MARCH 2012

virus BULLETIN

Fighting malware and spam

CONTENTS

2 **COMMENT**

Why you need to hack yourself

3 **NEWS**

Certification group announced

Advanced stealth techniques used to avoid detection

3 **VIRUS PREVALENCE TABLE**

MALWARE ANALYSES

4 Not 'Mifeve'-ourite thing

9 DroidDream mobile malware

SPOTLIGHTS

12 What is DMARC and should you care?

14 NCSC: public-private cooperation is key

17 **END NOTES & NEWS**

IN THIS ISSUE

DOING THE MATHS

MATLAB may not be the first platform that comes to mind when talking about viruses, but with its vast collection of mathematical functions it lends itself to all kinds of problem-solving mischief. Peter Ferrie details the MLS/Mifeve virus.

page 4

CARING ABOUT DMARC

In early February, a new security project known as DMARC (Domain-Based Message Authentication, Reporting and Conformance) hit the headlines. The project involves some of the best known companies on the Internet and attempts to reduce email-based abuse by solving a couple of long-standing issues related to email authentication protocols. John Levine has all the details.

page 12

SECURITY WITH COOPERATION

The Dutch National Cyber Security Centre (NCSC) was officially opened in January. Wout de Natris looks at how the Dutch government aims to achieve a safe, open and stable information society, with a focus on public-private cooperation.

page 14

ISSN 1749-7027

vb

'Companies and governments need to hack themselves first.'

Jeremiah Grossman
WhiteHat Security

WHY YOU NEED TO HACK YOURSELF

Everyone who uses the Internet is affected by inherent security problems. Last I heard there are about two billion people online. I'd be willing to bet that the vast majority have had their computers hacked at some point and subsequently been infected with viruses, had online accounts taken over, or else they know people who have.

When it comes to hacking businesses and governments, computer breaches may even be more common than for the average individual. Professional cybercriminals are not only after money, they also seek to steal intellectual property, trade secrets, even military capabilities – all things vital to our economic well-being and national security. It's time we got serious about this problem.

The new threat landscape requires a different defence approach because laws against hacking have little impact when the perpetrators are transnational. And unfortunately, international law enforcement is ill-equipped at best. Safeguarding the Internet requires a whole new way of thinking. The solution: companies and governments need to hack themselves first.

My first step onto this path of realization came more than a decade ago when I hacked my own *Yahoo! Mail* account, just to see if I could. There was a way (several ways actually) to get into my inbox without even needing a password. But instead of exploiting the vulnerabilities, I let *Yahoo!* know the details – promptly and privately. *Yahoo!* was able to fix the issues and safeguard its mail users. Being proactive prevented a serious security breach and public relations nightmare.

A dialogue followed – hack *Yahoo!* before the 'bad guys' do – and I was offered a job. *Yahoo!* was open to hacking itself first. There are other companies taking this approach, like *Google*, *Mozilla* and *Facebook*, yet the majority of companies and governments rely on network and endpoint-security measures because they are mandated by compliance standards.

Compliance standards, generic as they are, really don't take into consideration an organization's actual security needs and, instead, apply a one-size-fits-all mentality that results in misplaced security spend. It also results in internal complacency for the business that thinks that checking a compliance box equals security. Because compliance standards impact companies differently, the outcomes related to following a compliance-only security programme are often detrimental, the opposite of the intended goal. For example, some organizations determine that the financial implications of non-compliance are less than the costs associated with compliance and decide to ignore the regulation despite a notification to comply.

However, government and/or industry-mandated regulations are only one barrier to protecting individuals, corporate intellectual property and national security. The other barrier is the traditional, outdated attitude toward security that asserts that firewalls and anti-virus software provide sufficient protection. They don't.

The majority of recent security breaches have been the result of web application vulnerabilities, an avenue of attack where firewalls and malware detection are of zero value. *AT&T*, *Citigroup*, *PBS*, *NASDAQ*, the CIA, *Siemens*, *Electronic Arts*, and the websites of thousands of others have been breached in just the last year, and most likely they were all stockpiled with traditional 'best-practice' controls. On an average commercial website, our labs can identify one or more security gaps, usually in under 20 minutes.

Some companies are taking steps toward identifying web application vulnerabilities by hacking themselves – the aforementioned *Google*, *Mozilla* and *Facebook* reward hackers that notify their security teams about security issues, collectively handing out millions in rewards so far – but this is just the first step.

The next step requires more security spend to go toward web application security given all the applications that users run on the web. The reality is that a problem as diverse and wide reaching as cybercrime cannot be solved by any one defence mechanism, but I'll tell you this: protecting the Internet requires a completely new way of thinking that goes beyond traditional security spend, current laws and compliance regulations if we want to see any measurable progress.

NEWS

CERTIFICATION GROUP ANNOUNCED

A new group has been set up to promote the benefits of formal cybersecurity training and certification.

Announced at the RSA Conference, the Cybersecurity Credentials Collaborative (C3) has been set up by a number of organizations that provide cybersecurity training and certification. Members include: ASIS International, Computing Technology Industry Association (CompTIA), International Council of Electronic Commerce Consultants (EC-Council), Global Information Assurance Certification (GIAC), International Association of Privacy Professionals (IAPP), IEEE Computer Society, ISACA, (ISC)2 and the National Board of Information Security Examiners (NBISE).

The group aims to highlight the value of cybersecurity certifications – both for individual workers and for the organizations that employ them. It also aims to advance the craft and practice of certification programme development and provide a forum to collaborate on matters of shared concern.

ADVANCED STEALTH TECHNIQUES USED TO AVOID DETECTION

Network security firm *Damballa* has issued a report describing the advanced stealth techniques being used by six prominent malware families to evade detection. The firm studied a new Zeus variant, Bamital, BankPatch, Bonnana, Expiro.Z and Shiz, and found that all six families have been using domain generation algorithms (DGAs) to escape detection by blacklists, signature filters and static reputation systems, and to hide their command-and-control (C&C) infrastructures.

The malware contains an algorithm that uses a 'seed' value (such as the current date), to generate hundreds of seemingly random domain names that all attempt to resolve to an IP address. However, only very few (or even only one) will actually resolve to an IP address. The attacker will register only a few (or one) of the domains and set them up so that they resolve to the malware's C&C infrastructure. The process repeats the next day – with the domains used for the previous day's connections discarded, thus reducing the chances of detection and protecting the C&C system from being shut down.

DGAs (also known as domain fluxing techniques) have been around for a few years, but according to *Damballa* – which is now able to detect and model DGA behaviour using machine-learning technology – the techniques have become more advanced and are increasingly being used by threats to evade detection and grow sizeable malicious networks.

Prevalence Table – January 2012[1]

Malware	Type	%
Autorun	Worm	9.02%
Java-Exploit	Exploit	5.72%
Heuristic/generic	Virus/worm	5.70%
Crack/Keygen	PU	4.47%
Blacole	Exploit	4.34%
Iframe-Exploit	Exploit	3.80%
Conficker/Downadup	Worm	3.78%
Adware-misc	Adware	3.63%
Heuristic/generic	Trojan	3.46%
BHO/Toolbar-misc	Adware	3.19%
Sirefef	Trojan	3.10%
JS-Redir	Trojan	2.98%
Agent	Trojan	2.50%
Downloader-misc	Trojan	2.36%
Sality	Virus	2.20%
FakeAV-Misc	Rogue	1.89%
AutoIt	Trojan	1.63%
Crypt	Trojan	1.58%
Kryptik	Trojan	1.55%
PDF-Exploit	Exploit	1.53%
LNK-Exploit	Exploit	1.49%
Virut	Virus	1.38%
Dropper-misc	Trojan	1.29%
Freeware-downloader	PU	1.20%
Encrypted/Obfuscated	Misc	1.07%
InstallCore	Adware	1.02%
Dofoil	Trojan	1.01%
Backdoor-misc	Trojan	0.92%
SWF-Exploit	Exploit	0.89%
Exploit-misc	Exploit	0.88%
Dorkbot	Worm	0.87%
Ramnit	Trojan	0.74%
Others[2]		19.73%
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 1

NOT 'MIFEVE'-OURITE THING

Peter Ferrie
Microsoft, USA

MATLAB is probably not the first platform that comes to mind when talking about viruses (despite a proof of concept having appeared in 2006[1]). However, with its vast collection of mathematical functions it lends itself to all kinds of problem-solving mischief, as we can see in the MLS/Mifeve virus.

