems

On-demand tests	WildList		Worms & bots		Polymorphic viruses		Trojans		Clean sets	
	Missed	%	Missed	%	Missed	%	Missed	%	FP	Susp.
Avast	0	100.00%	703	98.74%	10	99.42%	4339	97.44%	0	0
AVG	0	100.00%	1745	96.87%	50	97.93%	5788	96.58%	0	0
Avira	0	100.00%	102	99.82%	0	100.00%	733	99.57%	0	0
BitDefender	0	100.00%	120	99.78%	0	100.00%	1660	99.02%	0	0
Central Command	0	100.00%	2401	95.69%	187	90.52%	28019	83.45%	0	1
eScan	0	100.00%	2184	96.08%	0	100.00%	3817	97.75%	0	0
ESET	0	100.00%	768	98.62%	0	100.00%	9848	94.18%	0	4
Frisk	0	100.00%	6744	87.89%	0	100.00%	33450	80.24%	0	0
Kaspersky AV	0	100.00%	6035	89.17%	0	100.00%	11804	93.03%	1	0
Kaspersky ES	0	100.00%	6035	89.17%	0	100.00%	8002	95.27%	1	0
Norman	0	100.00%	7452	86.62%	281	85.29%	31725	81.26%	0	1
Quick Heal	0	100.00%	1748	96.86%	42	96.94%	7505	95.57%	0	0
Sophos	0	100.00%	1810	96.75%	0	100.00%	10213	93.97%	0	0
VirusBuster	0	100.00%	2401	95.69%	187	90.52%	28019	83.45%	0	1

(Please refer to text for full product names)

for many products of late. As replication of these in large numbers takes considerably longer than verification of less complex items, the set was compiled quickly. With several older file infectors removed from the list it shrank to a little over 5,000 samples, the bulk of which were from two remaining file infectors, a single strain of W32/Virut and the venerable W32/Polip.

The clean set was expanded with the usual batch of new items, focusing mainly on business-related tools this month to reflect the typical user base of the platform. The speed sets were augmented as usual by a selection of *Linux* files, this time taken from the core directories of one of our standard server systems. Some doctoring of our test automation processes was required to fit with the different platform – the on-access tests and performance measures were all run from a *Windows XP Professional SP3* client system, to emulate normal usage for a *Samba* file server protecting a network of *Windows* machines. To ensure fairness, speed tests were run one at a time with other network activity kept to a minimum.

With everything set up, all that remained was to get to grips with the solutions themselves. From past experience, we expected to see some nice, simple designs in between more challenging approaches, with ease of use depending greatly on the clarity of documentation as well as use of standard *Linux* practice.

Avast Software avast! for Linux 3.2.1

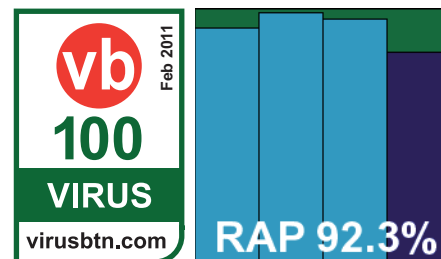
Version VPS 110105-1

ItW	100.00%	Polymorphic	99.42%
ItW (o/a)	100.00%	Trojans	97.44%
Worms & bots	98.74%	False positives	0

Avast started things off nicely, with a compact 37MB install bundle in tar.gz format, containing three .DEB packages.

Instructions were short and simple, running through the steps of installing the .DEBs, making a few tweaks to the system and getting the *dazuko* modules compiled and installed. With concise and comprehensive advice, it took only a minute or two to get everything set up just as we wanted.

With ample man pages and all the required executables easy to find, setting up and automating the full test suite was a simple process, and running through it was as rapid



On-access tests	WildList		Worms & bots		Polymorphic viruses		Trojans	
	Missed	%	Missed	%	Missed	%	Missed	%
Avast	0	100.00%	661	98.81%	10	99.42%	3614	97.87%
AVG	0	100.00%	1641	97.05%	50	97.93%	6415	96.21%
Avira	0	100.00%	102	99.82%	0	100.00%	731	99.57%
BitDefender	0	100.00%	120	99.78%	0	100.00%	1660	99.02%
Central Command	0	100.00%	3082	94.47%	187	90.52%	29346	82.67%
eScan	1	99.85%	121	99.78%	0	100.00%	1680	99.01%
ESET	0	100.00%	625	98.88%	0	100.00%	9692	94.28%
Frisk	0	100.00%	6742	87.90%	0	100.00%	33436	80.25%
Kaspersky AV	0	100.00%	6101	89.05%	0	100.00%	13029	92.30%
Kaspersky ES	0	100.00%	6101	89.05%	0	100.00%	9232	94.55%
Norman	0	100.00%	7872	85.87%	324	84.49%	37191	78.03%
Quick Heal	0	100.00%	7033	87.37%	42	96.94%	22996	86.42%
Sophos	0	100.00%	1810	96.75%	0	100.00%	10211	93.97%
VirusBuster	0	100.00%	3082	94.47%	187	90.52%	29346	82.67%

(Please refer to text for full product names)

as usual. Scanning speeds on demand were pretty decent, with on-access overheads perhaps somewhat higher than expected, but still pretty decent.

On checking the results we found solid scores in all sets, as expected, but in the RAP sets there was a bit of a surprise in that scores for the last few weeks were completely absent. We re-ran the tests keeping a close eye on the console output, and quickly diagnosed that the scanner had crashed on a malformed sample in the extended set, with a segmentation fault error. The test was repeated, skipping the offending section of the set. No further problems emerged, and even with this doubling of effort the hugely impressive speed of scanning infected files meant that all tests were completed in well under 24 hours. The core sets were handled effortlessly, and Avast notches up another successful VB100 pass.

AVG 8.5.863

Virus database version 271.1.1/3356; Scanner version 8.5.850

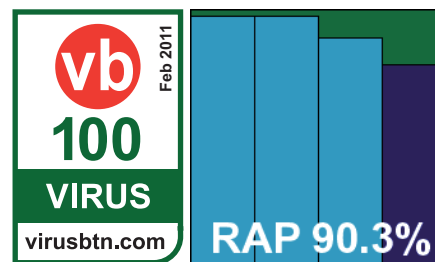
ItW	100.00%	Polymorphic	97.93%
ItW (o/a)	100.00%	Trojans	96.58%
Worms & bots	96.87%	False positives	0

AVG's Linux solution is a little more bulky, arriving as a 93MB .DEB package along with a licence key to activate

it. The set-up process was simple enough to start with, but once the package was installed considerably more work was required to

decrypt a rather fiddly configuration system. This involved passing configuration changes into the product as long and easily mistyped strings, rather than making changes to human-readable and self-explanatory configuration files, as is generally the case for Linux software. The layout, with multiple binaries with overlapping and bewildering names and functions, was also less than helpful, and man pages proved pretty eye-watering too, but in the end we got things working just well enough to get through the tests. The product has options to provide on-access protection through several methods, but we opted for the *dazuko* approach as the most simple to operate.

