[http://www.sans.org/infosecFAQ/covertchannels/covert_shells.htm]

Covert Shells
J. Christian Smith
November 12, 2000

Introduction

In modern information theory, multilevel systems by definition require inter-level communication, which in turn requires implicit or explicit security mechanisms between levels to mediate the communication process. The potential for a covert communication channel then is in piggybacking on an authorized communication channel between parts of the multilevel system. While the use of covert communication channels can be traced back to the days of clay tablets and papyri, computer networks are a logical extension forward of a multilevel system, one with a multiplicity of communication channels and authentication streams, which by being legitimized are susceptible to illegitimate uses.

A covert channel may be defined as any communication channel that can be exploited by a process to transfer information in a manner that violates a system's security policy. As a corollary of this, some form of concealment is usually required in order to maintain control of the channel once exploited. In this day and age, of e-mail and the world-wide-web, covert channel vulnerabilities are obviously unavoidable while attempting to maintain any form of communication with the outside world. This level of risk, however, may be acceptably low so long as no processes (trojan horses) exist to exploit them [1]. The insertion of a trojan horse is a necessary prerequisite to establishing a covert channel.

Covert channels can be developed in a variety of ways. In the most blatant sense, encrypting data basically creates a communications channel, one that operates outside the framework of the transport mechanisms that carry it. However, while concealed, it can't really be defined as covert because not only does one know that a message is being transmitted, but one knows *exactly* where it is. The most effective covert channel occurs in a datastream that looks like non-covert network traffic, serving a credible purpose.

A prime category of covert channel communication may be referred to as embedded channels. For example, the variable length data field of the ICMP protocol is largely extraneous to the actual purpose of the control message transmitted, so much so that firewalls, IDS, and humans rarely look beyond the headers, where the real meaning is actually carried by header flags. Therefore, some exploits use the data field to carry a communications channel, which is covert only to the extent that no one's looking, a strategy of confidence. The drawback is that it is a little obvious to use a "data" channel to hide data. This is akin to a hacker showing up to his day job in an "I'm a hacker" tee shirt.

More sophisticated implementations of embedded channel communications utilize header fields in protocols in the TCP/IP suite. Since portions of the headers of several core TCP/IP protocols, for example, 6 bits in the TCP header and 8 bits in IGMP, were left reserved for future implementation, very few security systems, at least initially, checked for data inside those fields. This makes them an excellent covert channel, because they are not intended for data transfer at all, and less likely to be examined. However, once one knows that this potential channel exists, its days of vulnerability are numbered. Although hiding data inside a header is a type of misdirection, in essence, this strategy has the same drawback as the simple embedding, by using a "reserved" channel to hide a channel. The analogy for this scenario would be a hacker showing up to his day job in a suit and tie.

Drawing on work from cryptography and steganography, subliminal channels are one way to get around the disadvantage of hiding in plain sight, by using legitimate communication to create the appearance of the absence of covert communications. As an example, here is the prisoners' problem:

> "Consider two prisoners in separate cells who want to exchange messages, but must do so through the warden, who demands full view of the messages (that is, no encryption). A covert channel enables the prisoners to exchange secret information through messages that appear to be innocuous. A covert channel requires prior agreement on the part of the prisoners. For example if an odd length word corresponds to "1" and an even length word corresponds to "0", then the previous sentence contains the subliminal message "101011010011" [2].

Here, the pre-establishment of a legitimate communication channel, and the lack of obvious artifice involved, serve to deflect investigation into the actual content of messages carried. The strategy employed here is one of disguise. Returning to the analogy, our hacker is now sending someone else in to his job, to collect his paycheck.

The key to the importance of covert channel communications in the toolkit of the hacker is simple: don't get caught. A system may be exploitable one time, all the data residing on it can be stolen, and it can be crashed or wiped. Yet, the attack is immediate, and potentially detectable and traceable. To safely "own" a remote system, however, it must be exploited, cleaned of evidence of the exploit, and made to serve hidden purposes, providing not just current data but potentially valuable information in the future. Due to the command-driven nature of most Unix systems, in order for those purposes to be flexible and limitless, rather than automated and scripted, covert shell control is required.

What follows is a review of available covert shell tools. Although this may not be a comprehensive list, the significant landmarks are included. Most of the tools were originally presented in underground "black hat" publications, and should be

considered malware. Yet they were found "in cages" not "in the wild," and may reflect what *black hats* want us to know, rather than what they have actually used. In the words of van Hauser from The Hacker's Choice (THC), the tools "are not elite of course, otherwise we wouldn't release them ;-)" [3].