ARTISTIC DIFFERENCES

The virus is extremely complex, but amazingly stable despite its size. All of the major bugs that I thought I had found (there were a few) turned out to be misunderstandings on my part (there were many). So, it has no great faults in terms of its logic. It does, however, have many faults in terms of its 'style'. The code has some non-optimal sections, but this contributes only a little to the size. For example, some blocks have been copied to other areas of the code, and then modified, which results in dead code due to the different context. Then there are little things like the fact that a line containing a minor bug has been reproduced multiple times – resulting in the bug appearing in multiple places. In one case, a string replacement function is used to search for a string that has already been replaced entirely in order to reach that line. In another case, a constant result is evaluated repeatedly due to its misplacement inside a while loop. In yet another case, a loop runs to completion without effect if a condition inside it evaluates to false. There is also heavy use of the fix(rand()) function, despite the existence of a single randi() function which combines the effects of both. Perhaps the author of the virus became as tired of writing it as I did of reading it.

LIFE STAGES

There are two versions of the code. One is a 'demonstration' version that shows the transformation of a simple message. The other is a 'release' version, which is fully self-replicating. The two versions have essentially the same engine functionality.

The virus goes through several steps to produce a new version of itself: (1) it splits its own code into parts; (2) it defines the parts in a random order; (3) it reconstructs the parts in the proper order using 'if/else' statements. The components of the 'if/else' statements are complex mathematical statements in the form of inequalities.

[1] Bontchev, V. Math baloney: yet another first. Virus Bulletin, June 2006, p.4. http://www.virusbtn.com/pdf/magazine/2006/200606.pdf.

STAGE 1

The virus begins by generating replacement names for each variable that is used in the code. The demonstration version has two key variables which are not in the list of variables, but the message to transform contains none of the names, so nothing is replaced. A funny bug exists here in both versions, which is that the random number generator has not been seeded yet. As a result, at least in earlier versions of *MATLAB*, the sequence of random numbers will be identical whenever *MATLAB* is restarted, until the generator is seeded (which might happen in the host code of an infected file). The virus avoids producing names that match existing variable names or keywords. The replacement names are between five and 19 lower case characters long, in the range of 'a' to 'y'. The letter 'z' cannot be generated due to a bug in the virus code.

At this point, the virus seeds its random number generator using the current time (there is a slightly different algorithm between demonstration and release versions, but the difference is not relevant) and the Mersenne Twister algorithm. The virus creates an array of offsets at which to split its own code. There can be as few as three offsets in both versions. For the demonstration version, there can be as many offsets as there are bytes in the code. For the release version, the maximum number of offsets is equivalent to about one fifth of the size of the code. The virus splits the code into parts whose size is determined by pairs of offsets, and creates a random name corresponding to each of the parts. The part names are between four and 18 lower case characters long, in the range of 'a' to 'y'. As before, the letter 'z' cannot be generated due to a bug in the virus code.

STAGE 2

The virus drops an ODE function file, which will be called if an ordinary differential equation is used. The name of the file is treated like the other variables in the virus code, and is therefore not constant, however its contents are. The virus creates a threshold for encrypting the individual parts. In the release version, there is an approximately 33% chance of encryption in all cases, an approximately 33% chance of no encryption at all, and a 'threshold' that is chosen randomly in all other cases (though it's really an anti-threshold, since it behaves as the upper limit – not the lower limit – for action). In the demonstration version, there is a 50% chance of encryption in all cases.

For each part of the virus code, the virus displays the index of the part and the number of parts as a kind of progress indicator. If the current part is not the first one, then with an approximately 60% chance, and if the previous part

has already been marked as processed, the virus chooses whether or not to encrypt it. If a randomly chosen value is below the 'threshold', then the part is encrypted. Otherwise, the part is stored as plain text. After processing, the part is appended to the previous part and marked as processed. This is followed by the generation of garbage code. The result is the line 'varPrev=[varPrev code]', where 'varPrev' is a random variable name.

If the current part is not already marked as processed, and if it is not the last part, then with an approximately 60% chance, and if the next part has already been marked as processed, the virus chooses whether or not to encrypt it. If a randomly chosen value is below the 'threshold', then the part is encrypted. Otherwise, the part is stored as plain text. After processing, the part is prepended to the previous part and marked as processed. This is followed by the generation of garbage code. The result is the line 'varPrev=[code varPrev]', where 'varPrev' is a random variable name.

If the current part is not already marked as processed, then the virus chooses whether or not to encrypt it. If a randomly chosen value is below the 'threshold', then the part is encrypted. Otherwise, the part is stored as plain text. After processing, the part is marked as processed, and the variable name is added to the list of defined variables. This is followed by the generation of garbage code. The result is the line 'varCurr=[code]', where 'varCurr' is a random variable name.

For each part, beginning with the second one, if the current part is already marked as processed, and if the previous part is also marked as processed, then with an approximately 60% chance, the virus will choose how to combine the current part. With an approximately 50% chance, the virus will combine the previous part and the current part into the previous variable and discard the current variable. Otherwise, it will combine the previous part and the current part into the current variable, and discard the previous variable. This is followed by the generation of garbage code. The result is the line 'varPrev=[varPrev varCurr]' or 'varCurr=[varPrev varCurr]', where 'varPrev' and 'varCurr' are random variable names.

After all parts have been processed, if any remain that have not been assigned, the code will execute the final routine repeatedly until all of them are assigned, and use a second method of garbage code generation. The end result is that all of the parts are combined into a single variable which will be the whole virus body. Once that operation is complete, there will be one garbage line for each real line.

ENCRYPTION

In order to encrypt the parts, the virus chooses randomly from several algorithms that are derived from a formal grammar: 'F(NOS)Q', 'F(SON)Q', 'F(S)Q'. Initially, each of these algorithms appears twice, thus there are two chances to select any of them. However, after the initial choice is made, one 'NOS' and one 'SON' algorithm is removed. The virus parses the algorithm while any of the 'S', 'O' or 'F' elements remain. Only these three need to be checked, because the 'N' and 'Q' elements will also be replaced while any of the others remain.

'S' is a start symbol. It is replaced either by 'if' or by another algorithm chosen randomly from a set. 'F' is a function symbol. It is replaced by a function chosen randomly from a set. 'O' is an operation symbol. It is replaced by an operator chosen randomly from a set. 'N' is a number symbol. It is replaced by a random floating-point number. This number is multiplied by 10 to produce a value which has a potentially non-zero digit to the left of the decimal point. 'Q' is a power symbol. It is either removed, or there is a 20% chance that it will be replaced. Given those rules, the grammar looks like this:

```
S -> F(NOS)Q | F(SON)Q | F(S)Q | if

if -> set of floating-point numbers

F -> sin | cos | exp | atan | sinh | cosh | log |
asin | acos | tan

N -> floating-point number

O -> "+" | "-" | ".*"

Q -> "" | ".^" Z

Z -> 2 | 3 | 4 | 5 | 6
```

If the number of parenthesis pairs exceeds 25, then the line is considered to be complex 'enough', and all algorithms are disabled to force the transformation to complete sooner. The 'if' that appears as part of the 'S' replacement is a placeholder for a set of random floating-point numbers ranging from zero to the length of the original string. Each of the values is multiplied by 100 to produce a value which has up to two digits to the left of the decimal point. The result of this transformation is a line such as 'tan(cos(1.396 2.*[75.6759 80.4688 ...]).^5.*5.7168)', which comes from 'F(F(NOS)QON)Q'.

This logic runs until several conditions have been satisfied. The conditions are: that the number of random numbers in the line is equal to the number of elements in the original string, that the line does not contain any INF (infinity) or NaN (Not a Number) or imaginary values, and that the sum of the values does not exceed 10,000. Then, with an approximately 33% chance, the sets are added together and a decryptor is produced which subtracts them. Otherwise, the sets are subtracted, and a decryptor is produced which adds them together. In the latter case, the order of the two sets is chosen randomly (that is, 'a+b' or 'b+a').