Once the configuration had been adjusted to our needs and the syntax of the scanner tool figured out, running the tests was much less of a headache. Speeds and overheads were good, and detection rates splendid, and with no issues in the core sets, AVG comfortably earns a VB100 award.

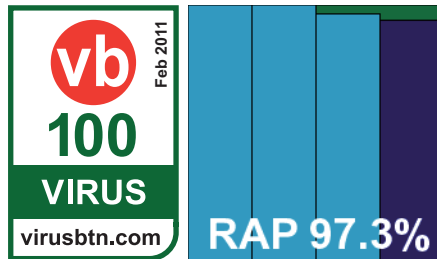


Avira AntiVir Server 3.1.3.4

SAVAPI-Version 3.1.1.8; AVE-Version 8.2.4.136;
VDF-Version 7.11.1.20 created 20110104

ItW	100.00%	Polymorphic	100.00%
ItW (o/a)	100.00%	Trojans	99.57%
Worms & bots	99.82%	False positives	0

Avira's Linux server solution was provided as a 55MB tar.gz archive bundle, along with an extra 37MB of updates. Inside the main



bundle was a folder structure containing an install script, which ran through the set-up process clearly and simply, including compilation and insertion of *dazuko*. Some additional options included a GUI for the *Gnome* desktop and a centralized management system, and the installation even informs you where the main control binaries are located, to avoid the scrabbling around often experienced with less helpful products. Despite its clarity and simplicity, the set-up still ends by urging the user to read the product manual for more detailed information.

After this exemplary install, using the product proved similarly unfussy and user-friendly, adhering to standard *Linux* practices and thus making all the required controls both easy to locate and simple to operate. Documentation was also clear and comprehensive. Speeds were super-fast and super-light, and detection rates were as excellent as ever. With no problems in the core sets, *Avira* easily earns another VB100 award.

BitDefender Security for Samba File Servers 3.1.2

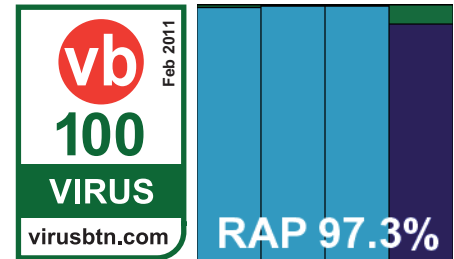
ItW	100.00%	Polymorphic	100.00%
ItW (o/a)	100.00%	Trojans	99.02%
Worms & bots	99.78%	False positives	0

BitDefender's product was a little different from most, with its 100MB submission provided as a .RUN file. When run as the filename suggested, this installed the packages and set things up as required. Part of the set-up involved compiling components (the *Samba* vfs object code required for the on-access component), and several other dependencies also had to be met prior to installation, but it was not too much effort and completed in reasonable time. Once again,

configuration was geared towards complexity rather than user-friendliness, with lengthy and fiddly commands

required to bring about any change in settings, but it wasn't too horrible once the esoteric formulae for generating adjustments had been worked out.

Running through the tests proved smooth and stable, although both on-demand speeds and on-access overheads were somewhat heavier than might be expected, but detection rates were impeccable and the core sets were stomped through without a problem, earning *BitDefender* a VB100 award.



Central Command Vexira Professional 6.3.14

Virus database version: 13.6.130.0

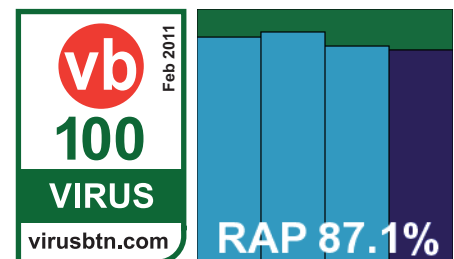
ItW	100.00%	Polymorphic	90.52%
ItW (o/a)	100.00%	Trojans	83.45%
Worms & bots	95.69%	False positives	0

Central Command has recently become a fixture in our comparatives, with a run of successes under its belt.

This month the

product was presented as a pair of .tgz archive files, the main product measuring 57MB and the additional update bundle 65MB. Unpacking the main bundle revealed a handful of .DEB files and a Perl install script. This ran through tidily, getting everything set up in good order. Some additional instructions were kindly provided by the developers with details of updating and adjusting settings. A secondary set-up script was also provided to change the settings of the *Samba* configuration, enabling on-access protection.

With everything set up, testing proved a breeze, although configuration of the on-access scanner was somewhat limited – at least as far as could be judged from the sparse documentation. Nevertheless, the default settings did well and it tripped along at a good pace. Scanning speeds were not bad and overheads were light, with the usual fairly



File access lag time (ms/MB)	Archive files			Binaries and system files			Linux files			Media and documents			Other file types		
	Default (cold)	Default (warm)	All files	Default (cold)	Default (warm)	All files	Default (cold)	Default (warm)	All files	Default (cold)	Default (warm)	All files	Default (cold)	Default (warm)	All files
Avast	155.43	155.46	155.43	49.52	49.18	49.52	310.49	308.17	310.49	67.36	65.38	67.36	86.67	83.72	86.67
AVG	12.13	11.75	NA	59.15	55.16	59.15	192.13	187.39	N/A	81.22	80.00	81.22	120.37	119.40	120.37
Avira	167.00	167.24	167.00	16.95	16.84	16.95	321.63	319.64	321.63	43.40	42.53	43.40	50.70	50.33	50.70
BitDefender	151.74	156.44	151.74	76.32	76.48	76.32	472.01	460.62	472.01	250.86	242.79	250.86	347.71	332.40	347.71
Central Command	6.94	6.90	NA	66.29	66.10	66.29	206.52	193.82	N/A	79.61	81.96	79.61	118.44	123.73	118.44
eScan	83.70	85.69	83.70	71.38	72.19	71.38	331.35	332.32	331.35	157.08	156.59	157.08	221.05	219.22	221.05
ESET	147.96	147.52	147.96	19.04	18.53	19.04	254.44	248.98	254.44	70.62	70.01	70.62	64.86	70.98	64.86
Frisk	87.05	87.28	87.05	67.08	67.54	67.08	195.47	181.07	195.47	39.61	38.17	39.61	67.52	63.28	67.52
Kaspersky AV	16.41	16.13	651.69	63.16	62.33	344.87	225.92	223.42	956.09	97.71	96.71	385.47	137.51	135.78	429.85
Kaspersky ES	17.04	16.98	517.11	47.48	48.19	200.70	195.81	192.98	777.46	79.86	77.22	227.50	100.52	96.36	248.56
Norman	9.42	0.00	N/A	93.48	0.00	93.48	413.01	0.00	N/A	303.73	0.00	303.73	402.69	0.00	402.69
Quick Heal	7.59	7.35	NA	61.77	61.24	61.77	409.14	405.03	N/A	273.07	267.20	273.07	224.01	227.45	224.01
Sophos	10.03	9.77	843.06	55.80	55.54	232.25	161.81	150.75	1295.57	46.16	45.54	215.38	101.38	100.11	279.55
VirusBuster	5.12	4.69	NA	49.44	48.79	49.44	195.23	189.73	N/A	55.93	59.70	55.93	88.68	87.09	88.68