The type of implementation this paper focuses on basically consists of a backdoor trojan server piece and a client that provides shell access over a covert channel to the server. The tools are presented chronologically in an attempt to illuminate the evolution of the ideas involved. To the extent that information is available, the history and source of each tool will be characterized, followed by an analysis of the actual technology employed. All tools were successfully built and employed in a test environment on Linux, Solaris, or NT systems, and their actions were observed with packet analyzers.

Tools

Loki

Loki was originally presented in August 1996 in the underground magazine Phrack, one of the oldest and most well-known hack-zines on the net, whose original mission was stated to "include articles on telcom (phreaking/hacking), anarchy (guns and death & destruction) or kracking" [4]. Loki appears to be the first generally available implementation of a covert shell, although the author disclaimed credit for pioneering the concept. In his introduction to Project Loki, route, also known as daemon9, stated that "many firewalls and networks consider ping traffic to be benign and will allow it to pass through, unmolested... use of ping traffic can open up covert channels through the networks in which it is allowed" [5]. The idea was to use the data field in ICMP type 0 (Echo Reply) and type 8 (Echo Request) packets for a bidirectional Unix shell client, an example of an embedded covert channel. While the data field in these legitimate "ping" packets may serve as an integrity check, and sometimes carries timing information, its implementation is optional, and it is rarely examined by routers, hosts, or humans.

The actual implementation of the idea, in source code, was presented in a follow-up article in September 1997 [6]. At this time, the idea was extended to include not only a ping channel, but also the option of using the data field in the UDP protocol as a covert channel, specifically masquerading as DNS name lookup traffic. Also, on-the-fly swapping between the two protocols was implemented for Linux, useful for added misdirection. UDP, like ICMP, is a connectionless protocol, and is therefore less likely to be seen on-system as an active connection (e.g., via netstat), although trojans that listen to the ports may be visible. Since both symmetric and asymmetric key exchange and blowfish encryption were available in Loki at build time, there was clearly a realization that covert was no guarantee, and the confidentiality of the commands embedded in the packets certainly reduced the risk of detection.

Shortly after the original Loki concept paper, but before the second, Craig Rowland discussed covert channels at some length, and presented covert_tcp, which uses a low-rate header-embedded covert channel for file transfer. At the time, he indicated that the ICMP tunneling concept was being publicly explored in underground resources and had prompted the release of his paper. Providing evidence that this kind of tool had been available in privileged circles for some time, he went on to say that "the ideas represented here are largely based on concepts heard a while ago about ICMP 'Telnet' like programs and similar utilities supposedly in use by some of the darker forces in government" [7].

Although daemon9 also claims that this vulnerability had been known and exploited for years, kudos go to Loki from just about every other covert shell implementation obtainable, begging the question of what kind of code actually existed before Loki. Several years later, Simple Nomad credits Loki as the origin of key facets of TFN ("Tribal Flood Network"), a well known modern distributed denial of service (DDOS) attack tool, which employed encrypted echo-reply packets for a control channel [8]. Given that there is even a "butt-plug" module for BO2K (Back Orifice 2000) that allows remote control over an ICMP tunnel, clearly the ICMP covert channel concept has become mainstream in the hacker community.

Daemonshell-UDP

THC's van Hauser asserts in the first release of Unix Hacking Tools (UHT), that the code presented, although dating from March 1998, is the previous year's model [3]. Included in the archive was icmptunnel, a number of relatively portable functions for setting up a covert channel between two locations, using the data field of ICMP Echo-Reply packets only, which are less likely to be blocked by firewalls than Echo-Request packets. Instead of a shell front end, a chat channel is presented. (Later, in 1999, Itunnel, a much more flexible and efficient ping reply talk tunnel was released by the Austrian group teso [9].) Also provided in UHT1, however, are two versions of daemonshell, written in perl. Daemonshell does nothing more than listen on an arbitrary port, password authenticate, and spawn a standard Unix shell. Client connection is made with a tool like netcat, and buffered data is transmitted in the data portion of TCP or UDP packets, depending on which version of the program is used. This is an easily implemented and flexible pathway, which requires no root compromise, which is needed for raw ICMP socket access in Loki. Unlike Loki, it attempts to be clandestine; out of the box it masquerades as vi. The UDP version provides an added element of covertness by being a more unexpected exploit channel, which can be used to take advantage of any underscrutinized, legitimate user-level UDP channel that passes a firewall (e.g., real audio). All this is provided in 44 lines of easy to hack perl code. Nonetheless, since no attempt is made to emulate legitimate traffic, and cleartext shell transactions are plainly visible in the packets, detection is far from impossible.

