CASE STUDY:
CODE RED WORM

Parth Laturia-180050071
Rajat Jain-180100093
Devansh Chandak-180110027
Anish Deshpande-180100013
Sunil Meena-180050107

CONTENTS

1. Introduction

Code Red was a computer worm observed on the Internet on July 15, 2001. It attacked computers running Microsoft's IIS web server. It was the first large scale, mixed threat attack to successfully target enterprise networks.

The Code Red worm was first discovered and researched by eEye Digital Security employees Marc Maiffret and Ryan Permeh when it exploited a vulnerability discovered by Riley Hassell. They named it "Code Red" because Code Red Mountain Dew was what they were drinking at the time.

Although the worm had been released on July 13, the largest group of infected computers was seen on July 19, 2001. On this day, the number of infected hosts reached 359,000

CODE RED

Common name	Code Red
Technical name	CRv and CRvII
Type	Server Jamming worm
Isolation	July 15, 2001

2. Basic Concept

2.1 Exploited vulnerability:

The worm showed a vulnerability in the growing software distributed with IIS, described in Microsoft Security Bulletin MS01-033, for which a patch had been available a month earlier.

The worm spread itself using a common type of vulnerability known as a buffer overflow. It did this by using a long string of the repeated letter 'N' to overflow a buffer, allowing the worm to execute arbitrary code and infect the machine with the worm. Kenneth D. Eichman was the first to discover how to block it, and was invited to the White House for his discovery.

2.2 Worm payload:

The payload of the worm included:

- Defacing the affected web site to display:

  HELLO! Welcome to http://www.worm.com! Hacked By Chinese!

- Other activities based on day of the month:
  1. Days 1-19: Trying to spread itself by looking for more IIS servers on the Internet.
  2. Days 20–27: Launch denial of service attacks on several fixed IP addresses. The IP address of the White House web server was among those.
  3. Days 28-end of month: Sleeps, no active attacks.

When scanning for vulnerable machines, the worm did not test to see if the server running on a remote machine was running a vulnerable version of IIS, or even to see if it was running IIS at all. Apache access logs from this time frequently had entries such as these:

