SecurityFocus

DeepSight™ Threat Management System
Incident Analysis

IP Protocol 11 (NVP) Backdoor Tool

Version 1: June 1, 2002, 14:00 GMT

Analysts: Jason V. Miller, Sean Hittel, Dan Hanson, Jensenne Roculan, Mario Van Velzen

Executive Summary

The SecurityFocus Threat Analyst Team has obtained a binary copy of a remote control utility that masquerades its communication channel as Network Voice Protocol (NVP) data. An analysis of this tool has revealed several of its features, including remote command execution, and a distributed denial of service (DDoS) agent. Denial of service capabilities include resource exhaustion via TCP SYN flood attacks, and bandwidth consumption via ICMP and both direct and reflective/amplified UDP attacks.

The NVP was first implemented in December of 1973 in a small set of educational and commercial institutions. It was designed to allow real-time interactive voice communication over early networks.

Action Items

The SecurityFocus Threat Analyst Team recommends that:

- Best practices for firewall configuration are followed, including filtering all unnecessary protocols and services.
- Host-based intrusion detection systems or file integrity checking utilities be used to track changes to important system files.
- Hosts are kept up to date with the latest vendor-supplied security patches.

Urgency
Medium

Ease of Exploit
Not Applicable

Affected Systems
UNIX systems, including Linux

Technical Description

The IP Protocol 11 (NVP) Backdoor Tool is a utility that receives commands through a protocol designed to masquerade as NVP (Network Voice Protocol) traffic. It is designed to give a remote attacker the ability to control a machine through non-standard communication channels, and is able to take advantage of permissive firewalls that allow IP Protocol 11 packets to pass through unfiltered.

All communications are encoded with a custom-encoding algorithm. In the event that a communication packet contains commands for direct execution by the infected host, this encoding system prevents an eavesdropper from easily obtaining the commands in plaintext as they travel across intermediary networks. Additionally, the communications protocol is entirely stateless, thereby allowing an attacker to mask his identity by spoofing the source address of his communication.

All data sent and received from this remote control utility is contained within IP packets with the 8-bit protocol field set to `0x0Bh` (11). The SecurityFocus Threat Analyst Team believes that this communication technique was designed to avoid filtering from improperly configured firewalls and evade detection by intrusion detection systems (IDS).

Once executed, the backdoor tool opens a RAW socket to listen for incoming data marked as IP Protocol `0x0Bh` (11) in the IP header. Upon receiving incoming data, the parent process will decode the incoming data and will key on the second byte of the decoded payload in order to determine the requested course-of-action. If appropriate, the parent will then fork() a child process to complete the requested command, allowing the parent process to continue listening for further communications. The parent/child design used by this utility allows an attacker to maintain control of the machine in the event that a child process dies, or stops responding. Additionally, several of the DoS functions that are used by the child processes within this program will continue iterating indefinitely until the process is killed.

In the event that the attacker requires an actual response, he is able to instruct the utility to respond to a specific IP address by passing it in an encoded packet with the `0x02` / Initialization command selected in the command byte of the decoded payload. In order to obfuscate the attacker's location when responding to this address, the utility will, at the option of the attacker, respond to multiple random hosts in addition to the host specified by the attacker.

As analyzed, the server recognizes twelve (12) distinct commands passed in an 8-bit field in the second byte of the decoded payload. Details regarding these commands are detailed below:

`0x01h` Query server for status information
This command instructs the server to generate a response indicating the child process PID, if any, as well as the command number that the child process is currently executing. It may also report the list of random IP addresses that are being used for responses, as well as additional information about the infected host. Indications from the initial analysis show that the destination address is either randomly chosen, or used from the list generated by the `0x02h` command.

`0x02h` Initialization and attacker IP adjustment
This command will perform several actions. First, the infected host's IP address (determined by the destination address in the IP header) is stored in memory for later reference. Additionally, an IP address is specified within the decoded payload (as bytes 4 to 7, inclusive) that the server will use as the destination address for all subsequent responses. There is a special option within this command that instructs the server to create an array of ten IP addresses, of which all but one randomly chosen

entry (containing the IP address specified within decoded payload) will include randomly generated IP addresses.

The random number generation used by the IP Protocol 11 backdoor does not appear to be accomplished via standard GNU calls to srandom() or random(). In GNU C, random() uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return random numbers. Similarly, srandom() sets its argument as the seed for a new sequence of pseudo-random integers to be returned by random(). The random number generator used in the backdoor, on the other hand, appears to be based on a seeded engine that uses a dynamic look-up table, bit shifting, and other basic mathematic operations. Although this random number generation routine is similar to srandom() and random(), the SecurityFocus Threat Analyst Team was unable to reproduce a similar random number generation algorithm using GNU implementations of srandom() and random().

