Global Information Assurance Certification Paper

Code Red: A Field Study of a Worm in the Wild

L. Christopher Paul
21 June, 2001

Advanced Incident Handling and Hacker Exploits
SANS 2001 Baltimore
GCIH Practical Assignment
Version 1.5b

Option #2
Document an exploit, vulnerability or malicious program

© SANS Institute 2000 - 2005 Author retains full rights.

Introduction

We are interrupting the regularly scheduled SANS practical to bring a special report on the recent outbreak of the Code Red worm that has infested over 250,000 Windows computers running the IIS web server. While the original infection vector began on Friday the 13th [1] with some initial infestations, it was after the original report was written describing the worm's behavior was published, a second variant of the work appears to have been released that corrected a issue with the original version.

Acknowledgments

While I have been playing around with Code Red in a lab environment and confirming for myself much of what has already been written, I would like to state at the beginning of this Practical that this work would have completely impossible had not the entire security community banded together in sharing information about the recent outbreak of this worm. The work done by other plays heavily in this material and where others have been quoted as saying, "If I have seen further, it is because I have stood on the backs of giants," I can truly say that if it had not been for the giants, I may not have seen at all.

Please see the *Resources* section at the end of this paper for a list of the individuals and organizations that have enabled me to approach this topic. If I have failed to mention anyone it is simply an oversight and completely due to the rapidity with which some of this information has been and is being developed.

[1] There are isolated reports that some people have been tracking this worm since the 11th.

Exploit Details

Name	Code Red
Variants	CRv1 and CRv2
Operating System	Windows NT 4 (All services packs) with IIS installed. Windows 2000 (All Service packs)
Protocols/ Services	HTTP/ IIS Index Server (NT), IIS Indexing Service (WIN2k)
Description	This exploit takes advantage of a flaw in the Microsoft IIS program that permits the attacker to execute carefully crafted code on the target server. Once it executes on the server, it either propagates itself to 100 other servers or attacks a static IP address that was previously assigned to www.whitehouse .gov. This activity is depent upon the day of the month.

Protocol Description

The protocol that this worm utilizes is the standard web protocol (HTTP; *RFC 2068*, *RFC 2616*) to exploit vulnerable servers. In addition it takes advantage of the ISAPI extensions that Microsoft has built into their IIS Web Server.

These extensions provide additional functionality to the server; in this case they permit a web server to index local and remote document and return links to those documents in response to search queries. These features are implemented by making dynamically loadable libraries (dlls) available and by providing a hook in the web server to call those libraries based on the extension of the file requested from the server.

The traditional usage of the http protocol is that a client connects to the server on TCP port 80 and requests a specific document by name. If the document is available, a header is sent back to the client describing the type of file (html, text, jpg graphic file, etc.), possibly the file size and other information. The file is then sent back to the client and the connection is torn down. (There are also persistent http connections, but those are out of the scope of this document.)

How the Exploit Works

As previously mentioned, the IIS server that ships with NT 4.0 and Windows 2000 provides added functionality in the form of an Index Server (NT) or Indexing Service (WIN2k).

When the IIS receives a request for a file ending in .idq (Internet Data Query) or .ida (Internet Data Administration) it calls the idq.dll library with the data gathered from the client's http request. Unfortunately, as explained in a Security Alert released on June 18th, 2001 by eEye Digital Security[2] and confirmed by Microsoft[3], the IIS server did not perform proper validation on

[2] http://www.eeye.com/html/Research/Advisories/AD20010618.html
[3] http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-033.asp

the request submitted to the web server. This lack of validation permits a user to submit a carefully crafted request to the web server that overflows the buffer allotted for the variable space and to execute program instructions on the server.

The Code Red worm takes advantage of this flaw and utilizes the web server to search out other web servers with the same bug and reproduce itself. It also has a time-based trigger so that on a certain day, instead of propagating itself, every running copy of the worm begins to attack a single Internet site.

After attacking this one site, the worm goes dormant for a period of time until it then enters it reproductive stage again and the process repeats.