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNN
%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3
%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a  HTTP/1.0
```

The worm's payload is the string following the last 'N'. Due to a buffer overflow, a vulnerable host interpreted this string as computer instructions, propagating the worm.

3. Vulnerability Details

Name: Microsoft Internet Information Server (IIS) IDA/IDQ ISAPI Extension Buffer Overflow
CVE (Common Vulnerability and Exposure): CAN-2001-0500

Variants: Code Red II, Code Blue, Code Green

Systems Affected: Microsoft Windows NT 4.0 Internet Information Services 4.0, Microsoft Windows 2000 Internet Information Services 5.0, Microsoft Windows XP beta Internet Information Services 6.0 beta

Protocols Used: This vulnerability is accessible through the HyperText Transport Protocol (HTTP) protocol running over TCP/IP.

Services Used: This vulnerability is found in all versions of the Microsoft Internet Information Server (IIS) Web server running the Indexing service.

Description:

This vulnerability could allow an attacker, from a remote location, to gain full system level access to any server that is running a default installation of Windows NT 4.0, Windows 2000, or Windows XP and using the Microsoft Internet Information Services (IIS) Web server software.

The vulnerability lies within the code that allows a Web server to interact with Microsoft Indexing Service functionality, which is installed by default on all versions of IIS. The problem lies in the fact that the .ida (Indexing Service) ISAPI filter does not perform proper "bounds checking" on user inputted buffers and therefore is susceptible to a buffer overflow attack.

Attackers that leverage this vulnerability can perform any desired system level action, including but not limited to, installing and running programs, manipulating web server content, adding, changing or deleting files and even possibly using the compromised system to launch additional attacks directed against other systems.

This vulnerability was discovered by Riley Hassell of eEye Digital Security (http://www.eeye.com).

Payload was described in Section 2.2

4. Step by Step Analysis

Worm has 3 distinct actions: system infection, web page hack, and denial of service attack against www.whitehouse.gov. We will analyze them separately:

4.1 System Infection:

1. System infection begins when an IIS web server, which is vulnerable to the indexing buffer overflow, receives a HTTP get request that contains the Code Red exploit.
   The instruction pointer (EIP), which holds the address of the next instruction to be executed, is overwritten with an address that points to an instruction within msvcrt.dll. This causes the program flow to divert back to the stack and jump into the worm code that is held in the body of the initial HTTP request.

2. The initial code of the worm begins to execute.
   The worm sets up a new stack for its own use and then moves on to initialize its function jump table.

3. The worm begins to execute the data portion of the exploit.

   The worm then needs to set up a stack based internal function jump table to store function addresses (this gives the worm a better chance of executing cleanly on more systems).

   The worm loads the following functions:

From kernel32.dll	**From infocomm.dll**	**from WS2_32.dll**
GetSystemTime CreateThread CreateFileA Sleep GetSystemDefaultLangID VirtualProtect	TcpSockSend	socket connect send recv closesocket

The worm stores the base address of w3svc.dll which it will later use to potentially deface the infected website.

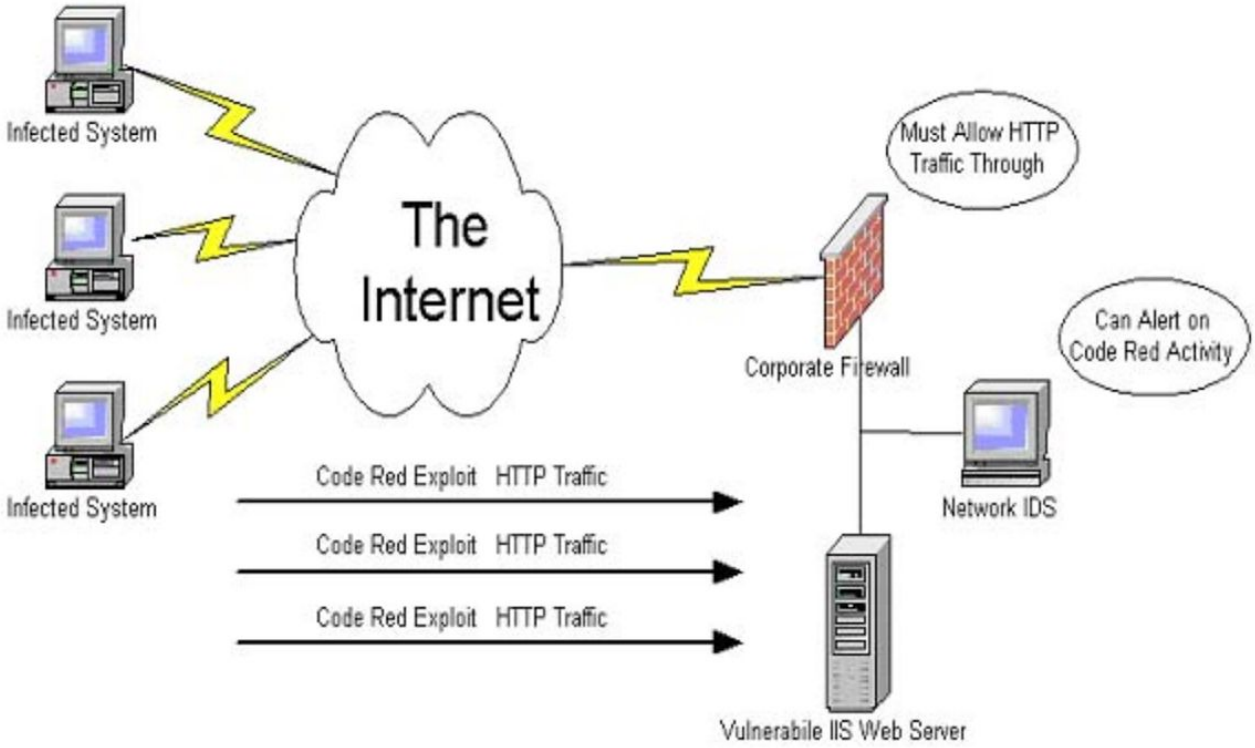
3. The worm performs a WriteClient (Part of the ISAPI extension API), sending "GET" back to the attacking worm possibly sending the message of a new infection.
4. The worm will count the number of threads currently in action. If the number of threads is 100 then control is passed to the web page hack functionality. If the number of threads is less than 100 the worm will create a new thread that is an exact replica of the worm.
5. The worm has a built in "lysine deficiency", a check to prevent the malicious code from spreading further.