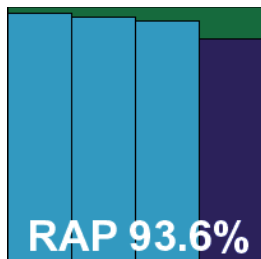
(Please refer to text for full product names)

decent level of detections. With no problems in the clean set or WildList set *Central Command* earns another VB100 award for its growing collection.

eScan for Linux File Servers 5.0-2

ItW 100.00% **Polymorphic** 100.00%
ItW (o/a) 99.85% **Trojans** 97.75%
Worms & bots 96.08% **False positives** 0

The *Linux* version of *eScan* comes as a handful of .DEB packages, installation of which required resolving a few dependencies, including for one package several components of the X desktop system – clearly this was one of those products leaning towards graphical rather than command-line usage. This was not a problem, as despite there being no evidence of configuration for some aspects (notably the on-access protection) at the local console level, it was easily accessible through a browser-based web administration tool. Checking



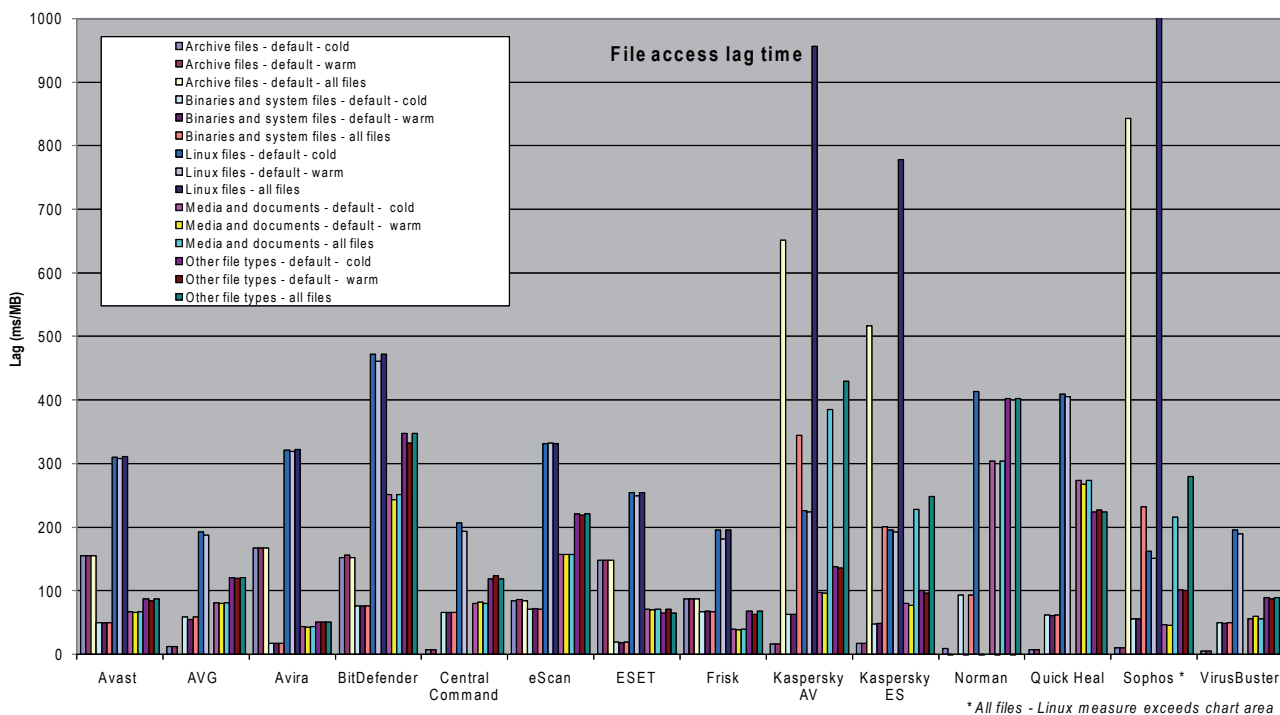
this out from another machine, we found it fairly clear, but in places a little prone to flakiness – resetting our changes on a number of occasions as soon as ‘apply’ was clicked. Local console documentation also seemed a little sparse, but we soon figured things out and got the test moving along.

Speeds held up well against the rest of the field, and detection was solid. All looked to be going swimmingly until a single item went undetected in the WildList set on access – with a default setting to ignore files larger than 13MB (a reasonably sensible level), *eScan* was extremely unlucky in that this month’s WildList contained a larger sample than this (25MB). This bad luck denies *eScan* a VB100 award this month, despite a generally decent performance.

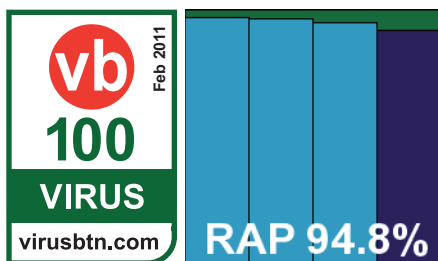
ESET File Security 3.0.20

ItW 100.00% **Polymorphic** 100.00%
ItW (o/a) 100.00% **Trojans** 94.18%
Worms & bots 98.62% **False positives** 0

ESET’s Linux edition was provided as a single 41MB .DEB package, and installed easily with minimal fuss. Clear instructions showed how to set up protection of *Samba* shares using a vfs object (*dazuko*-style protection was also



available), and the commands and configuration were properly laid out, conforming to expected norms.