Below we will take a look at the various aspects of this worm and analyze its behavior.

Buffer Overflows

In order to gain access to a server in the first place, the worm must somehow execute instruction on that server. The Code Red worm does this by taking advantage of a buffer overflow vulnerability in the Indexing Server that comes as part of the Microsoft IIS Web Server.

Buffer overflows are the most common vulnerability in use at the moment to exploit information systems. They can be used locally to gain greater privileges or remotely to by-pass services and gain access to the underlying operating system.

These vulnerabilities occur when the programmer fails to validate data received from outside of the program. In C, one of the more vulnerable languages to buffer overflows, all variables must be defined by type (e.g. an integer number, a large integer number, a single character, a string of character) and size (the exact number of bytes that will be set aside for that variable to store its data), and are allocated a specific portion of memory prior to their being used. Once these are defined, C takes very little care in how it handles them. In order to provide a flexible and powerful language, it lets the programmer handle those details afterwards. This of course is where the danger lies.

A buffer overflow occurs when an program stores too much data in variable's location in memory. If the data only exceeds its allotted space by a byte or two, this could only overwrite other variables with incorrect data which would cause anything from incorrect computations being performed to the application crashing.

If enough data is stuffed into a location, the data will not only overflow the space allocated for that variable, but the space allocated for all the variables. When this happens, the data is also stored in the memory locations where the program instructions reside. The stack is a contiguous area of memory where these variable are located and the process of breaking out of that area is known as 'Smashing the Stack.'

If the data that is written into the executable portion of memory is carefully crafted, it can actually cause the program to start executing the data instead of returning and executing the original

program.

The Code Red worm does precisely this by taking advantage of the vulnerability described. A lengthy string is sent to the web server with an extension (.ida) that causes the IIS server to pass the data to the idq.dll. The idq.dll library does not perform proper bounds checking on the data that is passed to it and this permits the worm to smash the stack and overrun the executable area. Once the worm's 'data' is being executed, the worm takes control and the IIS program and it performs as follows.

Reproduction

Once installed on the server, the worm checks the day of the month on that server and if it is less than the 20th, the worm proceeds to launch 99 separate application threads to search out other computers to infect. Due to the fact that Version 1 of this worm used a static number to seed the random number generator, there was a very small segment of hosts that had been infected by this worm. Version 2 of the worm had this *flaw* corrected and when it was launched sometime early on the 19th of July, the number of infected computers began to rise exponentially.

Random number generation

Contrary to what report-reading middle management often believes, computers have traditionally be very poor at generating truly random numbers. They very much like to come up with the same results for the same operation, every time.

To get around this, computers use what are called pseudo-random numbers. A function is called with a seed value that started the generation process. After that the randomization routine will return a sequence of numbers that may appear random, but are sequential based on the initial seed value.

If the same seed is used time and again, the same sequence of *random* numbers will be returned.

Denial of Service

If the date is greater then the 20th and less than the 28th, the worm spawns 100 threads that launch themselves at 198.137.240.91, port 80. This address was formerly the address for www.whitehouse.gov. If those threads connect, they proceed to take part in a denial of service attack against that host by bombarding it with data. After those threads complete this task, they then sleep for 16777216 (10,000,000 hex) milliseconds (a little more then 4 hours, 40 minutes.)

Wake up. Rinse. Repeat.

Web Defacement (Version 1 only)

As part of the reproduction stage, there was a 100th thread created that checked what the default language of the Windows installation was. If it was determined that this was English, the thread would hook into the web server's TCP socket and intercept web page requests.

If the default page was not English, the 100th thread would proceed to behave just as the previous 99. Note that version 2 of the worm did not deface the web site and the 100th thread behaved as the others.

