John Jenkinson

LevelTwo Advanced Incident Handling and Hacker Exploits
GCIH Practical Assignment
Version 1.4
Current as of December, 2000

Option 2 - Document an exploit, vulnerability or malicious program Locate and fully document an example of malicious code. When possible, the emphasis should be on the actual exploit and NOT the malicious code that implements the exploit. You may NOT select malicious code, software, or specific exploits that were covered in class, although variants are allowed. If you choose a variant, be certain to point out any differences between the code you are describing and the code that was described in your course. In addition, please avoid examples that are already posted as a practical or as part of the Information Security Reading Room without requesting permission in advance from giactc@sans.org.
Be certain to submit the malicious code itself with your documentation. Malicious code must be in a zip file that is password protected. Your write up must include the following:

Exploit Details:
CERT Advisory CA-2001-02 Multiple Vulnerabilities in BIND

VU#196945 – ISC BIND 8 buffer overflow in transaction signature (TSIG) handling code
CVE Name CVE-2001-0010
BIND versions affected: 8.2, 8.2-P1, 8.2.1, 8.2.2-P1, 8.2.2-P2, 8.2.2-P3, 8.2.2-P4, 8.2.2-P5, 8.2.2-P6, 8.2.2-P7, and all 8.2.3-betas.
This vulnerability is listed as severity CRITICAL and remotely exploitable with known exploits

VU#572183 – ISC BIND 4 contains buffer overflow in `nslookupComplain()`
CVE Name CVE-2001-0011

VU#868916 – ISC BIND 4 contains input validation error in `nslookupComplain()`
CVE Name CVE-2001-0013
These two vulnerabilities are related, have severity of SERIOUS, are also remotely exploitable with known exploits.
BIND versions affected: 4.9.3, 4.9.4, 4.9.5, 4.9.5-P1, 4.9.6, 4.9.7, possible earlier versions of BIND 4.9.x and BIND 4.9.

VU#325431 – Queries to ISC BIND servers may disclose environment variables
CVE Name CVE-2001-0012
This vulnerability has a listed severity of MODERATE, is remotely exploitable

with known vulnerabilities
BIND versions affected: 4.8, 4.8.3, 4.9.3, 4.9.4, 4.9.5, 4.9.5-P1, 4.9.6, 4.9.7, 8.1, 8.1.1, 8.1.2, 8.2, 8.2-P1, 8.2.1, 8.2.2-P1, 8.2.2-P2, 8.2.2-P3, 8.2.2-P4, 8.2.2-P5, 8.2.2-P6, 8.2.2-P7, possibly earlier versions of BIND 4.9.x and BIND 4.9.

The vulnerabilities are in versions of Internet Software Consortium's (ISC) Berkley Internet Name Domain (BIND) servers. Most vendors running BIND as an implementation of Domain Name System (DNS) base their offerings on ISC's BIND, thus the vendor list in the CERT advisory is long including Unix variants from vendors like IBM, Sun, SGI, HP and others. There have been several major versions of BIND on most systems that use BIND to implement DNS. BIND uses port 53 in both User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). Name queries usually are UDP based unless the answer packet is more than 512 bytes, in which case the request must use TCP. Zone transfers, which are used by servers to pull zone file information from outer servers, use TCP.
The best way to determine the version of bind you are running is to watch the console or output of the daemon facility of syslog when you start the daemon that implements BIND. An example daemon name is named. Some systems configure themselves to run the BIND daemon whether it is needed or not so you should check your systems for these vulnerabilities on every system. Just because you have not explicitly setup a BIND server does not mean that one is not implicitly running.
If you have a need to determine the version of BIND you are running without local access to the machine and/or a need to restart the BIND server daemon you can use:

```
nslookup –q=txt –class-CHAOS version.bind. 0
                              -or-
dig @<DNS server> version.bind chaos txt
```

with the understanding that the version reported may come from the config files and not from the BIND daemon itself.

The first three vulnerabilities were discovered by COVERT Labs at PGP Security, that advisory is at http://www.pgp.com/research/covert/advisories/047.asp
The VU#325431 vulnerability was discovered by Claudio Musmarra.

It is interesting to note the media coverage generated by the simultaneous warnings from CERT and COVERT Labs. Computerworld's story was headlined "Internet Security Hole called most serious yet". The article goes on to say the TSIG vulnerability is similar to the denial of service attack at Microsoft just after their recent DNS problems. While it is true that DNS servers are tempting targets due to their visibility on the Internet and it is a good strategy to keep these services at current versions, we should look at the details of the found vulnerabilities. Knowing the details should not slow down or influence the task of upgrading to a non-vulnerable version of BIND, but it is good to know the details.

Is this CERT advisory prelude to the most serious Internet security hole yet?

Protocol Description

DNS allows users to specify names for services they use for email, web browsing, file transfers, etc. DNS takes the name the user specifies like ibm.com and finds the IP address needed by the application to provide the user's requested service. DNS is a distributed database with a tree like structure with the root at the top. Each new level or subdomain has a relationship with the level above and beneath it. As the database is distributed it has some local control. That local control has the benefit of local management of subdomains appearing on the Internet, but it also has the drawback of an attack at the point of local control being felt on the Internet. DNS servers are good targets as they have to be known to the Internet to work on the Internet. We can hide a lot of our servers and services behind firewalls and routers, but DNS servers need to have DNS protocol ports known and somewhat open.

DNS is implemented as a client-server architecture. BIND is the server portion which listens on port 53 on both UDP and TCP protocols.

The client portion uses a resolver library called by applications with calls to routines like gethostbyname(). It is this resolver library that formulates and sends service requests to the DNS server, then interprets the result or error returned by the DNS server. DNS is not necessary for programs to use routines like gethostbyname(), the operating system can use a local host text file, use NIS (Network Information System), or some other means to return an IP address for a supplied name.

If the resolver library does use DNS, then the query is formulated according to protocols defined in Request for Comments (RFCs) and sent to the DNS server usually across the network. A list of these RFCs is given here.