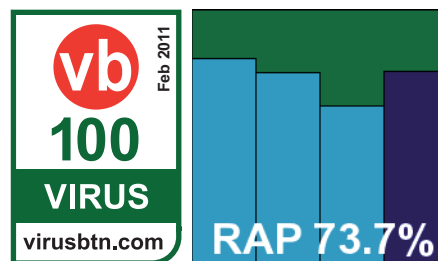
Running the test was fairly painless, although a couple of files in our extended sample sets did cause segmentation faults and required the restarting of scans. Speeds were pretty zippy and overheads nice and light, particularly in the binaries section. Detection rates were solid across the sets. The clean set threw up a few warnings of potentially unwanted items (most identified precisely and accurately) and a couple of packer warnings, but nothing could stop *ESET*'s inexorable progress towards yet another VB100 award.

Frisk F-PROT Antivirus for Linux File Servers 6.3.3.5015

Engine version: 4.5.1.85; virus signatures 201101040744 6e8837db11f3f34f0bfe050aa91a01a9

ItW	100.00%	Polymorphic	100.00%
ItW (o/a)	100.00%	Trojans	80.24%
Worms & bots	87.89%	False positives	0

Frisk's Linux product came as a 24MB .tgz archive, with an accompanying 26MB of updates and a small patch file.



Installation was basic and rudimentary, with a little install script creating symlinks to the main components without moving them from where they had originally been unpacked – a nice, unobtrusive approach as long as it is expected.

Getting things up and running proved a breeze, with both *dazuko* and *Samba* vfs objects supported (*dazuko* was used for all our on-access tests), and configuration and operation were made easy thanks to the product's conformance with the expected behaviour for *Linux* solutions.

With good scanning speeds and no stability problems, tests were completed in excellent time. Detection scores were decent, with a dip in the later parts of the RAP sets but a strong resurgence in the proactive week. No problems were noted in either of the core certification sets and a VB100 award is duly earned by *Frisk*.

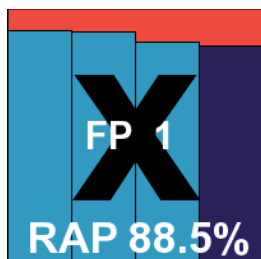
On-demand throughput (MB/s)	Archive files			Binaries and system files			Linux files			Media and documents			Other file types		
	Default (cold)	Default (warm)	All files	Default (cold)	Default (warm)	All files	Default (cold)	Default (warm)	All files	Default (cold)	Default (warm)	All files	Default (cold)	Default (warm)	All files
Avast	6.19	6.28	6.18	18.21	18.45	16.25	3.76	4.08	3.17	8.99	9.21	10.07	6.48	6.56	7.07
AVG	15.38	15.46	1.47	28.07	25.66	16.08	3.38	3.49	1.13	21.63	21.66	8.97	14.14	14.24	6.34
Avira	6.25	6.26	6.25	55.83	57.28	55.83	2.80	2.99	2.80	22.25	23.34	22.25	19.87	20.81	19.87
BitDefender	6.50	6.49	6.50	16.34	16.64	16.34	2.59	2.65	2.59	6.68	6.91	6.68	4.79	4.99	4.79
Central Command	7.20	7.21	4.39	15.13	15.39	15.43	7.05	6.62	2.04	6.32	6.34	5.76	4.75	4.81	4.57
eScan	6.14	6.16	6.14	17.91	18.11	17.91	3.40	3.40	3.40	14.46	14.57	14.46	10.25	10.02	10.25
ESET	6.55	6.62	6.55	23.30	23.68	23.30	3.69	4.03	3.69	11.62	11.40	11.62	9.26	9.41	9.26
Frisk	9.94	9.89	9.77	14.56	14.88	14.49	4.47	4.88	4.68	23.03	27.64	25.17	12.51	15.03	13.24
Kaspersky AV	2.65	2.71	2.65	21.90	21.99	21.90	1.52	1.54	1.52	12.75	13.07	12.75	10.17	10.50	10.17
Kaspersky ES	2.69	2.70	2.69	20.04	20.11	20.04	1.59	1.60	1.59	12.86	13.21	12.86	10.42	10.61	10.42
Norman	1.19	1.18	1.19	4.08	4.00	4.08	1.37	1.36	1.37	4.88	4.85	4.88	3.10	3.07	3.10
Quick Heal	2.56	2.55	2.56	24.57	24.63	24.57	1.50	1.50	1.50	6.78	6.81	6.78	8.04	8.07	8.04
Sophos	86.83	93.77	1.11	14.76	16.76	10.88	19.81	29.77	0.71	18.06	19.08	13.88	9.36	10.11	7.34
VirusBuster	7.24	7.30	4.45	19.68	19.78	19.75	6.62	6.45	2.01	6.25	6.31	5.77	4.80	4.83	4.59

(Please refer to text for full product names)

Kaspersky Anti-Virus for Linux File Servers 8.0.0.136

ItW	100.00%	Polymorphic	100.00%
ItW (o/a)	100.00%	Trojans	93.03%
Worms & bots	89.17%	False positives	1