Once hooked into the TCP socket, the worm would, without altering any of the web server's files, change the web page to:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html ;
charset=english">
<title>HELLO!</title>
</head>

<bady>
<hr size=5>
<font color="red">
<p align="center">Welcome to http:// 'www.worm.com !
<br>
<br>Hacked By Chinese!
</font>
</hr>
</bady>
</html>
```

Reformatted for clarity

It would continue to deliver this message to all visitors for the following 10 hours, after which time the server would return the proper page for each request.

Dormancy

If the date is greater than the 27th, then the worm becomes dormant waiting for the 1st of the month to roll around and resume its previous activity.

Lysine Deficiency[4]

Before the worm starts reproducing or attacking 198.137.240.91, it checks for the existence of a file in the root directory of the C: drive. C:\notworm. If this file exists, the worm takes no further action.

It should be noted that while this has been labeled a Lysine Deficiency (LD), it is in fact the opposite, or an Anti-Lysine Deficiency (ALD). Were the worm to check for a condition such as the existence of a file and fail, become dormant or not take action if it didn't find it, that would be a true LD.

The worm checks for the existence of this file by attempting to open it. If it is found it is never closed and as a result, the file can not be deleted while the worm in running.

[4] For information about IT systems and Lysine Deficiencies, see http://www.rootkit.com/papers/Lysinedeficiencies.txt

Description of Variants

CR v.2

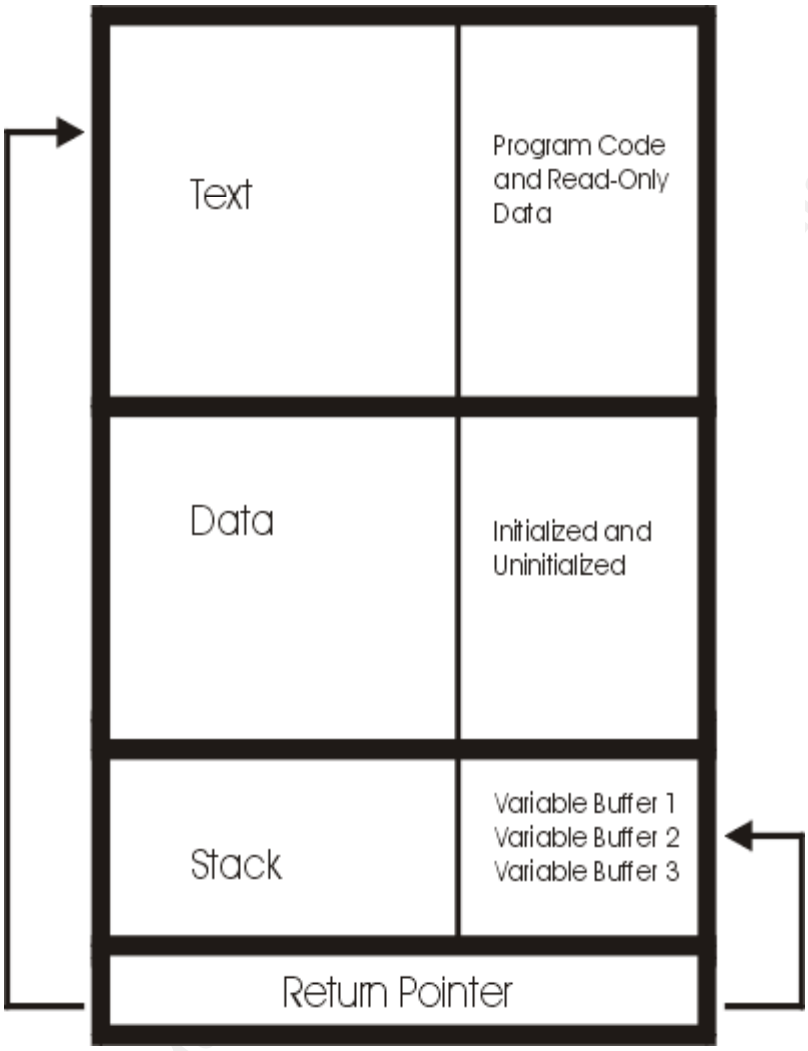
Version 2 of the Code Red worm differs from the original in two rather important ways. The first is that it no longer defaces the web site. This change prevents users and system administrators without properly configured Intrusion Detection Systems from immediately recognizing that the worm had attacked their site.

Secondly, the in Version 2 the worm's statically seeded random number generation routine was changed to one that relied upon the system clock. This change permitted the worm to attack a wider range of targets and it is believed that it eventually found residence on almost 300,000 unique servers – quite possibly every unpatched IIS server that was connected to the Internet at the time.

The change in the worm's behavior required that only 13 bytes be modified. Much of this functionality appears to have already been built into, but not implemented in CR v.1. This could mean that the original release was simply a test; but that has not, and may never be confirmed.

Diagrams

Anatomy of a Buffer Overflow



In a properly functioning program, the Return Pointer will return to the Text section of the application as indicated on the left side of the diagram. When the buffer has been overrun and the Return Pointer overwritten, the application can be forced to execute data contained in a Variable Buffer that resides within the Stack.

How to Use the Exploit

While the full original exploit that was used to launch this worm is not presently available, it is possible to deduce the manner in which it was launched. Included in this practical is a sample of proof of concept code that was used to attack this vulnerability in the Japanese language version of the IIS server. While the call to the web server and the Shell Code will have differed, the actual code was in all likelihood, very similar. This would have been the 'rocket' that launched the worm in the first place.

While the the worm is entirely memory resident and does not write itself to disk, samples of the 'warhead' portion of the worm were obtained by using 'netcat' program written by *Hobbit*[5].

This was accomplished by setting netcat to listen on port 80 and redirecting the output to a file.