As mentioned the client sends the request to the DNS server using UDP or TCP on well known port 53.

The DNS server receives the request(s) from the client, validates those requests, and returns the result or an error message. If the DNS server is implemented as BIND then the information needed to satisfy the request may come from a name cache, from BIND's local database, or BIND may need to ask another DNS server for the answer in a process called recursion. In the case of recursion, the server becomes the client of another DNS server.

A problem is how much information to give out on a request to DNS. If you are running an eBusiness you want any/everyone to be able to find your public webserver's IP address when they provide your eBusiness name. You probably don't want any/everyone to be able to find the IP address of your HR machine. When DNS was specified computers were not so plentiful as they are now and a site only had a small number, so such security issues were not a concern. It obviously is now so more security is part of DNS.

Another problem is redundancy. You would want more than one DNS server, thus these multiple servers need to communicate the changes and new information. This is done DNS server to DNS server via zone transfers. Those zone transfers need security as well as you typically do not want nodes to request a zone transfer that are not a secondary server for that zone or have a valid reason to request a zone transfer. With dynamic updates, sever records and other new features of DNS, security becomes a concern.

These and other security issues are added into BIND as BIND versions progressed from BIND V4 to BIND V8 now to BIND V9. In particular DNS Security Extensions (DNSSec) in RFC 2325 and RFC 2845 for Secret Key Transaction Authentication for DNS (TSIG).

Some security is provided in BIND's configuration files like /etc/named.boot or /etc/named.conf. Here you can place networks or nodes that can request zone transfers. You could also configure tcpwappers to restrict such zone transfers since they are TCP based. Problem with these approaches, IP addresses can be spoofed.
DNSSec provides a method of adding more security to zone transfers from server to server by adding a SIG record which is a signature over the resource records in that zone. Recall a signature is a message digest or hash encrypted with a private key.

TSIGs provide anti-spoofing and more security to transactions between client and server. Here we need to sign the entire request via request signature and we need to sign the reply via transaction signature (TSIG). As the whole request and reply are signed, the signature has to go into the additional data section of the reply record at the end of that data section. The TSIG exploit provides this additional data section. The exploit is triggered when this additional data is interpreted as a TSIG and no public key is available to decrypt this TSIG. DNSSec and TSIGs started to appear in BIND V8.2

Description of variants
Known variants of the exploits of these vulnerabilities at the time of this report include: bind.c, bind8x.c, bugtraq.c, tsl_bind.c, lion worm, adore worm, erkms toolkit.

The CERT advisory was issued January 29, 2001. On January 31, 2001 the first claimed exploit of the TSIG vulnerability was posted at Bugtraq. A February 1 post by Max Vision of Whitehats.com warned that the exploit was actually a disguised shell code trick to flood dns1.nai.com after sending a packet to the requested target DNS server. This trojan had the actual shellcode invoked by overflowing its own internal buffer to set a return address to the shellcode.

Next was bind8x.c. released about February 5, 2001 written by Ix and Lucysoft. The original post claimed the code would need some fixing to work. Since then the code has been fixed by Ian Goldberg and Jonathan Wilkins and Whitehats reports three known variants and it targets Linux in Intel exploiting the TSIG vulnerability

Last Stage of Delirium (LSD) released their exploit February 2001. This exploit uses the infoleak vulnerability as well as the TSIG vulnerability. Whitehats shows 4 variants and it also targets Linux on Intel.

The bugtraq.c exploit as released February 5, 2001. This too targets Linux on Intel but does not use the infoleak vulnerability (TSIG only) or appear to do any probing.

The tsl_bind.c exploit was out about February 2, 2001, authored by Gustavo Scotti and Thiago Zaninotti, it does do probing via the infoleak vulnerability.

The lion worm, perhaps derived from the Ramen worm, surfaced mid February. It attacks Linux on Intel currently but has the potential to be adapted to target other systems with the infoleak and TSIG vulnerabilities.

The adore worm, also known as the red worm, was reported about the first of April. It too currently targets Linux on Intel.

The erkms tool kit appeared mid February, 2001. It does probe for DNS servers at port 53 using TCP.

It did not take long for exploits of these BIND vulnerabilities to appear as postings in newsgroups and web sites which were quickly placed into use against sites on the Internet. I could find no exploits of the BIND V4 specific nslookupComplain vulnerabilities.

More detail on these exploits are in the **How to use the exploits** and **Signature of the attack** sections.

How the exploit works

Paraphrasing from the CERT and Covert Labs advisories:
ISC BIND 8 buffer overflow in transaction signature (TSIG) handling code
BIND 8 contains a buffer overflow in the implementation of TSIG for DNS security as defined in RFC 2845 that allows a remote attacker to execute arbitrary code. Since the overflow is in the initial processing of a DNS request it does not require an attacker to control an authoritative DNS server, as do the two vulnerabilities for BIND 4. In addition, the vulnerability is not dependent upon security configuration options and affects both recursive and non-recursive servers. During the processing of a transaction signature (TSIG), BIND 8 checks for the presence of TSIGs that fails to include a valid key. If such a TSIG is found, BIND skips normal processing of the request and jumps directly to code designed to send an error response. Because the error-handling code initializes variables differently than in normal processing, it invalidates the assumptions that later function calls make about the size of the request buffer.