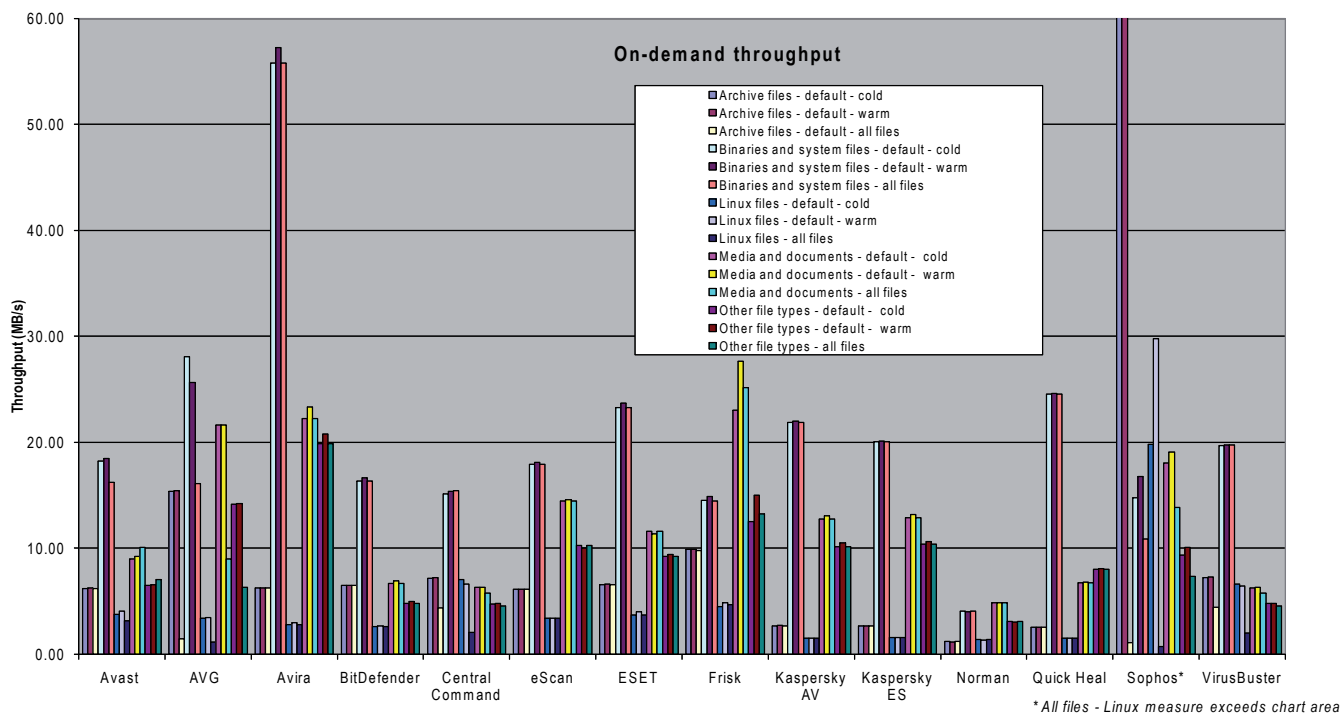
Despite the fact that our test deadline clashed with the Russian Christmas holidays, Kaspersky managed to submit two products this month – both from a new and heavily re-engineered Linux range, and with the slightly worrying assertion that the developers had intended them to be operated via a GUI. Installing the first – which seemed to be slightly more business-focused – proved fairly simple at the outset, with a handful of installer packages provided in different formats and a readme file for instructions. Sadly this proved not to be displayable, let alone legible, and after initially



running through the set-up steps of the .DEB package and finding more help was needed, we resorted to consulting the PDF documentation provided on the company’s website.

This showed a horrendously complex layout for operating the product from the command line, which was eventually mastered and rendered reasonably usable with some practice and much perusal of the 215-page manual, but left us hankering for some nice simple, readable configuration files. We tried some work using a web admin GUI, but found this equally fiddly, clumsy and unresponsive. Logging was also a major problem, with detection events dumped from logs after they reached a certain size – despite having set limits in the product’s controls to a considerably higher level than they ever reached.

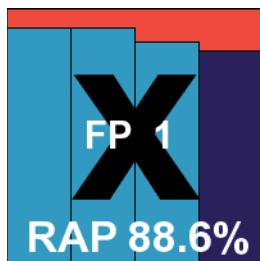
Eventually we got things moving along though, and scanning speeds proved to be very good, with excellent overheads on access with the default settings; turning the settings up to include archive formats and the like added to the overheads considerably, of course. Detection scores were very good, with no problems in the WildList set, but a single false positive in the clean sets was enough to deny Kaspersky’s business product a VB100 award.



Kaspersky Endpoint Security for Linux Workstations 8.0.0.24

ItW	100.00%	Polymorphic	100.00%
ItW (o/a)	100.00%	Trojans	95.27%
Worms & bots	89.17%	False positives	1

The second *Kaspersky* product seemed just about the same as the first, only with different names for some components and no sign of the web admin tool. Once again, we had to consult the manual and follow its advice to create a 40-odd-line configuration file to tweak the update settings, then enter a



>50-character command to get it read in by the product, but once this was done things were all ready for us.

There seemed to be some proprietary on-access system in use alongside *Samba* vfs objects, but it looked similar enough to *dazuko* to make little difference. Once again, on-access speeds were excellent, hinting that some nifty improvements had been made in this area.

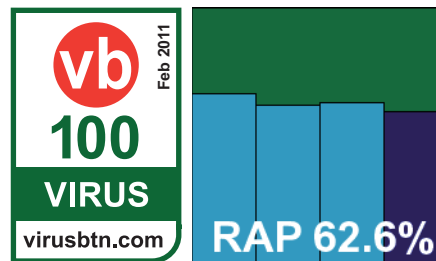
Once again logging proved problematic, with the default cap set even lower this time – an initial run produced suspect results despite backing up the log database file every 30 seconds. Retrying once this had been spotted

showed that the cap was removed after several restarts of the main service, but with time pressing some of the potentially suspect data still remained in the final results (which may thus be slightly inexact). Nevertheless, scores seemed close to those of *Kaspersky*'s first product – a fraction higher in most sets – but the same false positive, on a highly popular IM client, was enough to spoil *Kaspersky*'s chances of any VB100 awards this month despite a generally strong showing and solid coverage of the WildList set.

Norman Endpoint Protection 7.20

ItW	100.00%	Polymorphic	85.29%
ItW (o/a)	100.00%	Trojans	81.26%
Worms & bots	86.62%	False positives	0

Norman's product proved one of the most problematic at submission time, thanks to the requirement that it be installed with a live web