```
nc -l -p 80 > captured_worm_file
```

It would also be possible to re-launch this worm in the same manner in which it was captured. By taking the captured file and piping it through the netcat program directed at an already identified vulnerable server, it would be possible to set this worm in action again.

To accomplish this from a UNIX platform the user would simply have to type:

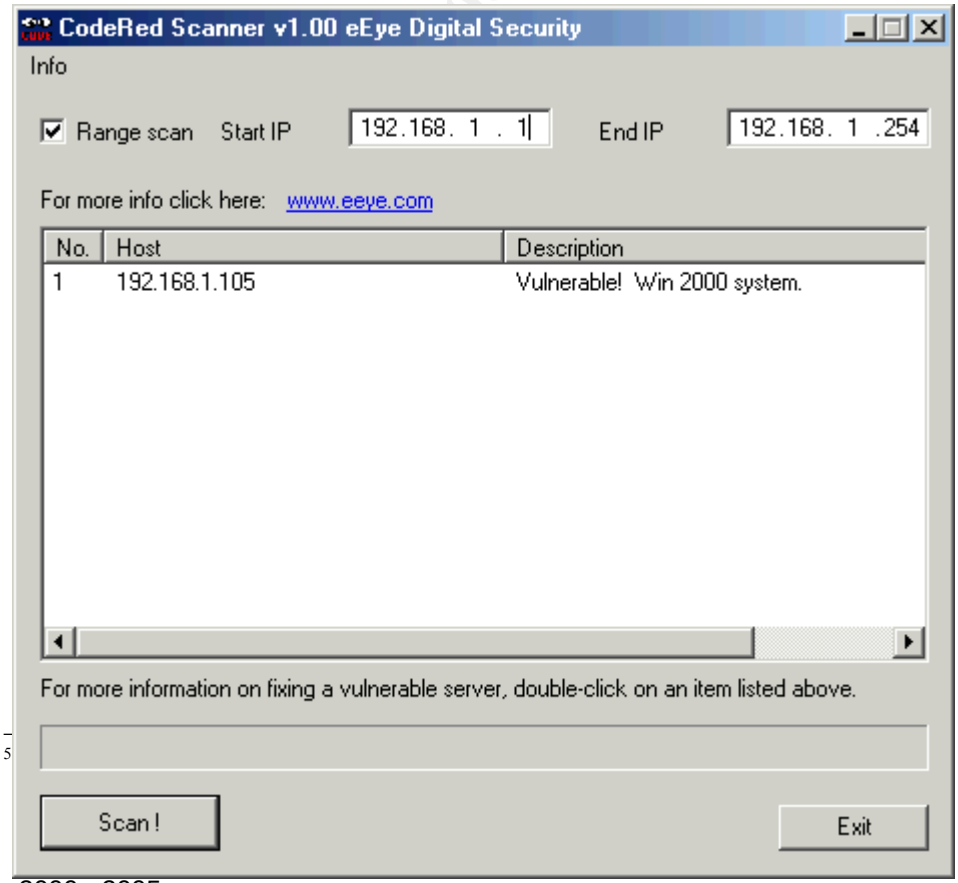
```
cat captured_worm_file | nc victim 80
```

Re-Targeting

While the current invocation of the worm is hard-coded to attack what was www.whitehouse.gov (198.137.240.91) it would be entirely possible for someone with an editor to re-target the worm at a separate IP address and launch the worm again. In a similar manner, it would also be possible to change the timing of the attack so that less time for response was available, change the number of threads that were created, etc.

Eligible targets could be identified by running the Code Red Scanner available at: http://www.eeye.com/html/Research/Tools/codered.html

This tool will scan up to a Class 'C' network for vulnerable servers and report them back. The user need only type in the address range and press the 'Scan!' button.



Side-effects

A number of security bulletins and reports have been released since CRv.2 was released indicating that many other devices with web management tools are also being affected by this worm. Among these are some Cisco devices[6,7] which use the IIS server and there are reports of HP's JetDirect cards locking up when hit by the worm.

Users have reported that the worm had an effect of a Denial of Service attack on these types of equipment.

Signature of the Attack

The following output is from an Apache web server that was probed numerous times in the last few days for the vulnerability.