Once these assumptions are invalidated, the code that adds a new (valid) signature to the responses may overflow the request buffer and overwrite adjacent memory on the stack or the heap. The mechanisms employed by BIND 8 make it susceptible to two potential methods of attack.
A stack based buffer overflow with two important qualifications: first, the number of bytes past the end of the buffer that the attacker can overwrite is limited in length, and second, that the values of those bytes are mostly fixed. On the x86 architecture the attacker can manipulate a sufficient number of bytes such that they can modify the saved frame pointer. Overwriting the least significant byte of the saved frame pointer can result in the execution of arbitrary code in certain predictable installations of BIND. The “infoleak” bug VU#325431 permits retrieval of stack frames from named which then allows calculations of the effect of the one byte overflow.
For heap buffer overflow, the attacker overwrites malloc’s internal variables if the operating system’s implementation of malloc stores those internal data structures in the allocated memory. When a DNS request is received, it is stored in the heap or on the stack, depending on the transport mechanism. For a UDP request it is read into a 513 byte buffer on the stack called “`u.buf`” by the function `datagram_read()`. For TCP the message is read by `stream_getlen()` into a 64k buffer called “`sp->sbuf`”, which is allocated from the heap for each socket. An interesting feature of BIND uses the incoming buffer of both transport

mechanisms to read the request from the network and then modifies it to create the appropriate response. Two key variables are maintained to track the usage of the buffer: one containing the actual length of the data in the buffer called "`msglen`" and a second variable "`buflen`" that tracks the remaining length free in the buffer.

When a DNS message is received, `msglen` is initialized to the length of the data received. For UDP this value comes from the `recvfrom()` call, while TCP gets this value from the length by the client. `buflen` is set to the size of the buffer used to read the message (512 bytes for UDP, 64k bytes for TCP).

Under normal circumstances, BIND appends the answer, authoritative, and additional records to the query. It then modifies the DNS header to reflect these changes and delivers the response. During this processing, `msglen` will reflect the length of the response as it is being built and `buflen` will track the remaining space in the buffer. Throughout this processing BIND assumes that `msglen` plus `buflen` will equal the original length of the buffer.

Upon receipt of a DNS message, it is processed as either a request or response based upon the query response flag set in the message header. If a request is received, BIND then determines whether it is a query, iquery, update, or notification. Beginning with BIND 8.2, prior to request processing, the additional section of the DNS message is examined for TSIG resource record. The function `ns_find_tsig()` is called to perform this functionality as well as enforcing a basic level of validity on the TSIG resource record. If a valid TSIG is identified but an appropriate security key cannot be found, an error is signaled and BIND bypasses the normal request processing. As a result, `msglen` and `buflen` remain close to their initial values instead of being set to their "working" values.

BIND processes the request as an error since TSIG was identified but an appropriate security key was not found. As part of the error generation, BIND reuses the request buffer and appends TSIG after the question section. At this point BIND assumes the size of the request is `msglen` plus `buflen`, which under normal circumstances, would be correct. However, in this special case, the request was never processes and `msglen` and `buflen` is almost twice the size of the original buffer. BIND then is willing to append a TSIG via `ns_sign()` beyond the limits of the buffer. Since a valid security key was not found, `ns_sign()` will only append a small number of bytes with limited values. As mentioned this makes the vulnerable BIND susceptible to two types of attack.

Combining this oversight with the way a compiler positions the stack variables in `datagram_read()`, it is possible to overwrite portions of the saved stack activation records in `datagram_read()` with certain fixed values. In this case, executing arbitrary code is possible under the x86 architecture by overwriting the saved frame pointer's least significant byte with zero resulting in the saved frame pointer pointing to the original DNS request in most cases. Predicting the effects of this one byte overwrite can be difficult as it depends on how BIND was started. However the "infoleak" bug allows retrieval of the stack activation record of `datagram_read()`. This information can then be used to calculate the exact number of bytes that will displace the frame pointer when the least significant byte of the saved frame pointer is overwritten by 0.

The second method of attack utilizes certain implementations of dynamic memory allocation. It is possible to overwrite malloc's boundary tags with predictable values, changing the standard library's notion of the length of the buffer following the buffer processed in the DNS request. Thus, the next set of boundry information is read from within a buffer that an attacker can control, allowing for a malicious pointer overwrite upon compaction.

This technique is applicable to malloc implementations that store linkage information in the actual allocated memory. COVERT knows these implementations to be exploitable:

IRIX libc, Linux glibc, and Solaris libc.

When combined with other buffer overflow exploitation techniques, an attacker can gain unauthorized privileged access to the system, allowing the execution of arbitrary code.
The arbitrary code would be code that the attacker would use in a buffer overflow exploit in any other buffer overflow exploit, not arbitrary arbitrary code.
This vulnerability may allow an attacker to execute code with the same privileges as the BIND server. Because BIND is typically run by a superuser account, the execution would occur with superuser privileges.

ISC BIND 4 contains buffer overflow in nslookupComplain()
BIND 4 contains a buffer overflow that can allow a remote attacker to execute arbitrary code. The overflow occurs when BIND reports an error while attempting to locate IP addresses for name servers. Exploitation of this vulnerability is restricted by the fact that the target name server is recursive and that the attacker has control of an authoritative DNS server.
The vulnerable buffer is a locally defined character array used to build an error message intended for syslog. Attackers attempting to exploit this vulnerability could do so by sending a specially formatted DNS query to affected BIND 4 servers. If properly constructed, this query could be used to disrupt the normal operation of the DNS server process, resulting in either denial of service or the execution of arbitrary code.
The vulnerability occurs within `nslookupComplain()` which is a static utility function used by `nslookup()`. Specifically a sprintf into a 999 byte stack buffer that occurs when BIND forms a message warning the administrator of an inconsistency or error resolving a Name Server (NS) record to an IP address.
When BIND encounters a query that it can not answer from its cache or zone files, it attempts to forward the query to a name server that is capable of resolving it or referring BIND to a more appropriate name server. When BIND forwards a query, it creates a `qinfo` structure to keep track of the request. It also creates this structure in order to track requests initiated by itself to find various linkage information. BIND can determine potential name servers to forward to by walking each label in the query in its database looking for stored NS records.
The purpose of the `nslookup()` function is to take a list of NS records and populate a `qinfo` structure with the corresponding IP addresses. BIND can then use those IP addresses as a list of name servers to forward or send the query.
The `nslookup()` function performs certain sanity checks on the information that it retrieves, eg NS records with an IP address of 0.0.0.0, which it will flag as an error. BIND then warns the administrator via syslog and moves to the next NS record. The function `nslookupComplain()` is called and it contains a stack overflow. To trigger this stack overflow, the attacker needs to get BIND to cache a NS record with a very large length and one of the problem conditions that invokes `nslookupComplain()`. This can be done by sending a query to a recursive name server asking it to resolve a large name that is under the authority of a malicious name server. That malicious name server then refers the request to another name server also with a large name and provides an additional record giving an invalid address for that name server.
The limitations of the character set allowed in domain names makes construction of a viable return address difficult. However, there is a potential for an attacker to make the name server return into memory that the attacker has forced the name server to allocate. Then the vulnerability is contingent upon the location of the heap and the amount of memory available as well as whether the operating system has a policy of lazy swap page allocation as opposed to eager reservation policy. COVERT has verified this is possible under Linux by growing the heap it sizes far exceeding the amount of memory and swap available. This is accomplished by utilizing specific patterns of memory allocations that maximize untouched memory.
The situation may be further complicated by overwriting of two other stack based buffers, `nsbuf` and `abuf`, which are read from within the same `sprintf` that overflows the stack based buffer.