connection. This was hastily performed on the deadline day

Archive scanning		ACE	CAB	EXE-RAR	EXE-ZIP	JAR	LZH	RAR	TGZ	ZIP	ZIPX	EXT*
Avast	Default	X/√	X/√	√	√	X/√	X/√	√	√	√	√	√
	All	√	√	√	√	√	√	√	√	√	√	√
AVG	Default	X/√	X/√	√	√	X/√	X/√	X/√	X/√	X/√	X/√	√
	All	X	X	X	X	X	X	X	X	X	X	√
Avira	Default	2	√	√	√	√	√	√	√	√	X	√
	All	2	√	√	√	√	√	√	√	√	X	√
BitDefender	Default	√	√	8	8	√	√	√	√	√	√	√
	All	√	√	8	8	√	√	√	8	√	√	√
Central Command	Default	2	√	√	√	√	X	√	√	√	√	√
	All	X	X	X	X	X	X	X	X	X	X	√
eScan	Default	√	√	8	8	√	√	√	8	√	√	√
	All	√	√	8	8	√	√	√	8	√	√	√
ESET	Default	√	√	√	√	√	√	√	5	√	√	√
	All	√	√	√	√	√	√	√	5	√	√	√
Frisk	Default	5/√	5/√	5/√	5/√	5/√	√	3/√	2/√	5/√	5/√	√
	All	5/√	5/√	5/√	5/√	5/√	√	3/√	2/√	5/√	5/√	√
Kaspersky AV	Default	√	√	√	√	√	√	√	√	√	√	√
	All	X/√	X/√	X/√	X/√	X/√	X/√	X/√	X/√	X/√	X/√	√
Kaspersky ES	Default	√	√	√	√	√	√	√	√	√	√	√
	All	X/√	X/√	X/√	X/√	X/√	X/√	X/√	X/√	X/√	X/√	√
Norman	Default	X	√	8	1	√	√	√	8	√	X	√
	All	X	X	X	X	X	X	X	X	X	X	√
Quick Heal	Default	X	√	X	X	√	X	√	X	√	X	√
	All	2	X	X	X	X	X	X	X	X	X	√
Sophos	Default	X	X/5	X/5	X/5	X/5	X/5	X/5	X/5	X/5	X/5	X/√
	All	X	X/√	X/7	X/7	X/√	X/√	X/√	X/7	X/√	X/√	√
VirusBuster	Default	2	√	√	√	√	X	√	√	√	√	√
	All	X	X	X	X	X	X	X	X	X	X	√

Key: X - Archive not scanned; X/√ - Default settings/thorough settings;√ - Archives scanned to depth of 10 or more levels; [1-9] - Archives scanned to limited depth; EXT* - Eicar test file with random extension; All others - detection of Eicar test file embedded in archive nested up to 10 levels.

(Please refer to text for full product names)

Reactive And Proactive (RAP) scores		Reactive			Reactive average	Proactive Week +1	Overall average
		Week -3	Week -2	Week -1			
Avast		92.27%	98.25%	95.98%	95.50%	82.55%	92.26%
AVG		97.13%	97.11%	88.86%	94.37%	78.07%	90.29%
Avira		99.71%	99.72%	96.05%	98.49%	93.55%	97.26%
BitDefender		98.76%	99.38%	99.01%	99.05%	92.10%	97.31%
Central Command		88.69%	90.68%	85.35%	88.24%	83.64%	87.09%
eScan		97.21%	95.98%	94.12%	95.77%	87.10%	93.60%
ESET		96.53%	96.34%	94.89%	95.92%	91.51%	94.82%
Frisk		80.83%	75.41%	62.50%	72.91%	76.00%	73.69%
Kaspersky AV		91.32%	90.54%	86.68%	89.51%	85.37%	88.48%
Kaspersky ES		92.41%	92.08%	86.71%	90.40%	83.25%	88.61%
Norman		66.49%	61.56%	62.78%	63.61%	59.48%	62.58%
Quick Heal		92.43%	85.20%	76.69%	84.78%	82.07%	84.10%
Sophos		91.46%	90.81%	92.84%	91.71%	87.72%	90.71%
VirusBuster		88.69%	90.68%	85.35%	88.24%	83.64%	87.09%

(Please refer to text for full product names)

– a little too hastily as it turned out, as the install process announces itself to be complete and returns control to the command line well before it has actually finished running. Our first attempt – when the network was reset to internal only as soon as the install seem to be done – was missing large portions of the product and a second attempt was needed. This time all went OK, but we found that most of the components refused to function without an X Windows system in place. We eventually managed to get some on-demand work done, but found that configuration for the on-access component was not possible without a graphical set-up (there was some confusion over whether or not a web-based GUI was expected to be fully functional – either way, we had no luck trying to use it).

In the end, we went ahead and installed the *Ubuntu* desktop system on one of the test machines – which was something of a mammoth task as it was not included with the standard install media and took some two hours to download, prepare and set up. With this done we finally got to see the interface, which closely resembled those of *Norman's Windows* products, and was plagued with the same wobbliness, time lags and occasional freakouts. All we used in the end was the option not to automatically clean files spotted on access, and the desktop was then shut down for the speed measures.

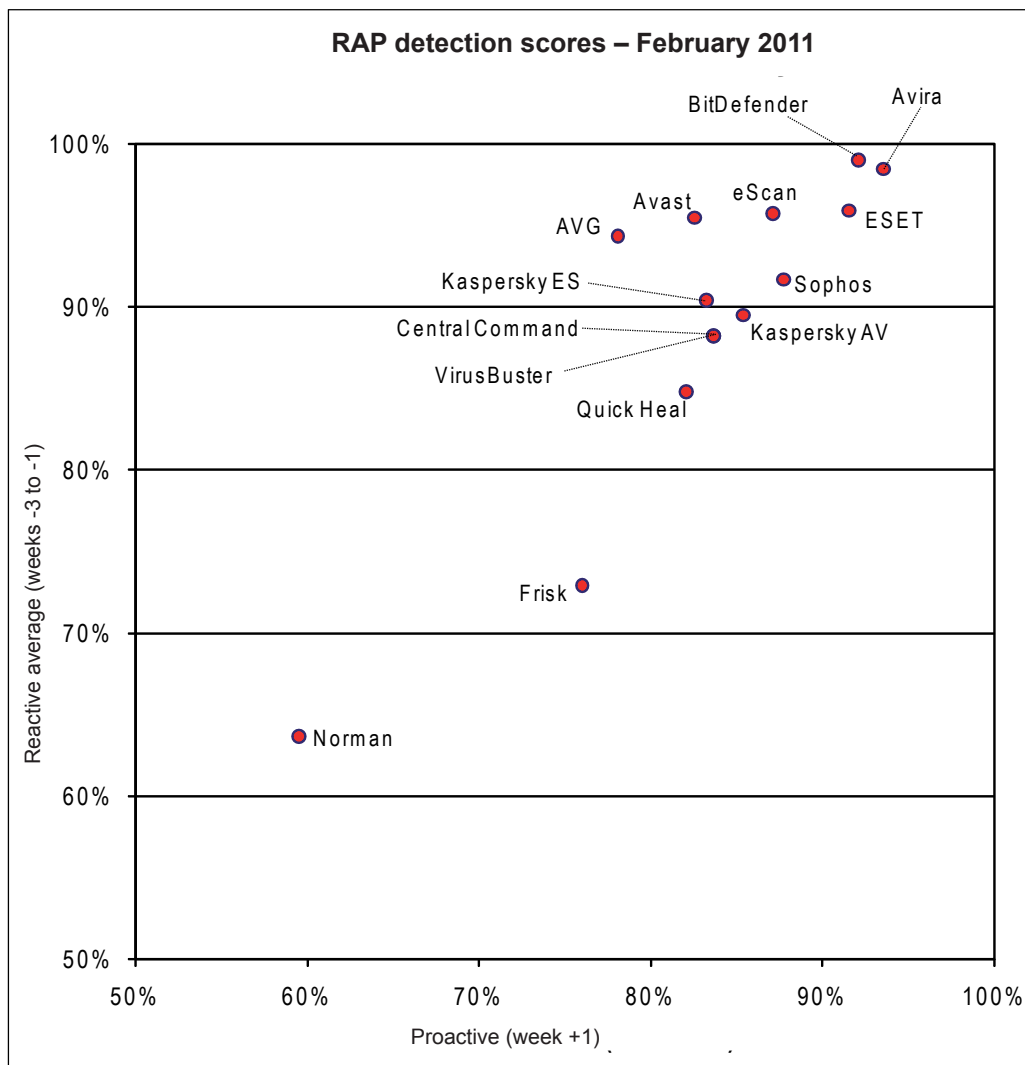
These showed the usual fairly slow times on demand, as the *Sandbox* system carefully picks each file apart. Much the same was observed on access, for the first run at least, but in the ‘warm’ measures, where files were checked for the second and subsequent time, an impressive improvement was observed.