GARBAGE IN, GARBAGE OUT

The same routine is used both for encryption of parts and for generating the garbage code. In the case of garbage code, a random section of the code might be encrypted. In some cases, a randomly generated name will be used, but the minimum length of the name is reduced to a single character. In other cases, one of the defined names will be used. The garbage code is fully functional, and will construct decrypted code, but it would likely concatenate the parts in the wrong order. However, the garbage code is never executed by the virus. It exists simply to camouflage the real code. There are two methods of garbage code generation, but they differ only in the chance of generating particular sequences. If a randomly chosen value is below the 'threshold', then the garbage code is encrypted. Otherwise, it is stored as plain text. There is a 'bug' in this behaviour, which causes the garbage code to be distinguishable in some cases from the real code. It doesn't help much for detection purposes, but it does allow those lines to be skipped.

If at least half of the code parts have been marked, then a selection of real names will be used as garbage names. This is safe because the code is never executed. If the first method of garbage code generation is in use, then there are several conditions which are checked. With an approximately 10% chance, the garbage string is assigned to a real name. Otherwise, with an approximately 20% chance, and if at least one garbage name exists, the garbage string is prepended to a random name. Otherwise, with an approximately 40% chance, and if at least three garbage names exist, then two variables are concatenated in a random order. Otherwise, with an approximately 20% chance, the garbage string is appended to a random name. Otherwise, the garbage string is compiled from between three and nine random lower case characters, in the range of 'a' to 'y'. The string might be encrypted in the same way as for the real code. The result is then assigned to a real name.

If the second method of garbage code generation is in use, then with an approximately 30% chance, the garbage string is prepended to a random name. Otherwise, it is appended to a random name.

STAGE 3

Once the code has been processed completely, the next stage of obfuscation begins. There is a 50% chance that the appearance of 'if' statements will be changed. There is an approximately 70% chance that a random number of spaces between zero and seven will be used per true clause. Otherwise, four spaces will be used. There is an approximately 70% chance that the spacing in false clauses will be the same as the spacing in true clauses. Otherwise, a random number of spaces between zero and seven will be used. The virus generates 11 unique random names for use in producing the conditional statements. It avoids producing names that match existing variable names or keywords. The random names are between five and 19 lower case characters long, in the range of 'a' to 'y'.

FUNCTION CONSTRUCTION METHODS 2 AND 4

For each line of real code, the virus chooses one of five possible methods. For the second and fourth method, the virus generates a random name which we will call 'R1'. The random name is between two and five lower case characters long, in the range of 'a' to 'y'. The virus avoids producing a name that matches any of the functions 'sin', 'cos', 'exp', or 'atan', or keywords. The virus makes a copy of this name for later use. We will call the copy 'R2'.

The virus starts with the algorithm 'SOS'. The virus parses the algorithm while any of the 'S', 'O' and 'F' elements remain. 'S' is replaced by either R1 or R2 or another algorithm chosen randomly from a set. With a 50% chance, and if R1 still matches R2, the virus generates a replacement random name for R2. The random name is between two and five lower case characters long, in the range of 'a' to 'y'. The virus avoids producing a name for R2 for which either R1 or R2 is a substring of the other, or that matches any of the functions 'sin', 'cos', 'exp', or 'atan', or keywords. 'O' is replaced by an operator chosen randomly from a set. 'F' is replaced by a function chosen randomly from a set. If the number of parenthesis pairs exceeds 25, then the line is considered to be complex 'enough', and the two algorithms are disabled to force the transformation to complete sooner. This forms a partial transformation of the line. Further processing occurs later. Given those rules, the grammar looks like this:

```
S -> (SOS) | F(S) | R1 | R2
O -> ".*" | "+"
F -> sin | cos | exp | atan
```

METHOD 1

The first method is all about matrices. The virus generates two vectors that contain a randomly chosen number of entries between three and seven, and one vector that contains the square of the number of entries in the first vector. Each of the entries will contain a randomly chosen floating-point number in the range of 0 to 1. The virus chooses a matrix algorithm randomly from the set: dyadic product, direct matrix, 'toeplitz', 'vander', 'pascal', 'magic', 'hilb', 'invhilb', 'wilkinson' or 'rosser'. In the case of the 'toeplitz'

or 'vander' matrix algorithms, the algorithm will be applied to the first vector. In the case of the 'pascal', 'magic', 'hilb', 'invhilb' or 'wilkinson' matrix algorithms, the size of the first vector will be used as an immediate value, but the actual vector will not be used any further. The 'rosser' algorithm returns a constant matrix, and no parameters are needed.

With an approximately 34% chance per round, the virus prepends a function chosen randomly from the set: 'sin', 'cos', 'sinh', 'cosh', 'exp', 'tan', 'sqrt', 'real' and 'imag'. This check is performed randomly between one and three times.

The virus chooses a name from the variable list and prepends a function chosen randomly from the set: 'sum', 'max' and 'min'. With an approximately 66% chance, the virus will contract the matrix to a vector, and then the vector to a scalar. The virus uses a random floating-point number for the scalar, which might be a negative number.

The virus repeats the logic above, beginning with the 34% chance per round of prepending a function from the first set, and finishing by prepending a function from the second set. Then, the virus repeats the logic but with only a 15% chance per round. The logic is executed one more time, using the 15% chance per round again.

The virus determines which is the larger of the first vector and the variable or value which was chosen second. The virus constructs an 'if' statement consisting of an inequality that contains a combination of the first vector and the variable or value, followed by 'true else false' clauses. The virus chooses randomly which clause will hold the real code and which will hold the garbage code. The 'if' statement will be constructed appropriately to always reach the real code. The result is a pair of lines such as:

```
pcuwsd=[28.7828 22.4722 17.9312 13.5236 1.5371 1.17
18.6505];
if((tan(max(pcuwsd)))<cosh(sum(exp(sum(hilb(7))))))
```

METHOD 2

The second method is numerical integration. There is an approximately 40% chance that the virus will replace R2 with R1. The virus will choose two elements randomly from the set: 'pi', 'log(2)', 'sqrt(2)', 'sqrt(3)', 'float1', 'float2', 'float3', 'float4' and 'float5' (where float1–5 are floating-point numbers). The sign of the elements is chosen randomly. With an approximately 30% chance for each element, the sign will be negative. The virus places the smaller of the two values first, and the inequality will use 'quad' as its operator, for a one-dimensional integral. Otherwise, the virus will choose four elements randomly from the same set as described above. As above, the sign of the elements is chosen randomly. With an approximately 30% chance for each element, the sign will be negative. The virus will separate the four elements into two pairs, and place the smaller of the two values in each pair first. The inequality will use 'dblquad' as the operator, along with the second pair of elements, and R2, for a two-dimensional integral.

After constructing the expression, the virus evaluates it. The virus checks that the integration takes *at least* 100ms to complete, and that it succeeds. The accuracy of the result is improved until either the expression takes 'long enough', or the tolerance is too small for a solution to be found. If the tolerance is too small, then the expression is abandoned. The result is a line such as:

```
quad(@(kwam)kwam+sin((kwam+(((((kwam+kwam)+kwam)+k
wam).*cos(atan((kwam+kwam))))+kwam))),0.099816,0.2
3802,1e-20)
```

or

```
dblquad(@(jbxf,ckt)sin(jbxf)+sin(jbxf),log(2),0.8952,
0.92058,sqrt(2),1e-17)
```

but the variables 'kwam', 'jbxf' and 'ckt' in these examples are components of an anonymous function, and are not defined anywhere else.

METHOD 3

The third method is interpolation. The virus creates a vector between one and 54 values long, containing random floating-point numbers. The numbers are multiplied by 1,000 to produce values which have up to three digits to the left of the decimal point. The virus creates an expression that requires interpolation to solve. With an approximately 40% chance, the interpolation uses a cubic spline method. The interpolation will be performed on a random subset of the vector. The result is a line such as:

```
interp1(yxrwj,21.2865,'spline')
```

In this example, 'yxrwj' is a variable that was defined earlier.

METHOD 4

The fourth method is a differential equation. The virus creates an ordinary differential equation in three stages. The first stage constructs the right side of the equation. The second stage solves the differential equation and returns the solution array. The time span is chosen randomly, but with a very limited range. The lower bound is in the range of -3 to 3, and the upper bound is in the range of the lower bound plus 1 to 4. The initial condition is a random floating-point number between 0 and 3.9999. The ODE function file is used during this stage to check the size of an interval. If the interval is too

small then the equation will be abandoned. A small interval indicates the presence of a singularity. If the size of the interval is acceptable, then the third stage uses interpolation to check the value. If the maximum time is reached while attempting to solve the equation, then the solution probably contains a singularity and the equation will be abandoned. The result is a set of lines such as:

```
msbqybsbtjnpibjnbt=inline('cos(cdx).*udq','udq','cdx
');
[rjwqtivdaes,yniaaytxnfpswott]=ode45(msbqybsbtjnpibjn
bt,[1 4],3.4908);
interp1(rjwqtivdaes,yniaaytxnfpswott,1.8901)
```

where each line can be separated by garbage instructions and other inequalities using values that were constructed earlier.