This does not come into play if the value chosen to overwrite the saved return address does not utilize the terminating null byte of the string. It is worth noting that this behavior could make it easier for an attacker to exploit the problem under operating systems that implement `sprintf` such that overlapping copies are handled correctly.

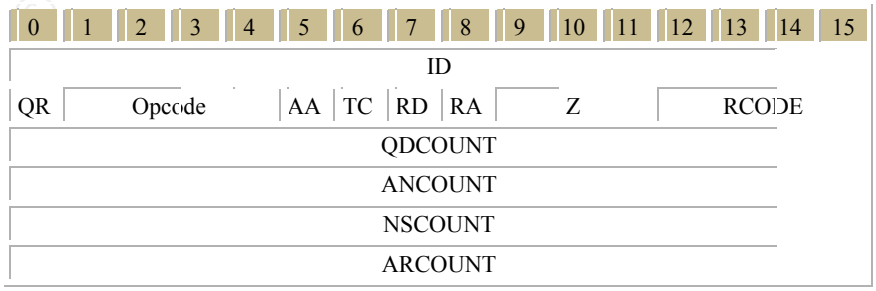
ISC BIND 4 contains input validation error in `nslookupComplain()`
BIND 4 contains a format string vulnerability that can allow a remote attacker to execute arbitrary code. This vulnerability also occurs when BIND reports an error while attempting to locate IP addresses for name servers, and thus has the same restrictions on exploitation as the buffer overflow. This vulnerability was fixed several versions prior to the current version of BIND 4 (4.9.5-P1) but is still present in certain Unix distributions.
The vulnerable buffer is a locally defined character array used to build an error message intended for syslog.
This vulnerability is in the same section of code as the previously described buffer overflow (VU#572183) and thus can be triggered in a similar fashion by using a malicious name server with authority for a crafted long name.
Attackers attempting to exploit this vulnerability could do so by sending a specially formatted DNS query to affected BIND 4 servers. If properly constructed, this query could be used to disrupt the normal operation of the DNS server process, resulting in the execution of arbitrary code.

Queries to ISC BIND servers may disclose environment variables
This vulnerability is an information leak in the query processing code of both BIND 4 and BIND 8 that allows a remote attacker to access the program stack, possibly exposing program and/or environment variables. This vulnerability is triggered by sending a specially formatted query to vulnerable BIND servers.
This vulnerability may allow attackers to read information from the program stack, possibly exposing environment variables. In addition, the information obtained by exploiting this vulnerability may aid in the development of exploits for VU#572183 and VU#868916.

From the whitehats site the snort IDS signature for this exploit is given as:
"|AB CD 09 80 00 00 00 01 00 00 00 00 00 00 01 00 01 20 20 20 20 02 61|"

Using O'Reilly's "DNS and BIND" by Paul Albitz and Cricket Lui and RFC 1035

The header contains the following fields:



where:

ID

A 16 bit identifier assigned by the program that generates any kind of query. This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries.

QR

A one bit field that specifies whether this message is a query (0), or a response (1).

OPCODE

A four bit field that specifies kind of query in this message. This value is set by the originator of a query and copied into the response. The values are:

0

a standard query (QUERY)

1

an inverse query (IQUERY)

2

a server status request (STATUS)

3-15

reserved for future use

AA

Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section.

Note that the contents of the answer section may have multiple owner names because of aliases. The AA bit corresponds to the name which matches the query name, or the first owner name in the answer section.

TC

TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.

RD

Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.

RA

Recursion Available - this be is set or cleared in a response, and denotes whether recursive query support is available in the name server.

Z

Reserved for future use. Must be zero in all queries and responses.

RCODE

Response code - this 4 bit field is set as part of responses. The values have the following interpretation:

0

No error condition

1

Format error - The name server was unable to interpret the query.

2

Server failure - The name server was unable to process this query due to a problem with the name server.

3

Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.

4

Not Implemented - The name server does not support the requested kind of query.

5

Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data.

6-15

Reserved for future use.

QDCOUNT

an unsigned 16 bit integer specifying the number of entries in the question section.

ANCOUNT

an unsigned 16 bit integer specifying the number of resource records in the answer section.

NSCOUNT

an unsigned 16 bit integer specifying the number of name server resource records in the authority records section.

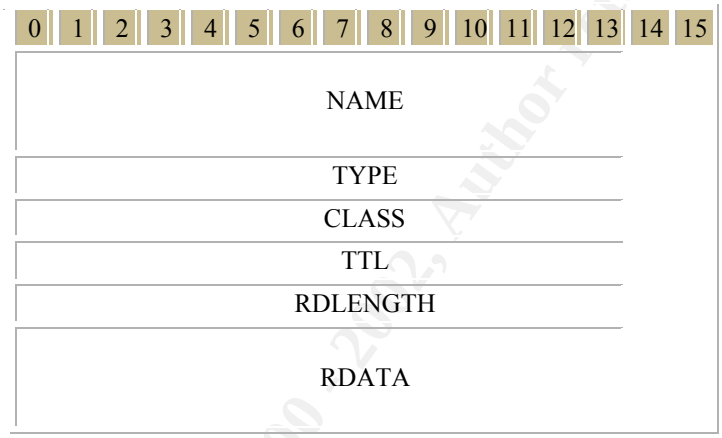
ARCOUNT

an unsigned 16 bit integer specifying the number of resource records in the additional records section.