   The worm performs a check for the file c:\notworm to determine if the worm has previously infected the system. If the file exists then the worm will become dormant. If the file does not exist then the worm will continue its infection.
6. The worm now determines the local time of the system (in UTC).
   If the time is greater than 20:00 UTC the worm will proceed to launch the denial of service attack against www.whitehouse.gov. If the time is less than 20:00 UTC the worm will continue to try and infect additional systems.
7. The worm will attempt to infect new hosts by sending the malicious code to any IP that it can connect to port 80 on.
   It uses multiple send()'s so packet traffic will be broken up. After a successful send it closes the socket and goes to step 6, repeating the loop infinitely.

4.2 Web Page Hack:

1. The worm will first attempt to determine if the local operating system language is English (US). If the infected host is an English (US) system then the worm will proceed to deface the local website with "Hacked by chinese !". If the system is not English (US) this worm thread will go to step 6 of the 'System Infection' functionality.
2. This worm thread now sleeps for two hours. The reason for this is not completely understood, although it is speculated that it gives the other threads time to spread the infection before making its presence known via the web page hack.
3. The worm now alters the systems web page by modifying code in memory, a technique known as 'hooking'. Modifications are made to w3svc.dll to allow the worm to change the data being written back to clients who request web pages of an infected server.
4. The worm then sleeps for ten hours after which this thread will return w3svc.dll to its original state.
5. Execution after this proceeds to step 6 of the 'System Infection' functionality.

4.3 Denial of Service Attack:

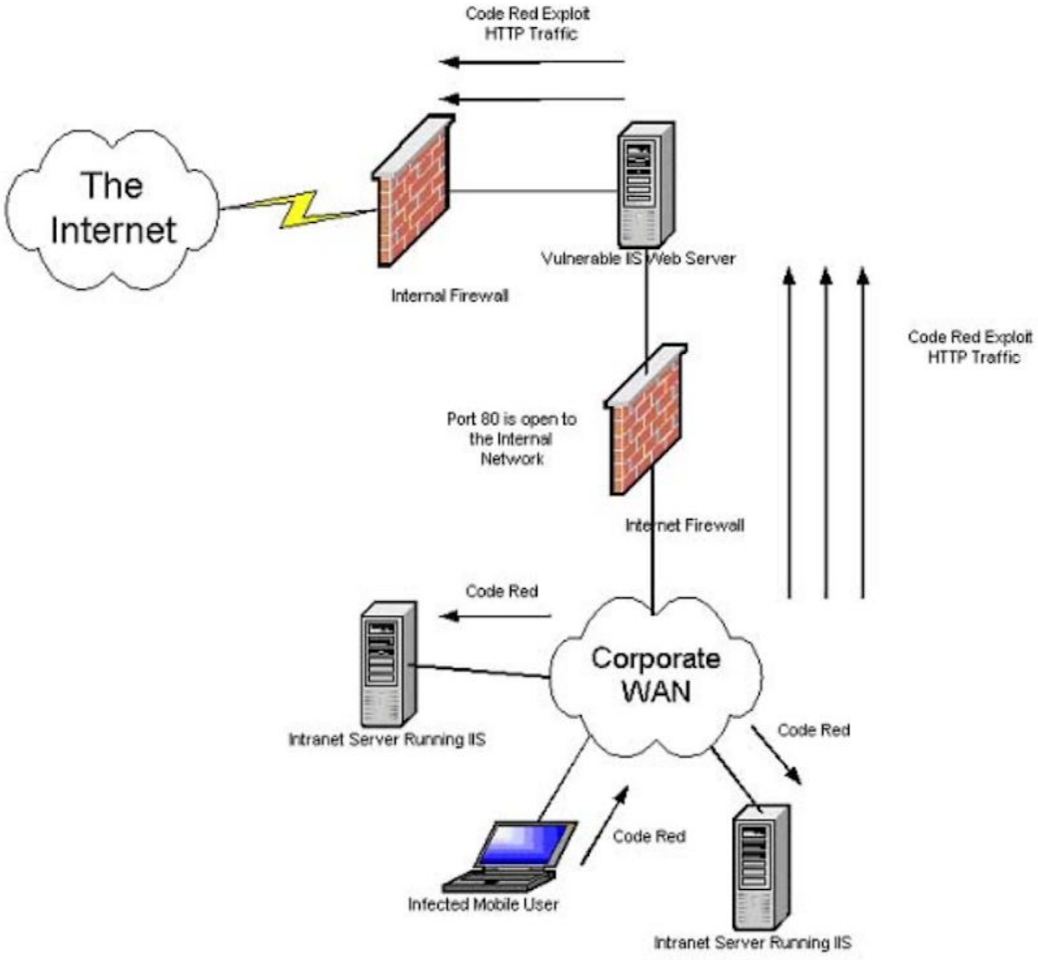
1. Each thread will attempt to target www.whitehouse.gov on port 80 by establishing a connection and sending 100k of data. If this connection is successful then the worm will create a loop that performs 18000h single byte send()'s to www.whitehouse.gov.
2. After this activity the worm will sleep for about four hours, it will then repeat this attack procedure.



The infected systems will scan the Internet looking for vulnerable systems.

The diagram above illustrates, from a very high level, how a typical company would have their web server connected to the Internet (in this case an IIS web server). People from all over the world would then be able to connect to this web server using a standard Internet browser. To allow for these connections the company would have to open TCP port 80 (HTTP default port) on their corporate firewall. Simple yet secure right. Not quite, there is one huge problem. The Code Red exploit also travels over TCP

port 80. The corporate firewall in this case offers absolutely no protection from the Code Red worm.



The worm can also be introduced internally, perhaps through an unprotected mobile user.

It is very common to see an organization focus less of their security efforts on their internal networks, this can prove to be a huge oversight. In the case illustrated above a default Windows 2000 server (residing on a mobile laptop) was attached to the internal network. This laptop could have been infected with Code Red through a home connection and later connected to the corporate WAN. At this point the worm can begin to wreak its havoc infecting all vulnerable machines in sight, internal and external.

5. Attack Signature

The following entry could be found in an IIS Web Server log if the system was probed by the Code Red worm (CRv1 and/or CRv2). It is important to note that this entry would be the same whether the system was infected or not. Therefore it is impossible to tell strictly of the basis of IIS logs if Code Red has infected the system. However, if entries like this are found and it is known that the system has not been patched for this vulnerability there is a very good chance that the system has been infected.

Identifying The Worm Through IIS Web Server Logs

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNN
%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3
%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a  HTTP/1.0
```