METHOD 5

For the fifth method, the virus starts with the algorithm 'F(F(S))'. The virus parses the algorithm while any 'S' remains. 'S' is replaced either by 'R' or by another algorithm chosen randomly from a set which introduces the 'F' and 'D' symbols. If the number of 'F' and 'D' elements exceeds 10, then the line is considered to be complex 'enough', and all algorithms are disabled to force the transformation to complete sooner. The virus parses the resulting algorithm while any of the 'D', 'F' or 'R' elements remain. 'F' is replaced by a function that accepts one parameter, chosen randomly from a set of 46(!) standard *MATLAB* mathematical functions, covering many areas. 'D' is replaced by a function that accepts two parameters, chosen randomly from a set. 'R' is replaced by a random floating-point number between -5 and 4.9999. The grammar looks like this:

```
S -> F(S) | D(S,S) | R
F -> sin | ... | sec | ... | exp | ... | log | ...
| sqrt | ... | abs | angle | conj | imag | real |
unwrap | fix | floor | ceil | round | sign | airy |
expint
D -> hypot | dot | cart2pol | pol2cart | atan2
R -> floating-point number
```

If the number of parenthesis pairs is fewer than 25, then the line is considered to be acceptable, otherwise the expression is abandoned. The result is a line such as:

```
sinh(asech(angle(acosh(cart2pol(4.5123,cosh(acot(angl
e(cos(acsch(-3.4196)))))))))
```

METHODS 2–5

For all but the first method, the virus checks the result of the expression for two conditions. Specifically, the virus checks that the result of the expression is less than or equal to a random subtraction value, or greater than or equal to a random addition value. While both of those conditions remain true, the virus will adjust the subtraction and addition values by random increments with precision ranging from five to nine decimal places, until both conditions are false. With an approximately 60% chance, the addition or subtraction value will be placed in a random variable which will be used later. Otherwise, the value will be used directly. An 'if/else' statement will be constructed such that one clause will contain the real instruction, and the other will contain the garbage instruction. With a 50% chance, the 'if/else' statement will compare the result of the inequality with the subtraction value. Otherwise, the statement will compare the result of the inequality with the addition value. With a 50% chance, the comparison in the 'if/else' statement will be reversed so that the 'true' and 'false' clauses will be reversed. The result is a line such as:

```
if(aeeynvlqvsfivdjip>acot(atan(unwrap(tanh(acoth(a
tan(tanh(floor(nextpow2(hypot(2.3959,dot(1.3456,0.
72016)))))))))))
```

IF-THEN-WHAT ELSE?

If a block of code consists of an 'if/else' statement, then there is a 50% chance that any of the 'true' clause, the 'else' statement, the 'false' clause, and the 'end' statement will be concatenated to the following component part. This is applied to all of the component parts, such that the block might be collapsed into a single line. If the appearance of 'if' statements was chosen to be randomly changing, then there is an approximately 70% chance that a random number of spaces from zero to seven will be used per true clause. Otherwise, four spaces will be used. There is an approximately 60% chance that the spacing in false clauses will be the same as the spacing in true clauses. Otherwise, a random number of spaces from zero to seven will be used.

At this point, the virus walks backwards through the code and assigns the real code lines to the final code array. With an approximately 70% chance per line of real code, the virus will assign a variable definition line to the final code array. If all of the real code has been assigned but some variables have not, then with an approximately 50% chance per iteration, the virus will assign one of the remaining variables. This action is repeated until all variables have been assigned. The result of this is a randomly ordered set of variable definition lines.

SEEK AND DESTROY

Finally, the virus searches the current directory for *MATLAB* module files. For each file that is found that is less than 1,000 bytes long (this check filters out infected files, which cannot possibly be that small), and is not the ODE function

file that belongs to the virus, the virus opens and reads the entire file, line by line. The virus searches each line for '%' (comment) and '...' (line continuation), except if they appear inside quotation marks. If '%' is seen, then the virus discards the entire line. If '...' is seen, then the virus appends the next line to the current line at the point where the '...' began, and rescans the line repeatedly until no more '...'s are seen.

After the first pass has completed, the virus identifies potential insertion points. If the current line is not inside a logic block, then it is considered to be a potential insertion point. The virus searches each line for any one from the set: 'if', 'for', 'while', 'try', 'switch' and 'parfor'. If one is found, then it must either be at the exact start of a line, or immediately following spaces, semicolons, or tabs. It must also be either the only thing on the line (which seems to be illegal, at least for earlier versions of *MATLAB*), or followed immediately by spaces, a left parenthesis or tabs in order to be considered valid. This marks the beginning of a logic block. Once inside a logic block, the virus searches each line for 'end'. If it is found, then it must either be at the exact start of a line, or immediately following spaces, semicolons, or tabs. It must also either be the only thing on the line, or be followed immediately by spaces, semicolons or tabs in order to be considered valid.

After the second pass has completed, a subset of the potential insertion points is chosen randomly as actual insertion points. The virus inserts the code backwards (which is now forwards, because of the backwards assignment, as described previously) while there is code left to insert. Since there can be fewer insertion points than parts of the virus, multiple virus lines might be grouped at a single insertion point. Finally, the combination is written back to the file. There will always be at least one host line before one virus line. If there are more insertion points than parts of the virus, then the remaining host code is appended after the last virus line.

CONCLUSION

This virus appears to have been written in response to a possible detection method for a previous version, whereby the plain text virus body could be produced by concatenating the individual parts. That is not possible with this version because of the difficult expressions that would need to be solved in order to decrypt the parts. However, the very nature of the polymorphism in this version essentially substitutes one kind of plain text for another. There are plenty of interesting and constant characteristics that can be identified very quickly. This allows us to perform a deeper inspection of only the most likely candidates without the performance hit of spending a long time looking at random files. This is great for us, and obviously not the result that the virus writer was expecting.

MALWARE ANALYSIS 2

DROIDDREAM MOBILE MALWARE

John Foremost
Independent researcher, USA

In 2011 one of the most notable mobile malware threats emerged in the wild: DroidDream (also known as Pjapps, Myournet, Lotoor, DroidRooter, and by several other aliases). DroidDream is a fully fledged mobile bot once a mobile device is rooted, with the ability to install applications of choice, navigate to websites, add bookmarks to the browser, manipulate text and voice messages, and communicate with a remote command and control server.

DROIDDREAM

The name DroidDream is derived from some of the author's comments in the code: 'If the droid isn't dream, don't do anything evil, cause nightmares later.'

DroidDream was distributed in conjunction with dozens of legitimate applications, including games, ring tones, and more. Three developers – we20090202, kingmail2010 and Myournet – had their *Google* accounts suspended for spreading DroidDream code via the *Android Market*. Over 100 applications were distributed before the threats were identified and removed from various locations on the Internet. For example, Bowling Time, with MD5 d4fa864e edcf47fb7119e6b5317a4ac8, contains DroidDream. Other infected applications included:

> Advanced App to SD, Advanced Barcode Scanner, Advanced Compass Leveler, Advanced Currency Converter, Advanced File Manager, Advanced Sound Manager, App Uninstaller, Basketball Shot Now, Best password safe, Bubble Shoot, Chess, Color Blindness Test, Dice Roller, Falling Ball Dodge, Falling Down, Finger Race, Funny Face, Funny Paint, Hilton Sex Sound, Hot Sexy Videos, hot. goddchen.sexyvideos, Magic Hypnotic Spiral, Magic Strobe Light, Music Box, Omok Five in a Row, org.droiddream. yellow4, Photo Editor, Piano, power.nick.ypaint, power. power.rate, powerstudio.spiderman, proscio.app.nick.ypaint, Quick Delete Contacts, Quick Notes, Scientific Calculator, Screaming Sexy Japanese Girls, Sexy Girls: Japanese, Sexy Legs, Spider Man, Super Guitar Solo, Super History Eraser, Super Ringtone Maker, Super Sex Positions, Super Sexy Ringtones, Super Stopwatch & Timer, super.mobi.eraser, super.sancron.ringtones.sexysb, Supre Bluetooth Transfer, Task Killer Pro and Tie a Tie.