It can be determined that this is indeed a inverse query with recursion requested. This is not normal as in inverse query (seeking the domain name that corresponds to a supplied IP address) would not be recursed. Albitz and Lui state that BIND 4 has the code for inverse query commented out by default and no code in Bind 8 to perform such query. We have an ANCOUNT of 1 so the rest of the packet should be interpreted as a resource record answer. Seems strange but again quoting from Albitz and Liu “In an inverse query, there is one answer in the query packet, and the question section is empty. The name server fills in the question.”

4.1.3. Resource record format

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:



where:

NAME
: A *domain-name* to which this resource record pertains.

TYPE
: two octets containing one of the RR type codes. This field specifies the meaning of the data in the RDATA field.

CLASS
: two octets which specify the class of the data in the RDATA field.

TTL
: a 32 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached.

RDLENGTH
: an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

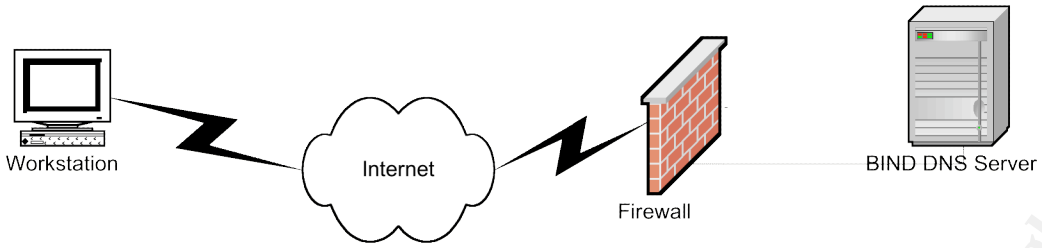
RDATA
: a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record. For example, the if the TYPE is A and the CLASS is IN, the RDATA field is a 4 octet ARPA Internet address.

So a domain name of null, type of 1, class of 1, time to live as 4 ASCII spaces, then RDLENGTH of 609 indicating an odd number of octets in the RDATA field. As the type is an A record and the class is a class IN the RDATA *should* be 4 octets of ARPA Internet address. Unsure why the packet is crafted this way, could be the RDLENGTH is enough to trigger the infoleak without the RDATA or it is necessary to be missing part of the packet. It should be noted that the signature could be changed by altering the DNS ID number.

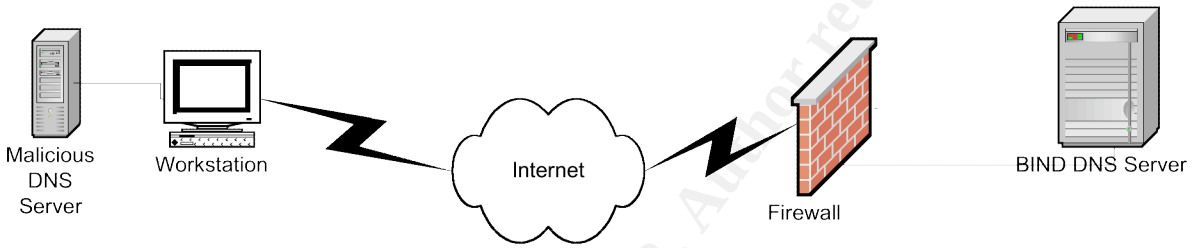
To summarize the advisories, BIND versions 8 prior to 8.2.3 and BIND versions 4 prior to 4.9.8 both have vulnerability that may disclose information about environmental variables and / or stack frames that may give an attacker more information to craft a buffer overflow. With or without the information gained by sending a vulnerable BIND server the specially formatted query, BIND Version 8 has a vulnerability in the TSIG handling code that may allow an attacker to craft a buffer overflow. BIND Version 4 has vulnerabilities in `nslookupComplain()`. Both buffer overflows occur in processing an error condition or composing an error message.

So we have buffer overflow vulnerabilities. What is a buffer overflow vulnerability? When computers were large in size but small in memory programmers had to (re)use memory more resourcefully. Stack pointer manipulations, using code as data and data as code, writing to buffers in creative ways, etc. were all tricks of the trade. A part of the code that had to be read and understood was the comments as they could give the real reason why some of the coding techniques were used. The 1988 Internet worm was a buffer overflow that used these techniques for other purposes. Since then several articles have made buffer overflows a well-practiced technique for exploits. This is due to the method of most attacks require, getting a service on another machine to execute code you have written and sent to it via ordinary input means. Picking a service on a server that has a large population makes the payoff of crafting the buffer overflow exploit quite large for the attacker. Compound the problem with the wealth of hacker-type newsgroups, bulletin boards, and web sites and one buffer overflow crafter has a many fold increase in buffer overflow attackers. But we can't just overflow any or all buffers to exploit programs. The buffer must be properly placed such that the overflow overwrites items we need (like a subroutine return address, a jump table, etc.) If we wish to have the result of the buffer overflow be the execution of our code then the code fragment we intend must be in memory somewhere or be placed into memory as part of the input the program accepts that causes the buffer overflow. With the Open Source movement, and tools designed for source code analysis, finding the conditions of buffers with no bounds checking and adjacent targets to overflow and write to like return addresses has gotten easier and more available than when source for programs and operating systems were not readily available to the general user community. Given the many buffer overflow vulnerabilities found and fixed in the past few years, the Open Source movement, programmer awareness, and program development tools designed to prevent buffer overflows we can hope these vulnerabilities are on the wane.

Diagram



For VU#325431 we just need a DNS query to a vulnerable BIND server. Once the infoleak bug is exploited we have the information needed to determine which vulnerability to exploit next and the hints required to craft the buffer overflow codes.