```
192.168.192.210 - - [20/Jul/2001:08:33:49 -0400] "GET /default.ida?NNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090
%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u
531b%u53ff%u0078%u0000%u00=a  HTTP/1.0" 404 282
```

If a site is not utilizing the Indexing Service, the '.ida?' portion of the signature can used by IDSs to log attempted exploits. These signatures for some of the more popular IDSs can be found at: http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids552&view=signatures

It would also be recommended that the .idq vulnerability be added to the IDS at the same time, That can be found at:
http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids553&view=signatures

How to Protect Against It

System Patches

When this vulnerability was originally detected, Microsoft was contacted and made a patch available at the same time that the Security Bulletin was released.

These patches may be found at:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=30833 (NT)
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=30800 (Win2k)

---

[6] http://www.cisco.com/warp/public/707/cisco-code-red-worm-pub.shtml
[7] http://www.cisco.com/warp/public/707/CBOS-multiple.shtml

Removing script mappings

It is also possible to prevent the .ida and .idq extensions from calling the flawed idq.dll library. This can be done for IIS 5.0 manually:
http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/prodtechnol/iis/tips/iis5chk.asp

Or IIS 4.0:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/security/tools/iischk.asp

Or with for Windows 2000, Microsoft's IIS Security Tool can be used:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/security/tools/tools.asp

Persons who select this method of dealing with this vulnerability should be aware that it Microsoft's stated position that future patches to various system components may reset these mappings to their default setting and thus make the server vulnerable to this exploit. It is recommended that this measure be taken **in addition**, not in place of, applying the appropriate patch.

Firewall Screening

While this methodology would not have prevented an IIS server from being infected by the Code Red worm, egress port filtering would have prevented the server from propagating the worm. There is little if any reason for a dedicated web server to be surfing the Internet. Web servers do not need to get directions from Mapquest, place orders at Amazon.com or Barnes & Noble or even order moo shu pork from the local Chinese take-out.

Firewall or border router rules that prevent servers from originating any unnecessary connections to the Internet will prevent them from being used to exploit other computers on the network. Had the originally infected computers been unable to make out-going web connections, the virus would have had a much hard time spreading.

In addition to being a good neighbor, in this instance it would have also prevented the computer from broadcasting to other computers that it was vulnerable to this exploit. Had this exploit been easier to take advantage of, the secondary damage caused by this worm could have been much worse as infected computers were identified and commandeered by waiting script-kiddies.

The Lysine Method

While adding the file C:\notworm to a system will prevent the worm from taking root, this is not a reasonable or rational way of dealing with this threat. While it may be a 'quick-fix' solution to enable the server to remain functioning while patches are applied, this is not considered industry best practice and can not be recommended.

Third-Party Software