Hostile components added to such apps included:

> com.advanced.scientific.calculator, com.advanced. soundmanager, com.app.aun, com.apps.tosd,

com.beauty.leg, com.bubble, com.dice.power, com.dice.power.advanced, com.dodge.game.fallingball, com.droiddream.advancedtaskkiller1, com.droiddream.android.afdvancedfm, com.droiddream.barcodescanner, com.droiddream.basketball, com.droiddream.blueftp, com.droiddream.bowlingtime, com.droiddream.comparator, com.droiddream.compasslevel, com.droiddream.daltonismo, com.droiddream.fallingball, com.droiddream.game.omok, com.droiddream.glowhockey, com.droiddream.howtotie, com.droiddream.lovepositions, com.droiddream.musicbox, com.droiddream.passwordsafe, com.droiddream.pewpew, com.droiddream.sexringtones, com.droiddream.stopwatch, com.droiddream.system.app.remover, com.editor.photoenhance, com.fall.down, com.fall.soft.down, com.free.chess, com.free.game.finger, com.hg.panzerpanic1, com.hz.game.mrrunner1, com.magic.spiral, com.power.basketball, com.power.demo.note, com.power.magic.strobelight, com.power.supersolo, com.quick.delete, com.sex.japaneese.girls, com.sexsound.hilton, com.sexy.hotgirls, com.sexy.legs, com.spider.man and com.super.mp3ringtone.

Despite *Google* responding quickly to abuse reports, over 50,000 downloads of known infected applications had already taken place. DroidDream demonstrated how trivial it is to create an online identity and subvert the weakly authenticated and weakly protected *Android* application marketplace. Consumers are quick to download and install whatever looks great, and is free – but with thousands of malicious applications now having been authored for the platform, users often get more than they bargained for.

DroidDream is configured within AndroidManifest.xml to run along with an application's legitimate code. Reviewing

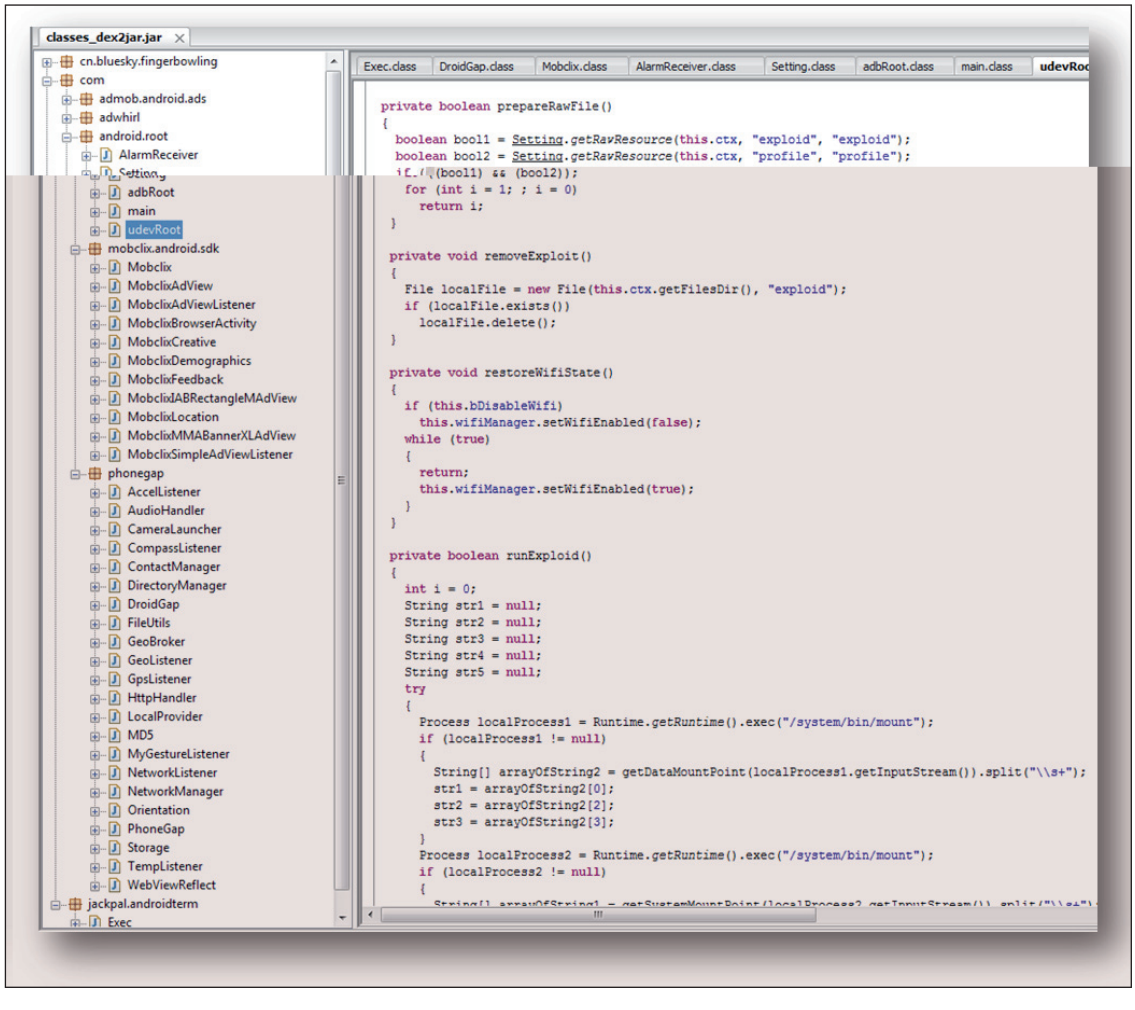

Figure 1: DroidDream-infected applications commonly include com.android.root. This is visible when viewing data from an APK, extracted, converted into a JAR, and then viewed within JD-GUI.

the AndroidManifest file of an infected application provides several clues immediately. The following is from what started out as a bowling game, but has been infected and become much more:

```
versionCode
versionNameinstallLocation
_____
namelabel
_____
iconscreenOrientation
configChangesprocessvalue
authorities
minSdkVersionandroid*http://schemas.android.
com/apk/res/androidpackagemanifest_______1.8com.
droiddream.bowlingtimeuses-permissionandroid.
permission.INTERNET#android.permission.READ_PHONE_
STATE$android.permission.CHANGE_WIFI_STATE$android.
permission.ACCESS_WIFI_STATEapplicationactivity&cn.
bluesky.fingerbowling.FingerBowlingcom.android.root.
main
intent-filteractionandroid.intent.action.
MAINcategoryandroid.intent.category.LAUNCHERservice:
remotecom.android.root.Setting:remote2-com.
android.root.AlarmReceiver       meta-datacom.
mobclix.APPLICATION_ID$E798E833-54EB-427E-
8289-8E67B27B41AA.com.mobclix.android.sdk.
MobclixBrowserActivityprovidercom.phonegap.
LocalProvider(com.droiddream.bowlingtime.
localprovideruses-sdk
```

DroidDream-infected applications commonly include com.android.root. This is visible when viewing data from an APK, extracted, converted into a JAR, and then viewed within *JD-GUI*, as shown in Figure 1.

Notice the string 'exploid' shown in blue. When run, the code attempts to leverage exploits developed by Sebastian Krahmer. The exploits are referred to as 'exploid' and 'rageagainstthecage'. If the exploits are leveraged successfully, the device is completely 'owned' and DroidDream is installed as a service called 'com.android.root.Setting'.

After installation, the bot may attempt to communicate with a remote C&C server. The original C&C involved with DroidDream was hxxp://184.105.245.17:8080/GMServer/GMServlet. Once a connection is established the bot attempts to report to the remote C&C the device's IMEI, Device ID, Line Number, and Subscriber ID. '/push/newandroidxml/' may be used for remote commands sent to bots. An infected device may then be controlled remotely by an attacker.

XML formatting is used in data communications with a remote C&C. For example, a command may be issued to call a premium rate line, resulting in charges to the victim's phone account. Com.android.root.adbRoot. crypto implements an XOR function to help obfuscate its communications.