For VU#572183 or VU#868916 we need a malicious DNS server that the attacker has authority over or knows to be configured to cause the buffer overflow.

How to use the exploit

The CERT advisory last revised 4-Apr-2001 indicates VU#196945, VU#572183, and VU#868916 have not been publicly exploited. The vulnerability descriptions seem to indicate that buffer overflow exploits were found by analysis of the code. This proved the vulnerabilities were exploitable, but would require a lot of trial and error to get the code into the stack via the buffer overflow. Then the infoleak vulnerability showed that this information could be obtained to help craft the overflow.

Then there was the lion worm, the adore worm and the erkms toolkit. The lion worm uses the TSIG vulnerability while the adore worm uses the TSIG vulnerability as well as rpc.statd, LPRng, and wu-ftp. The erkms toolkit also uses the TSIG and VU#325431 vulnerabilities.

From the excellent analysis of the lion worm by Max Vision at whitehats.com which also details the 3 versions (thus far) and the political background of the worm and its lineage:

The lion worm scripts the attack profile as:

randb executable selects and runs against a random class B network

pscan then attempts to complete a 3-way TCP handshake on port 53 (BIND/DNS)

each response from a host causes an executable *bind* against the responding host

once the system is compromised, the rootkit (t0rn, sometimes crew.tgz) may be installed.

For our interest in BIND exploits it is the Linux ELF binary of *bind* from LSD (Last Stage of Delirium) released at their website February 2001 to note.

The adore worm (not to be confused with the adore loadable kernel module) is also known as red worm seems to be derived from the Ramen worm. It seems to not attack a selected address with each of the 4 potential vulnerabilities but uses some other pattern of attack. Though recently released (April 1) it appears to be Ramen modified to add the TSIG attack and targets RedHat Linux 6.

According to the CERT Incident Note on BIND vulnerabilities, the erkms toolkit has 4 components to attack sites with the BIND TSIG vulnerability. The shell script *r* calls *rscan* to port scan a class C network at TCP port 53 (BIND/DNS). For hosts that respond *rscan* then executes *m.o* which in turn executes *l* against the victim.

The tsl_bind.c exploit run against Red Hat Linux version 6.2 is documented very well in several papers at SANS.

It is indeed disturbing. Run the exploits against Linux on Intel, get a shell prompt as root on that machine.

Instead of duplicating that effort, let's run the infoleak partion of these exploits aginst some other UNIX variants. As these do not yet exploit vulnerable versions of BIND, we will cover the signature of the exploits in the next section.

Signature of the attack

If a site is running BIND Version 4 prior to 4.9.5-P1 and they do not upgrade to a version not vulnerable to VU#868916, then they need to monitor those BIND servers for buffer overflow signatures. Two of the buffer overflows have a query producing the overflow. Monitoring DNS queries for machine dependent buffer overflow "eggs" seems more labor intensive than upgrading BIND. The TSIG buffer overflow has a transaction signature craft the buffer overflow so it will be much harder to detect, but much easier to monitor as transaction signatures are not near the volume as DNS queries. If the system is capable of running accounting then monitoring accounting records for unusual activity from the account running the BIND daemon could indicate this attack. Given the sizes of the buffers that the buffer overflow exploits then traffic on port 53 of TCP and UDP protocol stacks can be a signature. Also the large name and invalid address in a query stream would be a signature for this attack.

Incident Note IN-2001-03 from CERT warns that vulnerabilities VU#325431 (disclosing of environmental variables) and VU#196945 (TSIG) are now actively being exploited particularly against Linux systems. The traffic pattern is what you would expect:

Use the malformed reverse query to identify a vulnerable BIND server, then use the TSIG vulnerability exploit if a potential vulnerable BIND server is found. The lion or 1ion

worm and the erkms toolkit are those mentioned in the Incident Note. The attack patterns differ slightly for the two in that lion uses a 3-way TCP handshake where erkms uses UDP after the TCP sequence to send the inverse query and then the shellcode via UDP port 53.

William Stearns of Dartmoth's ISTS (Institute for Security Technology Studies) has scripts to find evidence of lion, adore, and Ramen worms after infection.

It is anticipated that more platforms will be exploited given the success of the Linux worms.

Snort and IDS signatures for several related BIND vulnerabilities exits at the Whitehats web site. These include:

IDS482/named-infoleak-lsd

IDS489/named-exploit-tsig-lsd

IDS490/named-exploit-tsig-lucysoft

IDS491/named-exploit-tsig-tsig0wn

The tsl_bind.c run against AIX V4.3.3 with a vulnerable BIND V4.9.3 from IBM produces a named syslog entry of:

named[14114]: No root nameservers for class CHAOS

Turning on debug level one we see:

datagram from [aa.bb.cc.dd].3152, fd 7, len 30; now <date>
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
req: nlookup(version.bind) id 21527 type=16
req: missed 'version.bind' as '' (cname=0)
findns: No root nameservers for class CHAOS?
ns_req: answer -> [aa.bb.cc.ee].3152 fd=7 id=21527 Local
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
prime_cache: priming = 0
sysquery: send -> [aa.bb.cc.ee].53 dfd=8 nsid=6 id=0 retry=989883292
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)

RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_add)
ENTER(1:.sprout)
MSGF(LESS. sprouting left.)
ENTER(2:.sprout)
MSGF(MORE: sprouting to the right)
ENTER(3:.sprout)
MSGF(LESS. sprouting left.)
ENTER(4:.sprout)
MSGF(LESS. sprouting left.)
ENTER(5:.sprout)
MSGF(grounded. adding new node, setting h=true)
RET(5:.sprout)
MSGF(LESS. left branch has grown.)
MSGF(LESS: case 1.. bal restored implicitly)
RET(4:.sprout)
RET(3:.sprout)
RET(2:.sprout)
RET(1:.sprout)
RET(0:.tree_add)

datagram from [aa.bb.cc.dd].53, fd 7, len 449; now Mon May 14 15:34:49 2001
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)