6. Protection against the Worm

6.1 Detection:

There are other possible means by which to detect the presence of Code Red on a network.

The system administrator may notice that the server is not performing as usual. Code Red may be using enough CPU cycles to impact the server's normal routine prompting the system administrator to investigate further.

A few techniques the administrator could employ are:

1. Using a 3rd party application (the process may be hidden from native tools) to determine exactly what processes are running. This would show if any unexpected processes were running.
2. Using the netstat command (netstat -an) to get a quick listing of all active TCP and UDP connections. This would show if there were an abnormal number of outbound connections originating from the server.

It may also be possible to use the auditing features of the corporate proxy server, the Internet firewall, or the border router to determine if any abnormal activity is taking place. These logs contain a wealth of information but are seldom reviewed.

6.2 Protection:

Now that the potential destructive capabilities of this worm have become evident, it is important to recognize some of the ways in which to protect against them. Many of the common security components or devices are not adequate to deal with this threat. Take a firewall for example. Since the worm travels over TCP port 80 traditional firewalls are not able to filter the traffic from reaching the web servers, lest the administrator wants to stop all valid web traffic as well.

IDS sensors are able to detect the signature of the worm but the actions they can take are limited. In the most extreme cases some IDS network sensors have the capability to reset connections. This will still not prevent the infection of a vulnerable system due to the fact that the exploit only needs to send one HTTP session for the worm to spread, this eliminates the possibility of the IDS resetting the connection before the server receives the malicious payload.

One of the only ways to truly prevent this worm from infecting a system is to make sure all applicable security patches are applied, a practice all too uncommon in the industry.

These patches can be found at the following locations.
Windows NT 4.0:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=30833

Windows 2000 Professional, Server and Advanced Server:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=30800

Another good means by which to proactively protect your systems is through OS and application hardening. Operating System (OS) hardening simply pertains to disabling services that are not needed for the system to perform its basic functions. As an example take a default installation of Windows 2000 Server. Some services, such as

SNMP will be enabled and started automatically during boot up. Most times these services are not needed and should be disabled. Otherwise your system can be unnecessarily exposed. The same actions can be taken with applications. In this case the IIS web server has many additional features installed and enabled by default, one of which is the Indexing service. With minimal configuration changes one can make their web server, or other application much more secure. Detailed hardening guidelines for both IIS and Windows 2000 (among other things) can be purchased from SANS.