Scanning of the infected sets was extremely slow – in part thanks to the deep *Sandbox* analysis – and occasionally flaky, with several runs failing to complete, or stopping output to logs part-way through. Several re-runs over two full weeks and on several systems, were still not quite complete several days after the deadline for this report, and as a result some of the data presented relies in part on on-access scores, which may be a fraction lower than the product’s full capability on demand. Detection rates were less than staggering, but not too disappointing, and with the WildList and clean sets causing no problems, a VB100 award is just about earned after all our efforts.

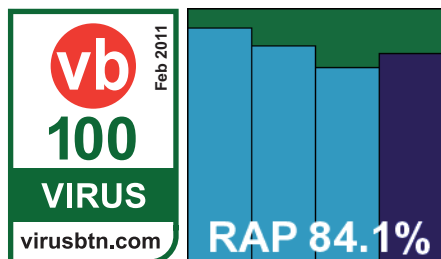
Quick Heal Anti-Virus for Linux 12.00

Virus database: 04 January 2011

ItW	100.00%	Polymorphic	96.94%
ItW (o/a)	100.00%	Trojans	95.57%
Worms & bots	96.86%	False positives	0



Back to something much simpler and more user friendly, *Quick Heal's* 141MB zip archive unpacked to reveal several folders and a nice install script, which took us through the steps of getting everything up and running. After resolving a single dependency, all went smoothly, including the set-up of *dazuko* - *Quick Heal* was one of only a few products to do this itself rather than dumping the work on the sysadmin.



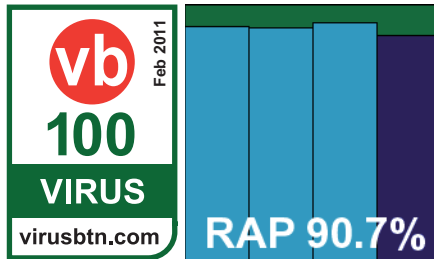
Configuration and documentation was clear, although man pages were lacking, and with simple, intuitive controls, testing went ahead without problems. Speeds were not brilliant, and overheads perhaps a little on the heavy side, but detection rates were impressive throughout. With no problems in the core sets, *Quick Heal* comfortably makes the grade for VB100 certification this month.

Sophos Anti-Virus for Linux 7.2.3

Engine version 3.15.0; Virus data version 4.61; User interface version 2.07.298

ItW	100.00%	Polymorphic	100.00%
ItW (o/a)	100.00%	Trojans	93.97%
Worms & bots	96.75%	False positives	0

Sophos was another product that took most of the load off the installer's shoulders, with its 232MB .tgz bundle containing a comprehensive installer utility. Detection of platform, compilation and insertion of required modules and so on was all carried out smoothly and automatically. A proprietary on-access hooking module is included.



Configuration was again via several control utilities, which were perhaps less than clear in their usage instructions and difficult to operate from a purely command-line setting. A web interface was also provided, but we never got it working, mainly because the default settings got us through most of the jobs we needed to carry out without much trouble.

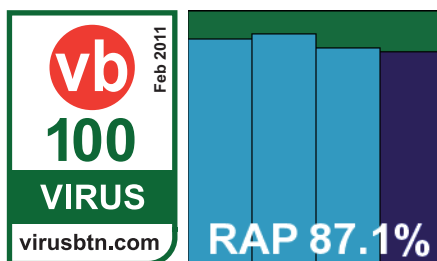
Scanning speeds were excellent (especially using the default settings, where no archive types are analysed), and on-access overheads were among the very lightest. Detection rates were not bad, with RAP scores a little below what we have come to expect from this product, but fairly strong nevertheless. The core sets presented no issues, and *Sophos* easily earns its VB100 award this month.

VirusBuster for Samba Servers 1.2.3_3-1.1_1

Scanner 1.6.0.29; virus database version 13.6.130.0; engine 5.2.0.28

ItW	100.00%	Polymorphic	90.52%
ItW (o/a)	100.00%	Trojans	83.45%
Worms & bots	95.69%	False positives	0

VirusBuster's product proved one of the simplest to set up and test, thanks to a very similar process having already been performed with the *Central Command* solution. Running the installer scripts and following instructions to set up *Samba* settings took just



a few minutes. The on-demand scanner has a slightly quirky syntax but is soon rendered familiar and friendly. However, trawling through the several configuration files in /etc in the vain hope of finding some settings for the on-access scanner was abandoned quickly.

Scanning speeds proved very good indeed, with similarly impressive on-access lags, and detection rates were pretty decent too. With just a single item in the clean sets warned about, being protected with the Themida packer, *VirusBuster* has no problems claiming its latest VB100 award.

CONCLUSIONS

As is usually the case with our *Linux* tests, it was something of a roller-coaster month, with moments of joy and comfort intermingled unpredictably with moments of bafflement and horror. For the most part, the products lived or died by the clarity of their documentation and the simplicity of their approach; the usability of a tool is usually significantly greater if it runs along the same lines as other things of a similar ilk, rather than attempting a radical new approach. For those wishing to try something new, demanding that the user read carefully through several hundreds of pages of documentation – which cannot even be displayed on the machine they're trying to use the product on – may be a little much.