eEye Digital Security, the group performed the first research on this vulnerability and which led the effort to dissect and analyze the Code Red worm has a product that acts as an application firewall and isolates suspicious web traffic from the IIS server. SecureIIS[8] analyzes incoming

traffic for anomalous patterns such as buffer overruns and prevents these requests from reaching the IIS server.

Source Code

Rather than include over 75 pages of source code, the disassembled virus and analysis done by eEye, etc. I have included them as separate files included in the .zip file that contains this Practical. The actual captured worm contained in a separate password protected .zip file to prevent any potential contamination. The password to that file is the word 'infected'.

Proof of Concept Code

Shortly after the vulnerability was reported, a proof of concept code was written. Fortunately or unfortunately due to the offsets involved in the internal Microsoft code, this code is designed to exploit the Japanese version of the OS which, alas, I was unable to test.

It is included for the sake of completeness and can also be found at:
http://www.securityfocus.com/data/vulnerabilities/exploits/iis5idq_exp.txt

Disassembled Code (Code Red v.1)

This code and the accompanying analysis is available at
http://www.eeye.com/html/advisories/codered.zip

Captured Virus

As described above, this copy of the worm was captured off of the Internet by setting up netcat to listen on port 80 and redirecting the output to a file.

This is contained the file entitled:
CodeRedWorm.Captured

Resources

RFC 2068:
http://www.rfc-editor.org/cgi-bin/rfcdoctype.pl?loc=RFC&letsgo=2068&type=ftp

RFC 2616:
http://www.rfc-editor.org/cgi-bin/rfcdoctype.pl?loc=RFC&letsgo=2616&type=ftp

IIS Index Server Description:

[8] http://www.eeye.com/html/Products/SecureIIS/index.html

http://www.windowsitlibrary.com/Content/405/18/toc.html

eEye Digital Security's announcement regarding the vulnerability:
http://www.eeye.com/html/Research/Advisories/AD20010618.html

Microsoft's announcement regarding the vulnerability:
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-033.asp

Lysine Deficiency:
http://www.rootkit.com/papers/Lysinedeficiencies.txt
http://www.chiroweb.com/find/tellmeabout/lysine.html
Also see: "*Jurassic Park*" by Michael Crichton and movies of the same name.

Proof of Concept Code:
http://www.securityfocus.com/data/vulnerabilities/exploits/isapi-dos2.c
http://www.securityfocus.com/data/vulnerabilities/exploits/iis5idq_exp.txt

eEye's SecureIIS
http://www.eeye.com/html/Products/SecureIIS/index.html

Netcat page at L0pht Heavy Industries:
http://www.l0pht.com/~weld/netcat/

Smashing the Stack for Fun and Profit
http://www.codetalker.com/whitepapers/other/p49-14.html

"Here be Giants"

The folks at eEye who spent many hours disassembling, pouring over and analyzing this thing to keep the IT community informed:
Marc Maiffret
Ryan Permeh

Ken Eichman at Chemical Abstracts who seemed to be on one of the busiest Class 'B' networks and correlated the data that helped track the total number of presumed infections.

Alfred Hunger and the folks at Security Focus for the BugTraq and Incidents mailing lists that have provided the security community with a forum to deal with these sorts of issues as they arise.

Stuart Staniford at Silicon Defense for analyzing the infection vectors and insisting that the second worm was out there.

I am sure that I have missed other who have been posting to the various mailing lists this past week with critical information and answering questions. It is completely unintentional.