13 root servers

Switching the same AIX V4.3.3 machine to Bind V8

ksh$ ./tsl_bind aa.bb.cc.dd
. ISC bind 8.2.2-x remote buffer-overflow for linux x86
. (c)2001 Tamandua Laboratories - www.axur.com.br
. (c)2001 Gustavo Scotti <scotti@axur.org>

. TCP listen port number 25000
. waiting for server response... 8.2.2-P5
. probing ebp... ebp is b4101700

*. waiting for connect_back shellcode response... * this ebp is not vulnerable. sorry!*

The AIX system's syslog shows no entries and the debug level 1 shows:

Debug level 1
Version = named 8.2.2-P5 Fri Mar 10 20:29:01 CST 2000
build@pebbles.austin.ibm.com:/build/obj/power/tcpip/usr/sbin/named8
configuration file = /etc/named.conf
datagram from [aa.bb.cc.dd].3169, fd 23, len 30
req: nlookup(version.bind) id 21658 type=16 class=3
req: missed 'version.bind' as '' (cname=0)
ns_req: answer -> [aa.bb.cc.dd].3169 fd=23 id=21658 size=63 rc=0
datagram from [aa.bb.cc.dd].3169, fd 23, len 465
FORMERR IQuery message length off

ns_req: answer -> [aa.bb.cc.dd].3169 fd=23 id=21658 size=718 rc=1

Now running the fixed bind8x.c

ksh$ ./fixed-bind8x aa.bb.cc.dd
[*] named 8.2.x (< 8.2.3-REL) remote root exploit by lucysoft, Ix
[*] fixed by ian@cypherpunks.ca and jwilkins@bitland.net

[*] attacking <node> (aa.bb.cc.dd)
[d] HEADER is 12 long
[d] infoleak_qry was 476 long
[*] iquery resp len = 719
[d] argevdisp1 = 080d7cd0, argevdisp2 = 00000000
[*] retrieved stack offset = 0
[d] evil_query(buff, 00000000)
[d] shellcode is 134 long
[d] olb = 0
[x] could not write our data in buffer (offset0=24, rroffsetidx=8)

[x] error sending tsig packet

On the BIND 4.9.3 server, nothing in the syslog and the debug level one shows:

datagram from [aa.bb.cc.dd].3161, fd 7, len 476; now <date>
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
FORMERR IQuery message length off
ns_req: answer -> [aa.bb.cc.dd].3161 fd=7 id=48879 Local
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)
RET(0:.tree_srch)
ENTER(0:.tree_srch)

RET(0:.tree_srch)

Again, now using a vulnerable BIND V8.2.2-P5

./fixed-bind8x <node>

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

[*] attacking <node> (aa.bb.cc.ee)
[d] HEADER is 12 long
[d] infoleak_qry was 476 long
[*] iquery resp len = 719
[d] argevdisp1 = 080d7cd0, argevdisp2 = f82ff22f
[*] retrieved stack offset = b4101720
[d] evil_query(buff, b4101720)
[d] shellcode is 134 long
[d] olb = 32
[x] could not write our data in buffer (offset0=62, rroffsetidx=7)

[x] error sending tsig packet

the level 1 debug output:

Version = named 8.2.2-P5 Fri Mar 10 20:29:01 CST 2000
build@pebbles.austin.ibm.com:/build/obj/power/tcpip/usr/sbin/named8
configuration file = /etc/named.conf
datagram from [aa.bb.cc.dd].3171, fd 23, len 476
FORMERR IQuery message length off

ns_req: answer -> [aa.bb.cc.dd].3171 fd=23 id=48879 size=719 rc=1

You should get similar results on other implementations of BIND. If you are in a situation where you can not upgrade the version of BIND, you should run the exploits against your BIND DNS servers to obtain the signature of the attacks and setup scripts to watch for those signatures.

We could also list the results after applying the APR to fix the AIX supplied BIND, but the vulnerabilities are reported by IBM to be fixed.

How to protect against it

Upgrade to a non-vulnerable version of BIND.

Imagine the upgrade is not possible. The vulnerabilities in this alert go against only BIND as in implementation of the DNS service. So for all DNS servers, we take away all those not running BIND. We now have a smaller number of DNS servers. Now take away those not running one of the vulnerable versions of BIND, a smaller number still. For those servers running on operating systems that do not store linkage information in allocated memory for malloc () we can also remove them from the possible population of vulnerable systems. Take away again systems that don't have pre crafted buffer overflow eggs.

If our system(s) are still in the potential candidate list there are some methods of preventing the exploits if circumstances prevent a BIND upgrade.

We can run BIND in a chroot jail. This gives BIND all the resources it needs to run as a non-root user and still have the ability to run the BIND service. The chroot jail thus protects the rest of the system's files from any exploit of the BIND service or the BIND service's non-root user.

We can use split DNS or split-split DNS. Split-split DNS separates DNS servers into scopes for internal users for internal systems, internal users for external systems, and external users for external (internal to the company, but names published externally to the Internet). While this is not total protection, it does offer some protection and enhances the security of the company's DNS service by allowing separate security policies for the separate DNS servers.

We can also configure BIND to lie about its version. Most attacks start with an attempt to fingerprint the DNS servers for versions known to contain the vulnerability the attacker has ready to launch at the just acquired target. Most attackers won't believe replies to version requests, but will do further fingerprinting or just launch the attack anyway and monitor the result.

Configuring the firewall an/or border router to restrict access to the BIND servers with the vulnerabilities. This assumes those devices can do state based UDP filtering and our DNS traffic patterns can be qualified to formulate firewall rules and / or router filters.

Men and Mice produce a survey indicating a quick response to the vulnerability alerts for these 4 vulnerabilities in BIND. For the survey done shortly after the alerts came out the survey showed 40.27% of .com BIND DNS servers were vulnerable. One week later the survey showed 16.73% were vulnerable. The latest survey shows only 10.3%. Temper this drop with the realization that these figures are on a random selection of 5500 servers from a potential population of over 21 million. Also temper this with the method of survey checking the version reported via DNS query, thus a site could have a patched version 4 BIND server running (no mention I can find on the survey indicates how such BIND servers are counted). Sites could also be running vulnerable V8 versions but configured them to respond with a non-vulnerable version to try to discourage attacks.