DroidDream also includes an SQLite database management component. com.android.providers.downloadsmanager.DownloadCompleteReceiver runs in memory to look for an SQLite database sync. If a sync hasn't taken place for at least five days it initiates one. To ensure regular updates a task is scheduled to run every two hours, with a delay of two minutes between executions, via com.android.providers.downloadsmanager.d.

A remote C&C server is also capable of manipulating phone numbers, including blacklisting. DroidDream uses the format '($blacklist_URL) + "/?tel=" + ###', to blacklist numbers, where ### is a mobile number. Various commands may be used such as 'push:sms' for spimming, 'soft' for installing packages, 'window' for browser manipulation, 'mark' for adding a bookmark, and 'xbox' whose functionality is unknown. A prioritized list of browsers is included with the DroidDream installation, which includes com.android.browser, com.kolbysoft.steel, com.tencent.mtt, com.opera.mini.android, com.skyfire.browser, com.uc.browser and mobi.mgeek.TunnyBrowser.

FUTURE ATTACKS

With full functionality in such a bot, copycat attacks are likely to follow. One of the most dangerous parts of this code is the rooting component. Obviously all types of devices are vulnerable to various attacks that may allow for such rooting. *Android* is particularly vulnerable given the popularity of the operating system and the exploits that have emerged in recent months. Others are likely to be abused in a similar manner. Mobile malware attacks have matured from using the common social engineering tactic to more serious rooting attacks that can perform just about any function desired. From a vector standpoint, DroidDream reveals a small sliver of the widespread abuse that is ongoing even as this article is being authored, within a weakly authenticated and poorly controlled application market for *Android* and other operating systems.

ADDENDUM TO 'STATIC ANALYSIS OF MOBILE MALWARE'

The article 'Static analysis of mobile malware' (see *VB*, February 2012, p.6) referenced the freeware tool *DexID* but omitted to mention that the tool was developed by Dr Vesselin Bontchev, who made the tool available for free use to all interested in researching *Android* malware. The tool can be obtained via http://dl.dropbox.com/u/34034939/dexid.zip.

SPOTLIGHT 1

WHAT IS DMARC AND SHOULD YOU CARE?

John Levine[1]
Taughannock Networks, USA

In early February, a new group called DMARC (Domain-Based Message Authentication, Reporting and Conformance) received a great deal of press attention. Some of the breathless reporting suggested that this was the FUSPP (Final, Ultimate Solution to the Phishing Problem) – needless to say, it isn't. DMARC is a modest, but interesting security project involving some of the best known companies on the Internet[2].

Some of the big names involved in the group include: *Google*, *Yahoo!*, *Microsoft*, *AOL*, *Comcast*, *Facebook*, *American Greetings*, *LinkedIn*, *PayPal*, *Bank of America* and *Fidelity Investments*, along with infrastructure companies such as *Cloudmark* and *Return Path*. Having all these big gorillas on board means that whatever DMARC does is likely to have fairly widespread adoption. *Google* is already checking DMARC and sending status reports (described in more detail later).

BACKGROUND OF DMARC

Phishing is a huge problem for the institutions that are targeted in phishing campaigns, and indirectly for ISPs whose users fall for them and who have to help clean up the mess. Authentication schemes, notably DKIM and SPF, now provide tools to verify that a message was sent by the apparent sender (or more specifically, from a certain domain), but until now the ability to use that knowledge to deter phishing has been limited.

Part of the problem is that SPF and DKIM offer (by design) only limited tools for handling phishy email. They can tell recipients that they authenticate all their mail (the SPF -all option, and DKIM ADSP all and discardable), but that doesn't translate directly into useful advice for receivers. Furthermore, most large senders of emails will have a hodgepodge of sending systems, and it is a challenge to achieve 100% authentication coverage across all those systems. DMARC provides some support for senders with less than perfect authentication, and provisions for feedback so they can see how they're doing.

DMARC limits itself to what it calls 'domain phishing' – that is, phishes that use the exact domain name of the target, such as paypal.com or americangreetings.com. A lot of phishes use 'cousin' domains, which are similar but not identical to the target. I asked some of the DMARC group whether cousins would make DMARC irrelevant, and they told me that a surprising fraction of phishes actually use the exact domain. Since domain phishes are a technically much more tractable problem than cousins, that's where DMARC is starting.

PARTS OF DMARC

DMARC consists of three interrelated parts: an authentication framework, a way for domains to publish their policies, and a system for receivers to send feedback to senders. The draft specification (which is on the DMARC website at http://www.dmarc.org/) is subject to change, although I don't expect it to change much.

DMARC, SPF AND DKIM

The only identifier that DMARC authenticates is the domain of the address on the From: line, not Sender:, Resent-From:, or anything else. There are two ways to authenticate that domain, SPF and DKIM. The domain is authenticated if there is a successful SPF or DKIM check of a domain that matches the From: domain.

Authenticated domain matches can be either *strict* or *relaxed*, as determined by the sender. A strict match is an exact match – if the return address is sales@mktg.bigbank.com, the authenticated domain must be mktg.bigbank.com. A relaxed match only requires that the 'organizational domains' match. Roughly, that is the domain at the level at which it was registered with an external registry – such as bigbank.com or bigbank.co.uk. While there is no exact way to identify organizational domains, in practice it seems unlikely that this will be a problem since there aren't a lot of major phishing targets in domains with obscure registration points.

For SPF authentication, the receiver makes the usual SPF check on the envelope MAIL FROM address. If the check passes, and the domain in that address matches the one in the From: line, the domain is authenticated.

For DKIM authentication, the receiver performs the usual DKIM validation of any DKIM signatures on the message. If a valid signature has a d= domain that matches the one in the From: line, the domain is authenticated.

A From: domain is authenticated if any of the authentication methods (just SPF and DKIM at this point) succeed. There's no way for a sender to state which methods it uses – if it doesn't use one, it won't publish verification records so the method will fail, but that doesn't matter if another method succeeds.

[1] John Levine is, among other things, the designated liaison between DMARC and MAAWG.
[2] See http://www.dmarc.org/ for background information, a list of participating organizations and the current draft spec.

POLICY RECORDS

Senders can publish DMARC policy records to describe their signing policy, offer advice about what to do with mail that fails authentication, and ask for feedback reports.

A domain's DMARC record is a DNS TXT record named _dmarc.<domain>, where the domain is the domain in the From: line of mail the sender sends. The format of a DMARC record is similar to that for a DKIM record. Here's one of mine:

```
v=DMARC1; p=none; rf=afrf; rua=mailto:dmarc-a@abuse.
net; ruf=mailto:dmarc-f@abuse.net
```

It starts with a version tag, followed by a list of tag=value clauses. The p= tag must come first. Others are optional and can appear in any order. P stands for policy and indicates what the sender wants receivers to do with unauthenticated mail. The options are none, quarantine and reject. Quarantine is a request to turn up the filters, put the unauthenticated message into a spam folder, or otherwise treat it with extra scepticism, but still accept it. Reject is a request to reject the message at the end of the SMTP session, and not deliver it at all. None, which is the default, indicates that the receiver should handle the message however it would have been handled otherwise. It's up to a receiver how much attention it pays to the sender's suggestions, if any, since there's no way to tell whether an unknown sender's policy statement accurately represents what the sender really does. (This is a well known problem for SPF -all and ADSP discardable.) An optional sp= tag has the same values as p=, to be applied to subdomains.

To manage authentication, the aspf= and adkim= options specify whether to use relaxed or strict domain matching on SPF and DKIM, respectively.

The DMARC spec is a little vague about which DMARC record(s) a receiver should look up if the domain in a From: line or in the SPF or DKIM check is a subdomain of an organizational domain. That is, if the From: address is bob@sales.example.com, the receiver looks up _dmarc.sales.example.com, but if that's not found the receiver is then supposed to look up the organizational domain, _dmarc.example.com. Or, if the From: domain is bob@example.com and DKIM is d=sales.example.com, and _dmarc.example.com isn't found, it's not clear whether the receiver is supposed to look up _dmarc.sales.example.com. The draft spec mentions using DNS wildcards, but _dmarc.*.example.com doesn't do what one might hope. There are ways around this, but none is particularly elegant.