This feature allows the attacker to obfuscate the IP address that he is using to listen for responses by forcing the server to send out multiple responses to random IP addresses, only one of which will actually be destined for the location specified by the attacker.

`0x03h` Execute specified commands via `/bin/csh`, and respond with output
This command instructs the server to fork() a child process and execute the supplied commands (encoded within the packet) via `/bin/csh`. Output from this command is redirected to a temporary file, "`/tmp/.hj237349`", and after execution has completed, this file is opened and the contents of it are sent as a response. Indications from the initial analysis show that the destination address is either randomly chosen, or used from the list generated by the `0x02h` command. The file containing the output is then removed from the system via an unlink() call.

`0x04h` UDP Flooder Using DNS Reflection Attacks
This command instructs the server to fork() a child process, and initial analysis suggests that the child will attempt to utilize an internal list of DNS servers as intermediary hosts in a DNS Reflection attack against a user-specified target. It appears that a small delay is initiated after each packet is sent out. Nearly all fields in the IP header and UDP header are randomly generated, and filtering or identifying the packets responsible for this attack based on header information alone is very difficult; typically, the only consistent data within the network layer protocol header is the UDP protocol identifier, `0x11h` (17). The transport layer protocol header is similarly varied, with only the source port 53 (DNS) remaining constant.

`0x05h` UDP or ICMP Attack
This command instructs the server to fork() a child process, and initial analysis suggests that the child will flood specified IP addresses with either UDP or ICMP flood attacks. The attacker specifies the type of attack in the command packet, either UDP or ICMP. The ICMP packets generated by this attack consist of type 8, code 0, or ECHO_REQUEST packets, and the UDP datagrams contain a destination port specified by the attacker. Packets generated by this command contain spoofed source addresses, and initial analysis suggests that they are a combination of user-specified and randomly generated addresses.

`0x06h` Open password-protected portshell on TCP port 23281
This command instructs the server to fork() a child process and listen for a TCP connection on port 23281. Upon connecting, it issues a single call to recv(), and checks for the ASCII string "`SeNiF`" followed by `0x10h` or `0x13h` before spawning an instance of `/bin/sh` and binding the standard file descriptors to the open socket. It should be noted that due to the fact that there is only one call to recv(), the entire password must be present in the infected host's receive buffer when the recv() call

stops blocking. Thus, under normal circumstances, this password cannot simply be sent interactively with a keystroke-by-keystroke protocol, such as the default communications method in most telnet clients.

`0x07h` Execute specified commands via `/bin/csh`

This command instructs the server to fork() a child process and execute the supplied commands (encoded within the packet) via `/bin/csh`. Output from this command is discarded.

`0x08h` Signal child process, if any, with `SIG_KILL`

This command instructs the server to signal the child process, if any, with `SIG_KILL`, thus causing it to terminate. The child process PID is typically stored in a global variable when forked, allowing this command to terminate a hung process. Additionally, most commands check for an active child process before following through with forking, and will abort such an action if a child process is already active.

`0x09h` UDP Flooder Using DNS Reflection Attacks

This command instructs the server to fork() a child process, and initial analysis suggests that the child will attempt to utilize an internal list of DNS servers as intermediary hosts in a DNS Reflection attack against a user specified target. This command is nearly identical to the `0x04h` command, though it appears that `0x09h` allows for a delay in the flood after `n` user-specified packets, whereas `0x04h` initiates a delay after each packet. Nearly all fields in the IP header and UDP header are randomly generated, and filtering or identifying the packets responsible for this attack based on header information alone is very difficult; typically, the only consistent data within the network layer protocol header is the UDP protocol identifier, `0x11h` (17). The transport layer protocol header is similarly varied, with only the source port 53 (DNS) remaining constant.

`0x0Ah` TCP SYN Flooder

This command instructs the server to fork() a child process. Initial analysis suggests that the child process will engage in a continuous TCP SYN flood attack against the specified target host. It appears that a small delay is initiated after each packet is sent out.

`0x0Bh` TCP SYN Flooder

This command instructs the server to fork() a child process. Initial analysis suggests that the child process will engage in a continuous TCP SYN flood attack against the specified target host. This command is nearly identical to the `0x0Ah` command, though it appears that `0x0Bh` allows for a delay in the flood after `n` user-specified packets, whereas `0x0Ah` initiates a delay after each packet.