Why would a site do this? Sites I know that would be tempted to try such tactics are usually subject to severe change control. Getting management approval to implement a change in BIND servers is time consuming and probably difficult to sell. Some vendors BIND have vendor specific reasons for using the vendor supplied BIND. An example would be IBM's AIX for RS/6000s. BIND in this case is under the control of the System Resource Controller (SRC). IBM claims to fix all 4 vulnerabilities in APAR IY16182 for AIX Version 4.3.3. This version of AIX supplies both BIND 4 (reports version 4.9.3 before and after the APAR applied) and BIND 8 (reports 8.2.2.P5+Fix_for_CERT_till_01_30_01 after the APAR). The user chooses the version of BIND via symbolic link to named4 or named8.

Solaris uses BIND 4 for Solaris prior to version 7 and BIND 8.1.2 for Solaris 7 and 8.

Silicon Graphics IRIX uses BIND 4.9.7

Thus users using BIND from several of the major UNIX vendors and with barriers to upgrading the BIND version to 9 or one of the nonvulnerable versions would contribute to the count of the sites in the survey that indicate residual vulnerabilities.

Source code/ Pseudo code

It is interesting to note that exploits of the exploits are starting to appear.

The annakournikova.jpj.v_b_s virus spawned a BugTraq mail list message claiming to translate the code. The web site given is claimed to be infected with the VBS/Pica.worm.gen virus.

In the case of this CERT there is a trojan claiming to exploit the TSIG vulnerability. It actually attempts to attack dns1.nai.com with a well-disguised shellcode trick. This from an analysis from Max Vision at whitehats. It overflows its own buffer in the `set_ptr` function. That trojan is given in the example of malicious code as tsig_exploit.c

The Linux bind.c from LSD is included as bind.c

The first bind8x from Ix and LucySoft is included as bind8x.c

The fix to bind8x by Ian Goldberg and Johnathan Wilkins is included as fixed-bind8x.c

Gustavo Scotti and Thiago Zanino produced the code included as tsl_bind.c

Packetstorm published an exploit for TSIG that is included as bugtraq.c.

Securityfocus announced that it would start a malware archive in an announcement from Ryan Russell to contain the lion worm, adore, etc. As of the writing of this report, that malware archive has yet to appear.

Source for lion worm, adore worm, crew.tgz, and the erkms toolkit come and go on the Internet as sites are taken down. Such a malware archive would help in those instances.

Conclusion

Warnings form vulnerability alert services like CERT, SANS, SecurityPortal, etc. are valuable in getting news of such vulnerabilities to the Internet community. Once the alert has been received, research should be done to evaluate the vulnerability's affect on your site. Take care in evaluating your options in mitigating the vulnerability and DO NOT run codes touted to evaluate the vulnerability without investigating those as well. It took a few months for serious exploits of the 4 cited vulnerabilities against BIND to make their presence known, but they became very well known due to the nature of these exploits. Of the known exploits all are against Bind8 and the TSIG vulnerability with some help from the infoleak vulnerability. Most of those are against Linux with most of those against RedHat V6.2.

Bottom line, these exploits are most easily circumvented by upgrading to a non-vulnerable version of BIND.

Additional Information

The Internet Software Consortium's BIND page
http://www.isc.org/bind.html

CERT
http://www.cert.org/

COVERT Labs at PGP Security
http://www.pgp.com/covert

The DNS Resources Directory
http://www.dns.net/dnsrd/

The BIND configuration Guide
http://www.isc.org/bind8/config.html

Acme Byte & Wire's "Ask Mr. DNS"
http://www.acmebw.com/askmr.htm

Secure BIND Template
http://www.cymru.com/~robt/Docs/Articles/secure-bind-template.html

Solaris 7 chroot jail configuration
http://www.cymru.com/~robt/Docs/Articles/bindjail-solaris.html

FreeBSD chroot jail configuration
http://www.cymru.com/~robt/Docs/Articles/bindjail-freebsd.html

SECURITYPORTAL Article "Foiling DNS Attacks"
http://www.securityportal.com/cover/coverstory20001113.printerfriendly.html

"Buffer Overflows: Why, How, and Prevention", by Nicole LaRock Decker
http://www.sans.org/infosecFAQ/threats/buffer_overflow.htm

Last Updated: May 28th, 2017

Upcoming SANS Penetration Testing

SANS Madrid 2017	Madrid, Spain	May 29, 2017 - Jun 03, 2017	Live Event
SANS Stockholm 2017	Stockholm, Sweden	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
Mentor Session - SEC542	Santa Monica, CA	May 31, 2017 - Jul 12, 2017	Mentor
Community SANS Virginia Beach SEC504*	Virginia Beach, VA	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC542: Web App Penetration Testing and Ethical Hacking	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Rocky Mountain 2017 - SEC560: Network Penetration Testing and Ethical Hacking	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Milan 2017	Milan, Italy	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Secure Europe 2017	Amsterdam, Netherlands	Jun 12, 2017 - Jun 20, 2017	Live Event
Mentor Session - SEC504	Reston, VA	Jun 13, 2017 - Aug 01, 2017	Mentor
Community SANS Nashville SEC542	Nashville, TN	Jun 19, 2017 - Jun 24, 2017	Community SANS
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
Community SANS Albany SEC560	Albany, NY	Jun 19, 2017 - Jun 24, 2017	Community SANS
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
Community SANS New York SEC542	New York, NY	Jun 26, 2017 - Jul 01, 2017	Community SANS
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS ICS & Energy-Houston 2017	Houston, TX	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Omaha SEC560	Omaha, NE	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Seattle SEC504	Seattle, WA	Jul 10, 2017 - Jul 15, 2017	Community SANS
Mentor Session - SEC560	Augusta, GA	Jul 12, 2017 - Aug 23, 2017	Mentor