ICMP Backdoor

Not to be outdone, a few months after THC's release, the CodeZero team published an ICMP backdoor client and server in their Confidence Remains High (CRH), an online hack-zine with no original mission other than to provide "news, exploits, scanning, telco, and enough shit to make you wonder 'why did I ever pay cash for this?!'" [10]. The brief code contains icmpd.c and icmpc.c, a server and client, collectively referred to as ICMP backdoor [11]. Humorously, each code part contains exactly one easy to fix typo, perhaps a simple attempt to filter out script-kiddies. This is not a pinnacle of coding sophistication: the client is dumped into the shell environment of the detached daemon, and while command execution and primitive buffered I/O are possible, there are no terminal capabilities, so simple tasks like changing directories and running man pages are impossible. While ICMP backdoor has the advantage over Loki of using ping reply packets only, a side effect of the lack of padding is that it produces very short ping packets, and may not actually be seen as ICMP. Among software packet sniffers for example, snoop characterizes ICMP backdoor as unknown IP packets, while tcpdump sees ICMP. The author, BiT, makes no effort whatsoever to annotate or extol the virtues of the program.

007Shell

In a recent interview, s0ftpr0ject, distributors of 007shell had this to say:

> "'s0ftpj's guys are young and good looking' so they have to get away with mad professors, big-whooper-breasts girls and screaming mammas, but despite this they are always brain-connected with the net and always active to get ahead with their big passion seated in front of a monitor with a keyboard on the legs" [12].

S0ftpr0ject can perhaps be more accurately as an Italian-based security team drawn from the hacking underground. They publish the aperiodic hack-zine BFI (Butchered From Inside), "created to familiarize netsurfers to the linux world and to sensibilize them about the excellence in data protection and its transmission" [13]. Published entirely in Italian, Issue 4 from December 1998 included 007Shell, self-described as "Covert Shell Tunnelling," with "ShoutOuts" given to Loki and daemon9 [14]. In some ways it is an improvement on previous work: the distribution is split into a reusable tunneling library and a shell front end, and the same program serves as either client or server. Data is padded out to multiples of 64 bytes which virtually guarantees that it appears to be ping packets, and it also employs ping reply packets only. In the words if FuSyS, the author, "it simply is so common to let ICMP ECHO REPLY slip through firewalls and not to log it." It provides a full-featured shell environment. On the downside, root permission is required for ICMP raw socket access, the program makes no attempt to hide itself on the local system, and data transmission is in cleartext, making it easy to detect - when one knows it is there.

Rwwwshell

Another release by THC's van Hauser, Rwwwshell was slightly ahead of its time. While version 1.6 of the code dated from October 1998, all evidence points to May 1999 as the real date of its distribution, as a Proof-of-Concept program for the paper "Placing Backdoors through Firewalls" [15]. Building off of some of the strengths of daemonshell, it was also written in perl, making it largely portable and flexible. The key strength of rwwwshell lied in the paradigm shift it represented: here the slave/trojan side, made the initial connection to the master rather than the reverse, a strategy designed to circumvent stateful packet inspection devices and proxies that only allow incoming traffic from pre-existing connections.

In practice, the rwwwshell slave can be actively used real-time, or configured as a passive backdoor, to wake up and contact the pre-configured master as a timed event, via an outgoing HTTP Request. The master side then originates shell commands as HTTP Response packets, and output from commands return from the slave as cgi script HTTP GETs. Each command-response exchange is a new TCP connection initially incrementing from port 1171, and by default connecting to port 8080, thereby allowing non-root use and appearing to be a caching webserver exchange. Commands and responses are uuencoded to further obscure the true nature of the conversation. In the passive mode, since the slave side may wake up every day and fail connection attempts to the master, by the time the true remote control session actually takes place, it may not even appear to be a new or unusual conversation. In addition, the master server responds to conventional connection attempts with 404 error messages, adding to the appearance of legitimacy. The reliable transport of TCP packets also ensures data integrity.