Thankfully, stability has been no more than a minor problem here – as one would perhaps expect from a platform which tends to need far fewer restarts than some others. Nevertheless, we did see a few problems – notably with GUIs and with those command-line tools which try to hijack and do overly funky things with the console display, returning it to its owner bedraggled, battered and occasionally broken. All in all, we saw a strong batch of performances, with a high percentage of passes; an unlucky maximum file size setting and a single clean sample (a popular product, but a fairly old version with limited usage) caused the only issues in the certification sets. Part of this is doubtless down to the solid field of regular high-achievers, but part may also be thanks to the absence of any new complex viruses.

We expect to see a tougher task next time around, when we revisit *Windows XP* and see just how many other products there are out there.

Technical details:

All products were tested on identical machines with AMD Phenom II X2 550 processors, 4GB RAM, dual 80GB and 1TB hard drives, running *Ubuntu Linux Server Edition 10.04.1 LTS i386*. On-access tests were performed from a client system running *Windows XP Professional SP3*, on the same hardware.

END NOTES & NEWS

RSA Conference 2011 will be held 14–18 February 2011 in San Francisco, CA, USA. For more information see <http://www.rsaconference.com/2011/usa/>.

The 12th annual CanSecWest conference will be held 9–11 March 2011 in Vancouver, Canada. See <http://cansecwest.com/>.

The 8th Annual Enterprise Security Conference will be held 14–15 March 2011 in Kuala Lumpur, Malaysia. The theme for the 2011 conference is 'Improving digital security to protect your assets from malicious cybercrime'. For details see <http://www.acnergy.com/>.

Black Hat Europe takes place 15–18 March 2011 in Barcelona, Spain. For more information see <http://www.blackhat.com/>.

Infosecurity Europe will take place 19–21 April 2011 in London, UK. For more details see <http://www.infosec.co.uk/>.

SOURCE Boston 2011 will be held 20–22 April 2011 in Boston, MA, USA. For more details see <http://www.sourceconference.com/>.

The New York Computer Forensics Show will be held 26–27 April 2011 in New York, NY, USA. For more information see <http://www.computerforensicsshow.com/>.

The 5th International CARO Workshop will be held 5–6 May 2011 in Prague, Czech Republic. The main theme of the conference will be 'Hardening the net'. Details will be available soon on the conference website at <http://www.caro2011.org>.

The 20th Annual EICAR Conference will be held 9–10 May 2011 in Krems, Austria. This year's conference is named 'New trends in Malware and Anti-malware techniques: myths, reality and context'. A pre-conference programme will run 7–8 May. For full details see <http://www.eicar.org/conference/>.

The 6th International Conference on IT Security Incident Management & IT Forensics will be held 10–12 May 2011 in Stuttgart, Germany. See <http://www.imf-conference.org/>.

TakeDownCon takes place 14–19 May 2011 in Dallas, TX, USA. The event aims to bring together security researchers from corporate, government and academic sectors as well as the underground to present and debate the latest security threats and disclose and scrutinize vulnerabilities. For more details see <http://www.takedowncon.com/>.

The 2nd VB 'Securing Your Organization in the Age of Cybercrime' Seminar takes place 24 May 2011 in Milton Keynes, UK. Held in association with the MCT Faculty of The Open University, the seminar gives IT professionals an opportunity to learn from and interact with security experts at the top of their field and take away invaluable advice and information on the latest threats, strategies and solutions for protecting their organizations. For details see <http://www.virusbtn.com/seminar/>.

The 2011 National Information Security Conference will be held 8–10 June 2011 in St Andrews, Scotland. Registration for the event is by qualification only – applications can be made at <http://www.nisc.org.uk/>.

The 23rd Annual FIRST Conference takes place 12–17 June 2011 in Vienna, Austria. The conference promotes worldwide coordination and cooperation among Computer Security Incident Response Teams. For more details see <http://conference.first.org/>.

SOURCE Seattle 2011 will be held 16–17 June 2011 in Seattle, WA, USA. For more details see <http://www.sourceconference.com/>.

Black Hat USA takes place 30 July to 4 August 2011 in Las Vegas, NV, USA. For more information see <http://www.blackhat.com/>.

VB2011 takes place 5–7 October 2011 in Barcelona, Spain. VB is currently seeking submissions from those wishing to present at the conference. Full details of the call for papers are available at <http://www.virusbtn.com/conference/vb2011>. For details of sponsorship opportunities and any other queries relating to VB2011, please contact conference@virusbtn.com.

ADVISORY BOARD

Pavel Baudis, *Alwil Software, Czech Republic*
Dr Sarah Gordon, *Independent research scientist, USA*
Dr John Graham-Cumming, *Causata, UK*
Shimon Gruper, *NovaSpark, Israel*
Dmitry Gryaznov, *McAfee, USA*
Joe Hartmann, *Microsoft, USA*
Dr Jan Hruska, *Sophos, UK*
Jeannette Jarvis, *Independent researcher, USA*
Jakub Kaminski, *Microsoft, Australia*
Eugene Kaspersky, *Kaspersky Lab, Russia*
Jimmy Kuo, *Microsoft, USA*
Costin Raiu, *Kaspersky Lab, Russia*
Péter Ször, *Independent researcher, USA*
Roger Thompson, *AVG, USA*
Joseph Wells, *Independent research scientist, USA*

SUBSCRIPTION RATES

Subscription price for 1 year (12 issues):

- Single user: \$175
- Corporate (turnover < \$10 million): \$500
- Corporate (turnover < \$100 million): \$1,000
- Corporate (turnover > \$100 million): \$2,000
- *Bona fide* charities and educational institutions: \$175
- Public libraries and government organizations: \$500

Corporate rates include a licence for intranet publication.

See <http://www.virusbtn.com/virusbulletin/subscriptions/> for subscription terms and conditions.

Editorial enquiries, subscription enquiries, orders and payments:

Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England

Tel: +44 (0)1235 555139 Fax: +44 (0)1865 543153

Email: editorial@virusbtn.com Web: <http://www.virusbtn.com/>

No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein.

This publication has been registered with the Copyright Clearance Centre Ltd. Consent is given for copying of articles for personal or internal use, or for personal use of specific clients. The consent is given on the condition that the copier pays through the Centre the per-copy fee stated below.

VIRUS BULLETIN © 2011 Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England. Tel: +44 (0)1235 555139. /2011/\$0.00+2.50. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form without the prior written permission of the publishers.