Since a site sending a lot of mail may take a while to get its authentication under control, two clauses in the policy record allow senders to try out policies while limiting the damage if they're wrong. The pct=NN clause specifies that the DMARC policy should be applied only on NN% of incoming mail, e.g. pct=5 would check and potentially quarantine or reject only every 20th message from the domain. The pct= clause doesn't affect reporting; any reports are supposed to include all mail received.

DMARC FEEDBACK

The rest of the DMARC spec is about receivers sending reports back to senders – both reports of individual authentication failures and daily (or more frequent) aggregate reports. In the _dmarc record, a sender can include an ruf=URI tag to tell receivers where to send individual failure reports, and an rua=URI tag to tell them where to send aggregate reports.

Individual reports can be in either IODEF (RFC 5007) or AFRF (Authentication Failure using ARF, still in draft form[3]). My impression is that most reports will be AFRF, since it is specifically designed to include elements needed to diagnose an SPF or DKIM failure.

Aggregate reports take the form of XML files compressed into a ZIP file, because reports for busy domains can be quite large. They are normally sent once a day, but the ri=NN tag can be used to request a reporting interval of NN seconds, such as ri=3600 for hourly reports. The XML includes a copy of the fields from the _dmarc record used to generate the report, together with a summary of all the sources that sent mail with the domain's From: address and the authentication results. *Google* is now sending daily reports – so far, it is the only receiver to do so. In one of my more heavily forged domains, a daily report included 672 entries, each of which was an IP address that sent one or more (often many more) messages purporting to be from my domain, along with information about DKIM signatures and the MAIL FROM domains checked by SPF, and what *Gmail* did with them. The reports are voluminous, and not easy for humans to read, but they are eminently suited to being parsed and put into a database. They can help to find both people forging one's domain, and equally important, legitimate mail that failed to authenticate.

The spec allows reporting URLs to be either mailto: (to send the report as a mail attachment) or http: (to upload it to a web server). At this point, *Google* only supports mailed aggregate reports, and as far as I can tell, nobody is sending failure reports at all. I've published DMARC records for most of the domains that my mail server handles, and have

[3] The current draft of AFRF, the spec for ARF authentication failure reports, is at http://tools.ietf.org/html/draft-ietf-marf-authfailure-report. It is likely to become an RFC in mid 2012.

received lots of aggregate reports from *Google*, but no individual reports yet.

WHERE NEXT?

DMARC is a work in progress, but an interesting one. The aggregate reports are worth getting, and I'd encourage anyone who cares whether their mail is delivered to publish a _dmarc record to collect daily reports. Most senders should publish a p=none policy (don't do anything special when the mail arrives, just send reports).

A few parts of DMARC still need to be cleaned up. One of those is the issue of subdomains and wildcards, as I mentioned above, to clarify what policy records apply to what subdomains.

Currently, a sender can put any email address or URL into the ruf= or rua= clauses, which offers a way to remotely mail bomb someone. My DNS server currently handles DNS for about 50 domains, so I've published 50 _dmarc records and get 50 daily reports from *Google* every morning. That's fine, since I want the reports and they go to a special mailbox I set up, but if I accidentally or deliberately misdirected the reports, and added ri=3600 to the _dmarc records so that the reports went out hourly, that could send over a thousand messages a day to an unwilling recipient. This is straightforward to fix, either by requiring that reports be sent back to the same domain as they're about, or by providing a way for the targets of the reports to publish their own DNS records to say that they want them. Since the reason they allow arbitrary addresses is probably to make it easy to send reports to third-party analysis services, the latter fix is more likely.

DMARC is designed to be extensible, so it's possible that other authentication schemes will be added, perhaps S/MIME, as well as finer-grained reporting. A huge gap, which the DMARC group acknowledges, is that it deals only with exact From: domain matches. If a message comes from accounts@banqofamerika.com, there's no way to tie that to a policy published by bankofamerica.com. Also, many mail programs display the From: line comment rather than the address, allowing spoofs like

```
From: PayPal Security <phish@criminal.biz>
```

These are vastly harder problems to address, so it makes sense that DMARC is starting with the low hanging fruit. It may well turn out that those problems are insoluble, and the only way to separate the real from the fake is to keep manual whitelists of known legitimate domains, put a gold star next to authenticated mail from them, and try to teach users that if it doesn't have a star, it's not your bank. But in order for that to happen, mail has to be authenticated in the first place, and DMARC is a small step towards making authentication work.

SPOTLIGHT 2

NCSC: PUBLIC-PRIVATE COOPERATION IS KEY

Wout de Natris
De Natris Consult, The Netherlands

On Thursday 12 January 2012 the Dutch National Cyber Security Centre (NCSC) was officially opened. With the push of a big red button, Minister of Security and Justice Ivo Opstelten proudly started a spectacular laser show in celebration of the event. Now that the lights have faded, let's take a look at what the Dutch government aims to achieve through the NCSC.

THE NATIONAL CYBER SECURITY STRATEGY

On 22 February 2011 the Dutch government published the National Cyber Security Strategy ('the strategy'). This document came about as the result of a motion adopted in Parliament [1] requiring an interdepartmental strategy. The document was created under pressure, but also in openness. Two semi-public meetings were organized, allowing all parties a chance to view, respond to and feed back on the first draft of the strategy. The public sessions saw civil servants from all relevant ministries gathered with cybersecurity experts and representatives from law enforcement, regulatory bodies and industry (including industries deemed to be vital to national security). The feedback gathered from these sessions found its way into the final version that was sent to Parliament. (For example, my feedback contributed to a more pronounced emphasis on international cooperation.)

The rationale behind the public meetings becomes clear when we cite the official government publication announcing the strategy:

> 'Government and industry will cooperate shoulder to shoulder to increase resilience against ICT disturbances and cyber attacks. A coherent approach is necessary and essential for the growing (international) problem' [2].

In other words, public-private cooperation is key. Two bodies were announced: the National Cyber Security Council and the NCSC. This cooperative approach is not unusual for the Netherlands. Permit me a brief history lesson about the 'polder model'.

Polder model

It is often said that 'polderen' is unique to the Netherlands. Since the Second World War, through a combination of negotiations and cooperation, government, industry and

unions have made consensus decisions on the road forward based on what is best for all concerned. It is believed that the consensus decision-making model may originate from the very early history of the northern and western parts of the Netherlands when (local) governments, cities, landlords and farmers worked together to contain rivers, dig canals, build and uphold dikes, create polders and win land over from the bogs and sea [3].

It is little surprise that the government has fallen back on this model to fight all things cyber – recent history has made it clear that no single actor alone can make a lasting impression on cyber perpetrators.

THE NATIONAL CYBER SECURITY COUNCIL

On 30 June 2011 Minister Opstelten instated the Cyber Security Council [4]. The two chairs and their respective backgrounds are indicative of the approach taken: Eelco Blok is CEO of *KPN*, the Dutch incumbent telecommunications company, and Erik Akerboom is National Coordinator for Counterterrorism and Security. The Council is responsible for advising government and industry alike (including the NCSC) on all matters concerning developments in cybersecurity. The Council can set priorities in the approach to ICT threats and assess the need for further research and development as well as determine how information can best be shared with participating public and private parties. Government, industry, end-users and academia are represented in the Council. However, the Council is a separate entity from the National Cyber Security Centre.

THE NATIONAL CYBER SECURITY CENTRE

The NCSC is based on three pillars that are highlighted in its mission statement: 'The NCSC cooperates in enhancing the defensibility of the Dutch society in the digital domain. Our goal is to realize a safe, open and stable information society by sharing knowledge, offering insight and also offering a proper action perspective' [5].

Incident response

The first goal manifests itself in the fact that the national Computer Emergency Response Centre, Govcert.nl, has been incorporated into the NCSC. The function of Govcert.nl hasn't changed, but will be added to. Despite the fact that, as Minister Opstelten stated at the opening, all outside government remain responsible for their own cybersecurity, the Centre will play a more central role than before. As this is a familiar function, I will not elaborate here, except to stress that in times of crisis the Centre will act as coordinating body between the different partners involved.

Expertise and advice

The second goal is about the development of knowledge and disseminating it to all partners. Two stages are foreseen at present. First, the government will intensify cooperation between the founding ministries and the relevant agencies, e.g. law enforcement agencies, AIVD (intelligence service), public prosecution and the National Forensic Institute. This will be achieved in part by embedding liaison personnel at the Centre.