The only obvious drawback to the methods employed in rwwwshell relate to the credibility of the web traffic, particularly when all traffic is revealed to be calls to /cgi-bin/order. Embedding in web traffic is highly likely to be a successful strategy for covert channel establishment. An added example from 1999 is the more well known release of httptunnel which encapsulates a bidirectional data path in base-64 encoded HTTP GET/PUT messages [16]; it could certainly be used as an element of covert shell establishment. The lack of a ready-made shell front end bundled with this and other tunneling packages, however, prevents their classification as covert shells for the purposes of the discussion here.

B0CK

In the presentation of the actual code for Loki, daemon9 indicated that implementations of programs similar to Loki already existed for UDP, TCP, and IGMP. For Christmas 1999, s0ftpr0ject released BFI number 7, and under the hacking column is listed a variation on the ICMP covert shell. Written entirely in Italian, B0CK uses IGMP multicast messages, and was designed to improve upon what the author refers to as defects in route's work in Phrack (Loki), and FuSyS's 007Shell. The author, vecna, asserts that the trickle of ICMP that is still permitted to cross border filters is more and more frequently logged and investigated, and that continuous incoming streams of unsolicited ping reply traffic are too

suspicious and risky, particularly when the source address is consistent and reflects the real, unspoofed, malicious source [17]. Presumably he views IGMP as less likely to be blocked.

B0CK raises the bar in covert shell sophistication by incorporating both embedded and subliminal channel communications. Each line of output, from the client or server side, is broken up into four byte sections; short sections are padded with zeros. The corresponding ASCII code of each character/byte is then used as the spoofed source address of an outgoing IGMP packet. Even without a translation this is easiest to see from an example in the text:

> "Quindi se noi mandiamo al server remoto il comando 'ls -la', con un igmplogger leggeremo in ricezione:
>
> Sep 23 14:18:27 TruZz0 -algad-: from 108.115.32.45 IGMP type 0 code 0
>
> Sep 23 14:18:27 TruZz0 -algad-: from 108.97.0.0 IGMP type 0 code 0
>
> Con:
>
> 108.115.32.45 108.97.0.0
>
> l s " " - l a <0 = invio>"

All communication data is basically transmitted in code, subliminally embedded as the source address field of the IP header. Since the source address no longer indicates the real location of the client, and the server still needs to know where responses go, B0CK provides the option to compile in or send an initial command to the server containing the client's real address.

There are several problems, however. While B0CK packets have the IP Protocol field set to IGMP, they contain no encapsulated IGMP header; the packet is padded with 124 bytes of zeros after the 20-byte IP header. Therefore, they show up as obsolete type 0 IGMP packets (with invalid checksums) and are easily recognized - when you know they're there. Also, there is diminished vulnerability now that more and more networks disallow spoofed packets from crossing their borders.

AckCmd

Windows 2000 is the culmination of a progression in Microsoft operating systems where each successive version since DOS was left behind restores more and more administrative control to the command-line interface. Although it can be argued whether VBS and KIX scriptability are more dangerous than "del /S *.*," nonetheless 2000 is a viable target for covert shell control. Written by Arne Vidstrom, AckCmd provides a remote command shell on Windows 2000 systems [18]. Although it uses TCP packets for transport, like traditional remote shells, there are a number of aspects that make the program particularly covert. Designed to exploit a weakness in many firewall and IDS rules, AckCmd communicates using only TCP ACK segments, rather than more typically examined TCP SYN packets. The data segment of each originating ACK contains buffered command line data, and elicits a TCP RESET from the remote side; the response is then presented in a new ACK back to the originator. While an extremely fast typist might produce an abnormally large rate of reset packets, nonetheless the traffic pattern looks like an already established (and torn down) connection and is unlikely to raise any flags. In addition, the choice of client side TCP port 80, and server side port 1054, increases the likelihood of successful firewall traversal, and reduces risk of detection. However, the packets are not properly formatted for HTTP, the program is plainly visible on the task list, and the command line transaction is in cleartext - easily observed, if one know it is there.

Conclusion

To the extent that evolution is visible in the historical chronology of the tools discussed, there is a progression in sophistication from simple encapsulation methods to more advanced masquerading and subliminal encoding. By implication, as soon as effective defenses were erected, simple modification of existing tools or extension of the ideas represented was sufficient to continue to circumvent them. While awareness of ping-embedded covert channels appears to have disseminated throughout the security community, there is a lack of discussion of the threat from related methods of covert communication, exemplified by TCP ACK and HTTP channels. All of the tools presented above, nonetheless, make initial assumptions out-of-the-box about id or sequence numbers (ICMP tunnels), or source or destination port numbers (UDP and TCP), or have predictable data content. This in turn provides excellent identifying signatures for detection systems.