`0x0Ch` UDP Flooder Using DNS Reflection Attacks

This command instructs the server to fork() a child process, and initial analysis suggests that the child will attempt to utilize attacker specified IP addresses as intermediary hosts in a DNS Reflection attack against a user specified target. This command is very similar to the `0x04h` and `0x09h` commands, though the `0x0Ch` command allows the attacker to specify a list of IP addresses to use as intermediaries instead of having the server obtain the addresses from its internal list. Nearly all fields in the IP header and UDP header are randomly generated, and filtering or identifying the packets responsible for this attack based on header information alone is very difficult; typically, the only consistent data within the network layer protocol header is the UDP protocol identifier, `0x11h` (17). The transport layer protocol header is similarly varied, with only the source port 53 (DNS) remaining constant.

Corroboration

The SecurityFocus Threat Analyst Team wishes to thank the Honeynet Project, for giving access to this binary to the public. The utility was used to exercise control over a compromised host after an actual attack, indicating that this utility is currently in use by members of the blackhat community.

Item Descriptions

File Names

This utility was originally downloaded from a compromised Web server as "`foo`", and eventually ended up as "`/usr/bin/mingetty`" on the compromised system. However, it should be noted that none of these filenames are hard-coded anywhere and, therefore, could be changed easily.

MD5Sum for "`foo`", which was eventually renamed to "`/usr/bin/mingetty`":
`1d726de4f7fe7e580c8fad4b3e4703f6`

Port Numbers Involved

At the attacker's discretion, a password-protected portshell may be opened on TCP port 23281. All other client-to-server and server-to-client communications are performed through IP protocol 11 (NVP).

TCP and UDP datagrams used in DoS attacks typically have user-specified or randomly generated source and destination ports, with the exception of the DNS Reflection attacks, which have a destination port of 53 (DNS).

Packet Traces

A sample packet used in client-to-server communication is included below. Note that the communication uses IP protocol 11, reserved for the NVP (Network Voice Protocol). During communications, the IP header consistently contains no options, and a 0 value for type of service. The identification number appears to be randomly generated.

```
Attacker -> Target:  ip-proto-11 402 (ttl 237, id 27788, len 422)
0x0000    4500 01a6 6c8c 0000 ed0b 892b 6804 0b7e  E...l......+h..~
0x0010    ac10 b702 0200 1730 482a eea0 f910 273e  .......0H*....'>
0x0020    556c 839a b1c8 dff6 0d24 3b52 6980 97ae  Ul.......$;Ri...
0x0030    c5dc f30a 2138 4f66 7d94 abc2 d9f0 071e  ....!8Of}.......
0x0040    354c 637a 91a8 bfd6 ed04 1b32 4960 778e  5Lcz.......2I`w.
0x0050    a5bc d3ea 0118 2f46 5d74 8ba2 b9d0 e7fe  ....../F]t......
0x0060    152c 435a 7188 9fb6 cde4 fb12 2940 576e  .,CZq.......)@Wn
0x0070    859c b3ca e1f8 0f26 3d54 6b82 99b0 c7de  .......&=Tk.....
0x0080    f50c 233a 5168 7f96 adc4 dbf2 0920 374e  ..#:Qh........7N
0x0090    657c 93aa c1d8 ef06 1d34 4b62 7990 cbcd  e|.......4Kby...
0x00a0    e3b9 fc26 4261 9496 ac82 ebf0 0d2c 435a  ...&Ba.......,CZ
0x00b0    7188 ff01 17ed 2830 461c 1719 2f05 263d  q.....(0F.../.&=
0x00c0    546b f70e 253c 536a 8198 afc6 ddf4 0a21  Tk..%<Sj.......!
0x00d0    384f 1a1a 3006 1d34 4b62 019c bad9 f007  8O..0..4Kb......
0x00e0    1e35 4c63 7a91 ea01 182f 466d 849b b2c9  .5Lcz..../Fm....
0x00f0    e0f7 3047 5e75 8ba1 b7cd e40b 2239 5067  ..0G^u......"9Pg
0x0100    7e95 abc1 d7ed 132a 4158 738a a1b8 3904  ~......*AXs...9.
0x0110    2140 4b4d 6339 0606 1cf2 0920 374e 93aa  !@KMc9......7N..
0x0120    c1d8 afb1 c79d 05f8 1332 a9ab c197 0dd8  .........2......
0x0130    f514 0f11 27fd 142b 4259 d0d2 e8be f901  ....'..+BY......
0x0140    17ed 041b 3249 6077 8ea5 1c1e 340a 2138  ....2I`w....4.!8
```

```
0x0150    4f66 7d94 abc2 c10b 2c4b 82e7 0726 3e55  Of}.....,K...&>U
0x0160    192b 747c 9268 869d b4cb 6e70 865c d972  .+t|.h....np.\.r
0x0170    8fae c6dd f40b 2239 50a7 c5dc f30a 3963  ......"9P.....9c
0x0180    81a0 bed5 ec03 3a9f bfde a1a3 b98f e175  ......:........u
0x0190    92b1 e00a 2847 5e75 8ce3 0118 2f46 759f  ....(G^u..../Fu.
0x01a0    bddc f208 1e34                           .....4
```