Pim Takkenberg, Head of the Dutch National High Tech Crime Team, explains: 'The liaison personnel will be present at the NCSC for one or more days a week. They will establish a connection between their respective organizations and the NCSC and will be responsible for organizing the relevant or necessary expertise from within their organizations. In this way, not only is trust developed between the cooperating agencies, but also a common language. By reaching out and connecting in "normal" times, it becomes much easier and more natural to do so in times of crisis – which could possibly lessen the impact of incidents.'

Since 2006, regular meetings have been held in the Netherlands between law enforcement and security agencies to discuss cybercrime. This form of cooperation will now be taken to a new level as the liaison personnel will play an important role in times of crisis.

In the second stage, cooperation between the Centre and industry is foreseen. If everything goes according to plan, the Information Sharing and Analysis Centres (ISACs) created around and constituted by members of vital sector groups including telecoms, financial institutions, water and energy providers, etc., will link to the NCSC to make optimal use of information and actively share knowledge. The ISACs are already a feat of public-private partnership, although they are not unique to the Netherlands. At present they are organized through CPNI.nl [6]. Relevant industry partners from a vital sector gather with government, law enforcement, AIVD and Govcert to share threats, learn from and warn each other of perceived threats, and establish best practices in a safe, non-competitive environment. By treating cybercrime and threats as topics that require a common approach, putting competition aside, solutions and security for all can be established. As the sector provides the chair, industry is the driving force behind the agenda [7].

In my opinion this is the nucleus of the initiative. If all parties concerned can find, as Takkenberg puts it, 'a

common language', learn to work together and gain trust, the NCSC becomes the centre of expertise, excellence and esteem to which all concerned will look for guidance and coordination in times of crisis. Succeed here, and the rest will follow suit.

The NCSC has already published two reports. One describes how to recognize cybercrimes and when and how to report them [8]. The other is a report on ICT security guidelines for web applications [9]. The NCSC is already on the road to establishing itself as a centre of knowledge and advice.

Monitoring and reporting

Monitoring the threat level and reporting on it is the third pillar of the NCSC. The Centre aims for a broad participation, public and private, so it can collect data from divergent sources. This information is gathered at a more structural level, is more comprehensive and creates a better overview than ever before. Data can be studied, analysed, discussed, and reported to all the partners involved. The NCSC draws the analogy of laying out a puzzle: find and lay out all of the pieces in the correct order to get the complete picture. This way it 'will make an important contribution to increasing national resilience by means of the integral approach and the unique shape of the cooperation' [10]. The first national trend report on cybercrime and digital security in the Netherlands was prepared by Govcert and published on 12 November 2011 [11]. All relevant law enforcement and national security agencies contributed to the report for the first time.

INTERNATIONAL COOPERATION

As cybercrime does not stop at the border of nation states, the NCSC will also need to look to partners in other countries. At present the focus is on organizing itself, but in the future the Centre will reach out to other countries. In what form and with whom remains to be determined.

The EU, individual member states and several other countries are all contemplating how to go forward, but all seem to agree that a public-private form of cooperation is paramount. The Netherlands has established a blueprint on how to proceed. It could be worthwhile for other countries to study this model as a reference point for a way forward in the ongoing battle against cyber threats.

CONCLUSION

At present, the NCSC is a work in process. In 2011 a lot of effort was put into creating the Centre and getting very different organizations (and thus cultures) behind it. The coming months will undoubtedly pass with everyone finding their way, embedding liaison personnel, establishing optimal lines of contact and reaching out to industry through the ISACs. However, once all this has settled into place we will have a centre that shows the promise of being able to assess the level of cyber threats very quickly, and through its very foundation built on cooperation, will be able to coordinate in times of crisis at a national level, between all relevant parties. Next to that, a framework has been created to learn as well as teach lessons. As such, the NCSC holds a promise that goes far beyond the Dutch borders. It may not be unique in its intentions, but as an established, centralized centre it may well be so.

REFERENCES

[1] Motie Knops, Tweede Kamer, vergaderjaar 2009-2010, 32 123 X, nr. 66.

[2] http://www.rijksoverheid.nl/documenten-en-publicaties/persberichten/2011/02/22/nationale-cyber-security-strategie-gepresenteerd.html. (Translation, WdN).

[3] http://en.wikipedia.org/wiki/Polder_Model.

[4] http://www.rijksoverheid.nl/documenten-en-publicaties/persberichten/2011/06/30/cyber-security-raad-geinstalleerd.html.

[5] https://www.ncsc.nl/english/current-topics/news/the-national-cyber-security-centre-ncsc-bundles-knowledge-and-expertise.html.

[6] http://www.cpni.nl/.

[7] http://www.cpni.nl/informatieknooppunt/informatieknooppunt-cybercrime.

[8] https://www.ncsc.nl/dienstverlening/expertise-advies/kennisdeling/whitepapers/handreiking-cybercrime.html.

[9] https://www.ncsc.nl/dienstverlening/expertise-advies/kennisdeling/whitepapers/ict-beveiligingsrichtlijnen-voor-webapplicaties.html.

[10] https://www.ncsc.nl/english/current-topics/news/the-national-cyber-security-centre-ncsc-bundles-knowledge-and-expertise.html.

[11] https://www.ncsc.nl/dienstverlening/expertise-advies/kennisdeling/trendrapporten/nationaal-trendrapport-cybercrime-en-digitale-veiligheid-2010.html.

END NOTES & NEWS

APWG eCrime Researchers Sync-Up takes place 7–8 March 2012 in Dublin, Ireland. For more information see http://www.ecrimeresearch.org/2012syncup/cfp.html.

Black Hat Europe takes place 14–16 March 2012 in Amsterdam, The Netherlands. For details see http://www.blackhat.com/.

EC-Council Summit takes place 19–21 March 2012 in Washington, DC, USA. Other EC-Council Summits also take place this year in Boston (4–7 June) and San Jose (20–23 August). For details see http://www.eccouncil.org/training.aspx.

Cyber Defence Summit will be held 2–3 April 2012 in Muscat, Oman. The Summit aims to gather key regional stakeholders to discuss the heightened importance of cybersecurity throughout the Middle East. See http://www.cyberdefencesummit.com/.

SOURCE Boston 2012 will be held 17–19 April 2012 in Boston, MA, USA. For further details see http://www.sourceconference.com/boston/.

The 3rd VB 'Securing Your Organization in the Age of Cybercrime' Seminar takes place 19 April 2012 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 top security experts and take away invaluable advice and information. See http://www.virusbtn.com/seminar/.

Infosecurity Europe 2012 takes place 24–26 April 2012 in London, UK. See http://www.infosec.co.uk/.

The Sixth Counter-eCrime Operations Summit will be held 25–27 April 2012 in Prague, Czech Republic. For details see http://apwg.org/events/2012_cecos.html.

TakeDownCon Dallas takes place 4–9 May 2012 in Dallas, TX, USA. For more information see http://www.takedowncon.com/.

The 21st EICAR Conference takes place 7–8 May 2012 in Lisbon, Portugal. For details see http://www.eicar.org/17-0-General-Info.html.

The CARO 2012 Workshop will be held 14–15 May 2012 near Munich, Germany. For more information see http://2012.caro.org/.

NISC12 will be held 13–15 June 2012 in Cumbernauld, Scotland. The event will concentrate on 'The Diminishing Network Perimeter'. For more information see http://www.nisc.org.uk/.

The 24th annual FIRST Conference takes place 17–22 June 2012 in Malta. For details see http://conference.first.org/.

Black Hat USA will take place 21–26 July 2012 in Las Vegas, NV, USA. For more information see http://www.blackhat.com/.

The 21st USENIX Security Symposium will be held 8–10 August 2012 in Bellevue, WA, USA. For more information see http://usenix.org/events/.

VB2012 will take place 26–28 September 2012 in Dallas, TX, USA. *VB* is currently seeking submissions from those wishing to present at the conference – deadline 9 March. Full details are available at http://www.virusbtn.com/conference/vb2012/.

VB2013 will take place 2–4 October 2013 in Berlin, Germany. Details will be revealed in due course at http://www.virusbtn.com/conference/vb2013/. In the meantime, please address any queries to conference@virusbtn.com.

SUBSCRIPTION RATES

Subscription price for Virus Bulletin magazine (including comparative reviews) for one 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.

Subscription price for Virus Bulletin comparative reviews only for one year (6 VBSpam and 6 VB100 reviews):

- Comparative subscription: $100

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/