Since covert shells are long term control tools rather than actual attack vectors, it is unlikely that they will arouse notice that the original attack failed to trigger, without an increased awareness of the types and strategies available, as provided here. This presentation should make it apparent that short of cutting off external traffic altogether, stopping covert communications at the border of a network is not going to happen. A more viable approach has to start from the inside, with the basics, preventing trojans and backdoors from becoming inserted in the first place. It bears repeating that this is only going to occur when users are properly educated about security policy and procedures, are provided useable tools for information assurance, and the security of the multilevel system that is a computer network is regularly audited for compliance.

References

1. Proctor, Norman E. and Neumann, Peter G. "Architectural Implications of Covert Channels." Proceedings of the Fifteenth National Computer Security Conference, pages 28-43, Baltimore, Maryland, October 13-16, 1992. URL: http://www.csl.sri.com/neumann/ncs92.html (November, 2000).

2. RSA Laboratories. "What are covert channels." Cryptography FAQ, Version 4.1, Section 7.15. URL: http://www.rsasecurity.com/rsalabs/faq/7-15.html (November, 2000).

3. van Hauser. "README." THC-UnixHackingTools #1, October 17, 1998. URL: http://thc.pimmel.com/releases.htm, http://thc.pimmel.com/files/thc/thc-uht1.tgz (November, 2000).

4. Taran King. "Introduction to Phrack Inc." Phrack Magazine, Volume One, Issue One, File 1 of 8, November 17, 1985. URL: http://phrack.infonexus.com/search.phtml?view&article=p1-1 (November, 2000).

5. daemon9 and alhambra. "Project Loki: ICMP Tunnelling." Phrack Magazine, Volume Seven, Issue 49, File 6 of 16, August 1996. URL: http://phrack.infonexus.com/search.phtml?view&article=p49-6 (November, 2000).

6. route (daemon9). "LOKI2 (the implementation)." Phrack Magazine, Volume Seven, Issue 51, Article 6 of 17, September 1, 1997. URL: http://phrack.infonexus.com/search.phtml?view&article=p51-6 (November, 2000).

7. Rowland, Craig H. "Covert Channels in the TCP/IP Protocol Suite," November 14 ,1996. URL: http://www.psionic.com/papers/covert/covert.tcp.txt (November, 2000).

8. Simple Nomad. "Strategies for Defeating Distributed Attacks," 1999. URL: http://packetstorm.securify.com/papers/contest/Simple_Nomad.doc (November, 2000).

9. edi and teso. "ICMP tunneling tool," July 10, 1999. URL: http://teso.scene.at/releases.php3, http://teso.scene.at/releases/itunnel-1_2.tar.gz (November, 2000).

10. Tetsu Khan. "Welcome To Issue 1 Of Confidence Remains High." Confidence Remains High, Issue 1, April 16, 1997. URL: http://www.hackersclub.com/km/magazines/crh/crh001.txt (November, 2000).

11. BiT. "ICMP backdoor client and server." Confidence Remains High, Issue 9, May 11, 1998. URL: http://www.hackersclub.com/km/magazines/crh/crh009.txt (November, 2000).

12. sil. "Q and A with Team S0ft Project." Anti Offline. June 20, 2000. URL: http://www.antioffline.com/s0ftpj.html (November, 2000).

13. SMaster and \sPIRIT\. "s0ftpr0ject 2000," April 21, 2000. URL: http://www.s0ftpj.org/en/site.html (November, 2000).

14. FuSyS. "Progetto Ninja." Butchered From Inside, Number 4, Year 1, December 1998. URL: http://www.s0ftpj.org/bfi/bfi4.tar.gz (November, 2000).

15. van Hauser. "Placing Backdoors Through Firewalls," v1.5, May 1999. URL: http://thc.pimmel.com/files/thc/fw-backd.htm (November, 2000).

16. Brinkhoff, Lars. "GNU httptunnel." August 31, 2000. URL: http://www.nocrew.org/software/httptunnel.html (November, 2000).

17. vecna. "B0CK." Butchered From Inside, Number 7, Year 2, December 25, 1999. URL: http://www.s0ftpj.org/bfi/online/bfi7/bfi07-12.html (November, 2000).

18. Vidstrom, Arne. "ACK Tunneling Trojans," 1999-2000. URL: http://www.ntsecurity.nu/papers/acktunneling

[http://www.sans.org/infosecFAQ/covertchannels/covert_shells.htm]