Description of Vulnerability

This method of communication will take place after the compromise of a target host. The particular incident that led to the discovery of this tool was the exploit of the WU-FTP File Globbing Heap Corruption Vulnerability on a vulnerable host.

Attack Data

In the attack data provided by the Honeynet Project, this utility was downloaded as "`foo`" from a compromised Web server, ultimately ending up as "`/usr/bin/mingetty`". The machine was compromised through the WU-FTP File Globbing Heap Corruption Vulnerability, and a bash script was executed on the machine to perform the download and installation of the IP Protocol 11 (NVP) Backdoor Tool.

Mitigating Strategies

The SecurityFocus Threat Analyst Team recommends that all users ensure that they are following best practice firewall configuration methodologies, which follow a "that which is not explicitly allowed is denied" logic for packet approval. Unless required, all non-standard protocols should be dropped at the perimeter.

IDS Updates

The following three Snort signatures have been created in order to alert on IP protocol 11 data, and both client-to-server and server-to-client communications relating to this utility:

```
alert ip any any -> any any (msg:"Suspicious Traffic - IP Protocol 11 NVP";
ip_proto: 11; classtype:misc-activity; rev:1;)

alert ip any any -> $HOME_NET any (msg:"Possible IP Protocol 11 Remote Access
Tool Client to Server"; content:"|02|"; ip_proto: 11; offset: 0; depth: 1;
classtype:misc-activity; rev:1;)

alert ip $HOME_NET any -> any any (msg:"Possible IP Protocol 11 Remote Access
Tool Server to Client"; content:"|03|"; ip_proto: 11; tos: 0; offset: 0;
depth: 1; classtype:misc-activity; rev:1;)
```

Resources

The Honeynet Project's Reverse Challenge
http://www.honeynet.org/reverse/

RFC 741, Specifications for the Network Voice Protocol (NVP)
ftp://ftp.isi.edu/in-notes/rfc741.txt

WU-FTP File Globbing Heap Corruption Vulnerability
http://online.securityfocus.com/bid/3581

CERT Incident Note IN-2000-04, Denial of Service Attacks using Nameservers
http://www.cert.org/incident_notes/IN-2000-04.html

Glossary

If you are unfamiliar with any term this report uses, please visit the SecurityFocus glossary at http://www.securityfocus.com/glossary for more details on information security terminology.

Contact Information

Corporate Headquarters

400 S. El Camino Real, 3rd Floor
San Mateo, CA 94402
U.S.A.
1-866-577-6300 Toll-free
1-650-548-0228 Fax
tms@securityfocus.com

Canadian Office

100-4th Avenue S.W., Suite 710
Calgary, AB, T2P 3N2
Canada
1-403-213-3939 Main
1-403-233-9179 Fax
tms@securityfocus.com

About SecurityFocus

SecurityFocus™ is a leading provider of enterprise security threat management systems. SecurityFocus provides global early warning of cyber attacks, customized and comprehensive threat alerts, and countermeasures to prevent attacks before they occur. As a result, SecurityFocus customers can mitigate risk, manage threats, and ensure business continuity. The company also licenses the world's largest, most complete vulnerability database, hosts the most popular security community, Bugtraq™, and publishes original security content on its Web site at www.securityfocus.com.

DeepSight Conditions: The SecurityFocus™ DeepSight™ Threat Management System provides Incident Alert and Analysis Reports, as well as Daily, Weekly, and Monthly Summary Reports. These reports draw on IDS log data contributed to the SecurityFocus Incidents Database by DeepSight Analyzer™ members. Members submit this log data to the Incidents Database voluntarily and often anonymously. While SecurityFocus Threat Analysts make every effort to inspect this data for validity, SecurityFocus does not guarantee the accuracy of submitted data. SecurityFocus uses the aggregated log information to detect trends and provides it to customers AS IS. Should you have questions, please contact tms@securityfocus.com.