It is also good security practice to routinely conduct vulnerability assessments against ones own systems. Vulnerability assessments are a proactive process in which the system administrator or local security officer will simulate real attacks against a system to determine where that system's exposures are. This offers a means to keep current with patches and helps ensure that the overall risks are minimized. There are many good tools freely available to conduct such assessments, some of which exclusively look for Code Red vulnerabilities

7. Similar worms

On the 4th of August 2001, *Code Red II* appeared. Although it used the same injection vector, it had a completely different payload. It pseudo-randomly chose targets on the same or different subnets as the infected machines according to a fixed probability distribution, favoring targets on its own subnet more often than not. Additionally, it used the pattern of repeating 'X' characters instead of 'N' characters to overflow the buffer. Code Red II used the same buffer overflow to compromise systems but had a much different payload. This variant was more deadly, it required more than a reboot to clean the system, and instead of defacing web sites and launching denial of service attacks it installed a remote backdoor program. There was also greater care taken with the random subnet generation routine which increased the spread of infection.

eEye believed that the worm originated in Makati City, Philippines, the same origin as the VBS/Loveletter (aka "ILOVEYOU") worm.

References

https://en.wikipedia.org/wiki/Code_Red_(computer_worm)
https://www.researchgate.net/publication/220269622_Code-Red_a_case_study_on_the_spread_and_victims_of_an_Internet_worm
https://www.sans.org/security-resources/malwarefaq/code-red
https://en.wikipedia.org/wiki/Heap_overflow
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf
https://bufferoverflows.net/use-after-free-vulnerability-uaf/
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf
https://blog.exodusintel.com/2019/03/20/cve-2019-5786-analysis-and-exploitation/
https://resources.infosecinstitute.com/topic/heap-overflow-vulnerability-and-heap-internals-explained/
https://blog.rapid7.com/2019/06/12/heap-overflow-exploitation-on-windows-10-explained/
https://github.com/LauraWartschinski/overflow_with_joy
https://stackoverflow.com/questions/47607333/illegal-instruction-when-trying-to-get-shell-from-a-simple-stackoverflow
https://stackoverflow.com/questions/2500362/running-32-bit-assembly-code-on-a-64-bit-linux-64-bit-processor-explain-the
https://github.com/npapernot/buffer-overflow-attack
https://seedsecuritylabs.org/lab_env.html
https://resources.infosecinstitute.com/topic/heap-overflow-vulnerability-and-heap-internals-explained/
https://blog.rapid7.com/2019/06/12/heap-overflow-exploitation-on-windows-10-explained/
https://blog.exodusintel.com/2019/03/20/cve-2019-5786-analysis-and-exploitation/
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf
https://bufferoverflows.net/use-after-free-vulnerability-uaf/
https://sploitfun.wordpress.com/2015/06/07/off-by-one-vulnerability-stack-based-2/
https://nixhacker.com/exploiting-off-by-one-buffer-overflow/
https://www.exploit-db.com/docs/english/28478-linux-off-by-one-vulnerabilities.pdf
https://ctf-wiki.github.io/ctf-wiki/pwn/linux/glibc-heap/off_by_one/
https://csl.com.co/en/off-by-one-explained/
http://www.cis.syr.edu/~wedu/seed/Book/book_sample_buffer.pdf
https://www.thegeekstuff.com/2013/06/buffer-overflow/
https://blog.rapid7.com/2019/02/19/stack-based-buffer-overflow-attacks-what-you-need-to-know/
https://github.com/Subangkar/Buffer-Overflow-Attack-Seedlab
https://dhavalkapil.com/blogs/Shellcode-Injection/

https://dhavalkapil.com/blogs/Buffer-Overflow-Exploit/#:~:text=Buffer%20overflow%20vulnerability,of%20it's%20alloted%20memory%20space.
https://nixhacker.com/exploiting-off-by-one-buffer-overflow/
https://csl.com.co/en/off-by-one-explained/
https://vulncat.fortify.com/en/detail?id=desc.internal.cpp.buffer_overflow_off_by_one
http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf
https://www.geeksforgeeks.org/heap-overflow-stack-overflow/
http://www.cis.syr.edu/~wedu/seed/Book/book_sample_buffer.pdf
https://github.com/yehiahesham/Buffer-Overflow-Attack
https://github.com/npapernot/buffer-overflow-attack
https://github.com/RihaMaheshwari/Buffer-Overflow-Exploit
https://github.com/shashijangra22/Buffer-Overflow-Attack
https://github.com/hackutk/overflow-example